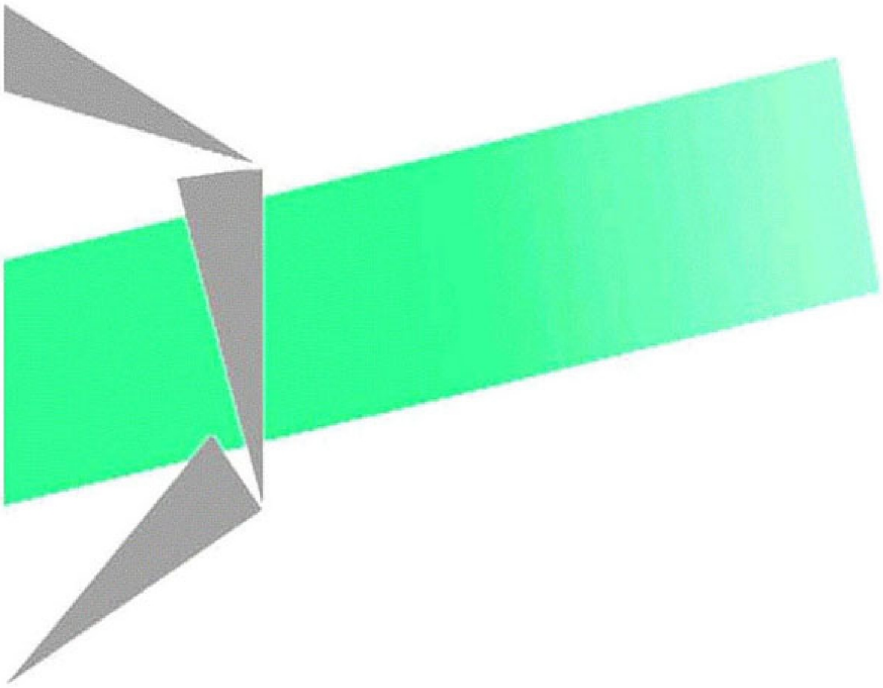
Les cahiers
Leibniz



MINIMIZING THE TOTAL DURATION OF THE WASHING STEP IN A HOSPITAL STERILIZATION SERVICE

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

Laboratoire G-SCOP
46 av. Félix Viallet, 38000 GRENOBLE, France
ISSN : 1298-020X

n° 188

October 2010

Site internet : http://www.g-scop.inpg.fr/CahiersLeibniz/

MINIMIZING THE TOTAL DURATION OF THE WASHING STEP IN A HOSPITAL STERILIZATION SERVICE

OZTURK Onur[(1)], ESPINOUSE Marie-Laure[(1)], DI MASCOLO Maria[(1)], GOUIN Alexia[(2)]
(1) Laboratoire G-SCOP (Grenoble-Science pour la Conception, l'Optimisation et la Production)
UMR5772, CNRS, Grenoble INP, UJF, Grenoble-France
46, avenue Félix Viallet - 38031 Grenoble Cedex 1 - France
{onur.ozturk; marie-laure.espinouse; maria.di-mascolo}@g-scop.grenoble-inp.fr
(2) GIPSA-lab (Laboratoire Grenoblois de l'Image, de la Parole, du Signal et de l'Automatique)
UMR 5216, CNRS, Grenoble INP, UJF, Grenoble-France
961, rue de la Houille Blanche, BP 46F - 38402 St. Martin D'Hères Cedex - France
alexia.gouin@gipsa-lab.grenoble-inp.fr

Abstract

In this paper, we study the problem of minimizing the total duration of the washing operations in hospital sterilization services. After use in operating blocs, reusable medical devices (RMD) are sent to the sterilization service, which is composed of various steps. In the washing step, different sets of RMD, used for different surgeries, may be washed together without exceeding washer capacity. It is generally not allowed to split RMD sets among several washers. We consider a batch scheduling problem where RMD sets are denoted as jobs having different sizes and different release dates, but equal processing times for the washing. We give optimal algorithms for a special case of the problem where job sizes form a strongly divisible sequence and for another case when job splitting is allowed, respectively. Afterwards, a mixed integer linear programming model is developed for our problem, and an approximation algorithm with worst case ratio of 2 is presented.

Keywords: OR in health services, hospital sterilization service, batch scheduling, mixed integer linear programming, approximation algorithm

1. Introduction

Hospital sterilization services aim at eliminating all infectious risks subject to the use of medical devices in surgeries. The most important objective is to prevent nosocomial infection. As a primordial operation, sterilization provides the reuse of medical devices in the next coming surgeries. After each use, sterilization guarantees the desired hygiene level of reusable medical

devices (RMD) for other uses in operating blocs. RMD can be defined as instruments used in surgeries that can be re-used. Beside the sterilization concern, in fact like all other sectors, hospitals are facing increased costs in their services, logistics or purchasing activities. Respecting hygiene conditions, increasing the number of reusable medical devices sterilized per day may help to cut costs about the purchasing of these equipments.

Sterile devices are designed for just one use, or for several uses. In case a sterile device is used for more than once, we speak of a reusable medical device. All RMD foreseen for a surgical operation must be sterilized. Sterilization process is regulated by some quality standards (see (AFNOR, 2005) for French quality standards of RMD sterilization).

The sterilization is a cyclic process (Fig.1) which is composed of several steps. Starting from the use in operating blocs, RMD are sent to sterilization service and pass the following steps: pre-disinfection, rinsing and washing, verification, packing, sterilization, storage and reuse in operating blocs.

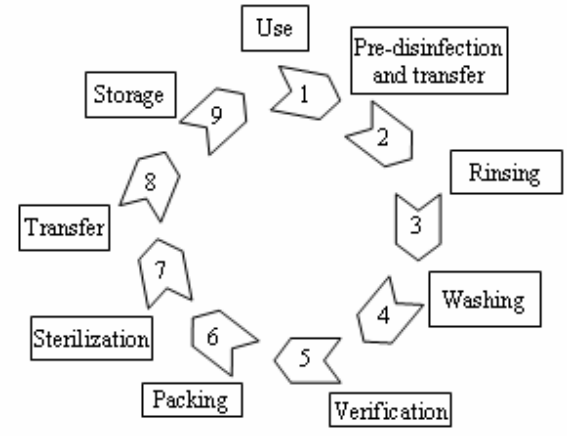


Figure 1. Sterilization Cycle

After use for a surgical operation, RMD are directly put in a substance, which enables pre-disinfection, and are transferred to the sterilization service. There, they are firstly rinsed and washed in washers. The rinsing is done either manually or automatically in washers. In our study, we consider that rinsing is carried out by washers. After washing, RMD are verified and packed into corresponding boxes. All items must be packed individually or grouped into boxes before sterilization. Afterwards, they are sterilized in machines which are called “autoclaves”, transferred to operating rooms and stored before reuse. Because the washing step is usually a

bottleneck over all the sterilization process (Albert *et al.* 2008; EESS, 2007), our aim is to minimize the total time spent for the washing operations.

The remainder of this paper is organized as follows. In section 2, we describe the problem of fulfilment of washers and show how this problem can be treated as a batch scheduling problem. In section 3, we give a literature review about batch scheduling problems. Sections 4 and 5 present optimal algorithms two different cases of the problem. In section 6, a MILP model and a 2-approximation algorithm are given. Section 7 is dedicated to the experimental design.

2. Problem description

2.1 Problem of fulfilment of washers

The scheduling problem investigated in this paper has its motivation from the washing step of a sterilization service. The number of different types of RMD is generally very high and for a typical hospital, there may be hundreds of RMD references. All RMD used for a surgical operation constitute the RMD set for this surgery. Because each surgery may require different numbers and types of RMD, sets may be of different sizes. For different reasons (surgery beginning times and durations, pre-disinfection procedure, etc.), RMD sets are ready for washing at different moments in a day. However, operating bloc scheduling helps to know the arrival times of RMD sets to the sterilization service. Hence, RMD set arrival times and sizes can be estimated in advance.

The washing of RMD sets is carried out by washers which can be identified with identical batching machines. It is possible to put more than one RMD set into a washer as long as its capacity is not exceeded. The decisions to make are then, which RMD sets to put together in order to constitute a batch for the washing, and when to launch a washing cycle. The characteristic of unequal RMD sets ready times for the washing complicates the decision of batching RMD sets. Note that in the washing step, RMD sets are not usually allowed to be split among several washers because of organizational and traceability reasons. In case of splitting, due to the multiplicity of RMD references, it takes a long time to reassemble the boxes of the RMD sets identically in the next steps. Moreover, splitting may cause some mistakes about the reassembling of RMD sets.

Due to great number of surgeries and capacity constraint of washers, the washing step is usually the bottleneck of the sterilization process (Albert *et al.* 2008; EESS, 2007). Hence, minimizing the duration of the washing step can help to increase the performance of the

sterilization process and also to increase the number of RMD sterilized per day. Another benefit of minimizing the duration of the washing step is that it can be possible to assign the washing operators to other posts as soon as the washing is completed thanks to the operator versatility.

In this paper, we define the problem of fulfilment of washers as a batch scheduling problem where RMD sets are denoted as jobs and washers as parallel batching machines. More formally, if we make a connection with scheduling problems, we are given a list of jobs $L = (j_1, j_2,\ldots, j_n)$, all of which have the same processing time p, but may have different release dates r_j, and different sizes w_j. We aim to schedule the jobs on identical parallel batch processing machines without pre-emption. A parallel batching machine is a machine that can simultaneously process more than one job as long as its capacity is not exceeded. Our aim is to explore opportunities for a better grouping of RMD sets for the washing in order to minimize the total completion time of washing operations.

2.2 Identification with a batch scheduling problem

To the best of our knowledge, Albert *et al*. (2008) are the first ones who study the problem of fulfilment of washers. In their work, they simulate different online fulfilment strategies (*i.e.* when the information about job sizes and release dates are not known in advance) for washers, like launching a washing cycle when a predetermined machine capacity is reached, or when RMD wait at most for a predetermined time, in order to minimize the number of launched washing cycles and the RMD waiting time before washing. Here, we model the problem of fulfilment of washers as a batch scheduling problem. So, in the following, RMD sets are denoted as jobs and washers as parallel batching machines. We make the following assumptions:

- There are n jobs to be processed. The release date and the size of job j are denoted by r_j and w_j, respectively. The processing times are equal for all jobs and denoted by p.
- All machines have the same capacity B and the size of a job cannot be greater than the machine capacity.
- Several jobs can be batched together respecting the machine capacity constraint.
- Once a processing for a batch is started, it cannot be interrupted.
- Since it is a parallel batching problem, the processing time of a batch is equal to the longest processing time of jobs in that batch (Potts and Kovalyov, 2000). As all the jobs in our problem 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 from the Graham's notation (Graham *et al*., 1979), we propose the following notation for our problem: $P \mid p\text{-}batch, r_j, p_j = p, w_j, B \mid C_{max}$. In this notation, P stands for identical parallel machines, *p-batch* for parallel batching, which means that several jobs may be executed together in a machine at the same time. r_j and w_j denote job release dates and sizes, respectively, $p_j=p$ stands for equal processing times and B for machine capacity. Finally C_{max} refers to the minimization of the total completion time, *i.e.* minimization of the makespan.

2.3 Problem complexity

Uzsoy (1994) studies the complexity of the problem $P \mid p\text{-}batch, p_j = p, w_j, B \mid C_{max}$. He shows that for equal job release dates and equal job processing times, the problem is NP-hard. As this special case of our problem is NP-hard, the problem we treat is also NP-hard.

3. Literature review

In the scheduling literature, batch scheduling problems may be divided into two groups: serial batching and parallel batching (Potts and Kovalyov, 2000). In the serial batching, jobs may be batched if they share the same setup on a machine and the processing time of a batch is equal to the sum of processing times of all jobs in that batch (*i.e.* one job is processed at a time) (Coffman *et al.*, 1990). In parallel batching, several jobs may be processed at the same time and the processing time of a batch is equal to the greatest processing time of jobs in that batch (Mathirajan and Sivakumar, 2006). Still, it is possible to divide parallel batching problems into groups according to job sizes or job families. For the classification according to job sizes, we can have two sub-groups: jobs requiring one unit of machine capacity (Lee *et al.*, 1992), or jobs having different capacity requirements (jobs having different sizes) (Uzsoy, 1994). Note that for this last one, if all release dates are equal, it can be possible to define the same problem as a bin-packing problem according to the criterion to optimize. If the problem is subject to job families, all the jobs in the same family have the same processing time, but may have different sizes and release dates (Potts and Kovalyov, 2000). If batches are limited to jobs from a single job family, incompatible job families are under consideration, else, compatible job families are considered (or a single job family). Clearly, our problem is a parallel batching problem with different job sizes and a single job family.

In parallel batching problems with different job sizes, the sum of job sizes that are put into a batch should not exceed machines capacity. Each job is assigned to just one batch. The processing time of a batch is given by the longest processing time of jobs that are put into the batch. In table 1, we give a brief classification of the literature dealing with parallel batch scheduling problems in presence of a single job family, different job sizes and different job processing times. We observe that most of the work focuses on makespan minimization.We can classify these papers into four groups as: 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. For the first group, Uzsoy (1994) studies the problem of minimizing the makespan and sum of job completion times. He proves that these two problems are strongly NP-hard. He provides several heuristic algorithms based on the first fit algorithm, which is one of the classical bin-packing algorithms, for the minimization of the makespan and a branch and bound algorithm for the minimization of the sum of job completion times. Ghazvini and Dupont (1998) develop several heuristics in order to minimize total flow time. Azizoglu and Webster (2000) consider also job weights which can be described as job importance. They give a branch and bound algorithm for the weighted sum of job completion times. Zhang *et al.* (2001) provide an approximation algorithm with a worst case ratio of 7/4 for the makespan minimization. They also analyse heuristics proposed by Uzsoy (1994) and compare them with their own solution method. Dupont and Dhaenens-Flipo (2002) propose an exact solution method with a branch-and-bound algorithm. They present dominance properties that can be used in an enumeration scheme. Kashan *et al.* (2006) propose two genetic algorithms for the problem. They report that their solution method outperforms the simulated annealing proposed by Melouk *et al.* (2004). For the second group, Kashan *et al.* (2008) report that the genetic algorithm they develop outperforms the simulated annealing given by Chang *et al.* (2004). It is clear that the third and fourth groups are more important for us since unequal release dates are considered. For the third group, Li *et al.* (2005) provide an approximation algorithm with a worst case ratio of $2+\varepsilon$ where ε can be arbitrarily small. Finally, for the last group, Chung *et al.* (2009) propose a MILP model and an heuristic approach. Their heuristic uses a first parameter in order to define a time horizon in which jobs are selected to be batched, and a second parameter to define the desired fullness ratio of batches. They experiment the heuristic with different values of these parameters. Damodaran *et al.* (2009) develop a “Greedy Randomized Adaptive Search

Table 1. The literature related to parallel batch scheduling problems with different job sizes and a single job family

Reference	Shop type	Release dates	Solution approach	Performance criterion
- Uzsoy(1994)	Single machine	Identical	Heuristic algorithms, B&B procedure	C_{max}, $\sum C_j$,
- Ghazvini and Dupont (1998)	Single machine	Identical	Heuristics	$\sum C_j$
- Azizoglu and Webster (2000)	Single machine	Identical	B&B procedure	$\sum w_j C_j$
- Zhang *et al.* (2001)	Single machine	Identical	Heuristics	C_{max}
- Dupont and Dhaenens-Flipo(2002)	Single machine	Identical	B&B procedure	C_{max}
- Melouk *et al.* (2004)	Single machine	Identical	MILP model, Simulated annealing	C_{max}
- Chang et al. (2004)	Parallel machines	Identical	Genetic algorithm	C_{max}
- Li *et al.* (2005)	Single machine	Different	Heuristic	C_{max}
- Kashan *et al.* (2006)	Single machine	Identical	Genetic algorithms	C_{max}
- Kashan *et al.* (2008)	Parallel machines	Identical	Genetic algorithm	C_{max}
- Chung *et al.* (2009)	Parallel machines	Different	MILP model, heuristics	C_{max}
- Damodaran *et al.* (2009)	Parallel machines	Different	Meta-heuristic	C_{max}
- Damodaran and Velez-Gallego (2009)	Parallel machines	Different	Heuristic	C_{max}

Objective: Cmax = total completion time (makespan), $\sum C_j$ = sum of job completion times, $\sum w_j C_j$ = weighted sum of job completion times

Procedure (GRASP)". They report that the GRASP approach guarantees the optimal solution for small instances and performs better than the heuristic proposed by Chung *et al.* (2009). Finally, Damodaran and Velez-Gallego (2009) propose a constructive heuristic. This heuristic operates by first determining a time horizon, and then it solves a 0-1 knapsack problem to select the jobs to be batched. They experiment the MILP model and heuristic given by Chung *et al.* (2009) and the GRASP approach developed by Damodaran *et al.* (2009). It is reported that their heuristic outperforms other heuristics in the literature and gives results close to those of the GRASP method. Note that our problem is a special case of the fourth group as we consider equal processing times for all jobs.

In this paper, we first provide an optimal algorithm for a special case of our problem where job sizes form a strongly divisible sequence. Afterwards, we give another optimal algorithm where job splitting is allowed. For the main problem, a MILP model and a 2-approximation algorithm are provided. We experiment and compare our MILP model and the 2-approximation algorithm to the MILP model proposed by Chung *et al.* (2009) and the heuristic given by Damodaran and Velez-Gallego (2009), respectively.

4. Job sizes forming a strongly divisible sequence

In this section, an optimal algorithm is given for a special case of our problem. We inspire from a special case of a bin-packing problem where job sizes form a strongly divisible sequence. In the standard one-dimensional bin packing problem, we are given a capacity B and a list of items $J = j_1, j_2, \ldots, j_n$, and are asked to partition the items into a minimum number of subsets such that the items in each subset sum to no more than B. Coffman *et al.* (1987) showed that if item sizes form a strongly divisible sequence, the first fit (FF) and the first fit decreasing (FFD) algorithms are optimal for the bin packing problem. Let us first remind these algorithms and the strongly divisible sequence.

Algorithm First Fit (Coffman *et al.* (1996))

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.

Algorithm First Fit Decreasing (Coffman *et al.* (1996))

This algorithm is the same as FF except that in step 1, jobs are sorted in non-increasing order of job sizes.

Strongly divisible sequence (Coffman *et al.* (1987))

Let W be a list of item sizes such that $w_1 > w_2 > \ldots > w_i > w_{i+1} > \ldots$. 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.

We consider a special case of our problem by such restriction that job sizes form a strongly divisible sequence. In application to washing operations, although there are a great number of different RMD set sizes, it is sometimes possible to approximate these sizes in order to associate to a strongly divisible sequence.

The Graham's notation can be modified as follows in order to represent the special case: $P \mid p\text{-}batch, r_j, p_j = p, w_j$ *(strongly divisible),* $B \mid C_{max}$. 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.

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

Property 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.

The proof of property 2 is evident. In b_1, if there are only jobs whose sizes are bigger than or equal to w_j, as these job sizes are multiples of w_j, then the batch b_1 is either completely filled or there is enough space to accommodate w_j in b_1.

Property 3. The aforesaid batch, b_1, in property 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 bigger than or equal to job j and group 2 for jobs smaller than job j.

These jobs can be arranged in non-increasing order of sizes. Then, the first group of jobs occupies a space which is an exact multiplication of w_j. Suppose that the batch b_1 is composed of sub-batches with 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 like applying FFD on these jobs where the batch capacity is equal to w_j. Thus, according to property 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 to accommodate job j completely in batch b_1 and b_1 becomes completely filled. The property 3 is going to be used in the algorithm in order to have fully complete batches.

Property 4. If all batches have the same processing time, then in the optimal solution, 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.

The algorithm we propose starts by creating a list, L_1, of jobs sorted in non-decreasing order of job release dates and a minimum number of batches is calculated using the first fit

algorithm. In each iteration, the first job of L_1 is put in a batch, and the number of batches to form with the remaining jobs is evaluated by applying the first fit algorithm. If there is a decrease in the total number of batches, the actual batch is closed. In case a job cannot be entirely put in a batch, then thanks to the property 3, some jobs are removed from the batch in order to fit this latest job in the batch. Finally, batches are sorted consecutively on machines in non-decreasing order of ready times. Let us now give an optimal algorithm that minimizes the makespan.

Algorithm SDS (SDS for strongly divisible sequence)

Step 1. Sort jobs in non-decreasing order of job release dates: L_1

Step 2. While L_1 is not empty, apply the "first fit" heuristic to L_1 in order to calculate the number of batches to form: n_k. Put the first job, j_{first}, of L_1 into a new batch, b_b. Update L_1 and re-apply the "first fit" heuristic to L_1 in order to calculate the new number of batches: n_l.

Step 2.1. While $n_k = n_l$, put the first job, j_{first}, of L_1 into the batch b_b. Update L_1 and re-apply the "first fit" heuristic to L_1 in order to calculate the number of batches: n_l.

Step 2.1.1. If the job, j_{first}, does not completely enter the batch, calculate the space needed: a_k to accommodate the job completely into the batch. Sort the jobs of the batch b_b in non-increasing order of sizes.

Step 2.1.1.1 While $a_k > 0$, remove the last job, j_{last}, from b_b. Set $a_k = a_k - w_{last.}$ Put j_{last} back into L_1 respecting the non-decreasing release date order of L_1.

Step 2.1.1.2 Set $n_l = n_l$ -1.

Step 3. Set ready time 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.

Each time a job is put in a batch, the new number of batches is calculated with the un-batched jobs by the first fit algorithm which has a time complexity of *0(nlogn)*. In the worst case, a number q of jobs is firstly placed in a batch then they all are removed from the batch for a large sized job. If q is sufficiently small according to the number of jobs, n, the number of times that q jobs are put into a batch and removed from that batch approaches to n. Then each time a job j, $1 \leq j \leq q$, is placed in a batch, first fit algorithm is implemented *0(nlogn)* times. Thus, the time complexity of the algorithm can be defined as $0(n^2 logn)$.

Theorem 1. Algorithm SDS is optimal for the problem:

$P \mid p\text{-}batch, r_j, p_j = p, w_j$ (strongly divisible), $B \mid$ minimizing the number of batches

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

Before showing the optimality of the algorithm for the makespan criterion, let us give another property about the minimum completion time after any job in a problem.

Property 5. For any problem with n jobs and M machines, the minimum number of batches, say nb, can be given by: $nb = \left\lceil \sum_{j=1}^{n} w_j / B \right\rceil$. Let jobs be sorted in non-decreasing order of release dates. Consider a job j such that $1 \leq j \leq n$ and $r_j \leq r_n$. After job j (including j), the minimum number of batches, say nb_j, can be calculated as: $nb_j = \left\lceil \sum_{k \in S} w_k / B \right\rceil$ where $S = \{k / r_k \geq r_j\}$. Then, the smallest completion time after job j can be given as: $r_j + \left\lceil \frac{nb_j}{M} \right\rceil * p$ where p is the processing time of batches.

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 for the minimum number of batches that can be created with jobs after j (including j). Batches are placed consecutively on machines. Hence, the division of nb_j by M is in order to find number of batches that will be placed on the same machine as job j.

Theorem 2. Algorithm SDS is optimal for the problem:

$P \mid p\text{-}batch, r_j, p_j = p, w_j$ *(strongly divisible),* $B \mid C_{max}$

Proof. Algorithm 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 in that batch, 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. We are going to show that the minimum completion time stated in property 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 starting time for batch b_m. Let us denote by s_{nb} the processing starting 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 starting time of the batch $b_{(nb-q*M)}$ *i.e.* $r_k = s_{(nb-q*M)}$. We have seen by property 5 that the minimum completion time after any job, say j, could be $r_j + \left\lceil \frac{nb_j}{M} \right\rceil * p$.

Thanks to the algorithm 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:

$$C^*_{\max} = r_k + \left\lceil \frac{nb_k}{M} \right\rceil * p \qquad \blacksquare$$

5. Job splitting is allowed

In the washing step of a sterilization service, it is generally not allowed to split RMD sets among several washers because of some organizational and traceability reasons. However, in some sterilization services, it is sometimes allowed to split an RMD set among two washers in order to complete the washing of this RMD set as soon as possible.

In this section, we give an optimal algorithm when job splitting is allowed. The proposed algorithm starts by creating a list 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. We start filling the batches with the first job of the previously created list. A batch is closed when it is completely filled or when there is 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. Let us express the problem by the following Graham's notation:
$P \mid p\text{-}batch, r_j, p_j = p, w_j\,(split), B \mid C_{max}$

Algorithm Split Job

Step 1. Sort jobs in non-increasing order of job release dates r_j : L_1

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_m

Step 3.1. Put the first job of L_1 into the batch b_m. Update L_1. If the batch is 100% filled or if there is no more elements in L_1, 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 time 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.

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)*.

Theorem 3. Algorithm Split Job is optimal for the problem:

$P \mid p\text{-}batch, r_j, p_j = p, w_j\,(split), B \mid C_{max}$

Proof. We can give a proof in line with the theorem 2 in order to show that the algorithm split job finds the optimal makespan. In the second step of the algorithm, a lower bound on the number of batches is calculated. The condition to close a batch is that it should be entirely filled or there should be no more jobs to place in the batch. This implies that all batches are necessarily 100% full, except for the batch having the smallest ready time.

Now, as we did in proof 2, consider a chain of batches without idle times on the machine that contains the last batch (*i.e.* the machine that sets the makespan). Then, the ready time of the first batch in this chain is necessarily equal to the release date of a job, let us say job *k*. As the rest of the batches are 100% full, after job *k* a minimum number of batches are formed. Moreover, as the batches are placed consecutively on machines, the number of batches after job *k* and on the same machine as *k* is also minimal. Hence, the minimum completion time expressed in property 5 is reached and the makespan found by the algorithm is optimal.

■

6. Solution approaches for the problem $P \mid p\text{-}batch, r_j, p_j = p, w_j, B \mid C_{max}$

In this section, we first define how a lower bound can be obtained for the problem. Then, we explain the MILP model and a 2-approximation algorithm that uses the lower bound procedure.

6.1 Lower bound for the problem

In section 5, 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, thus the minimum completion time was obtained. Then, the solution found by the algorithm with job splitting allowed can be treated as a lower bound algorithm for the case when job splitting is not allowed. Therefore, we use the Split Job algorithm as a lower bound algorithm for the problem: $P \mid p\text{-}batch, r_j, p_j = p, w_j, B \mid C_{max}$.

6.2. A mixed integer linear programming model (MILP model)

Index:

j: $1,\ldots,N$ for jobs

k: $1,\ldots,N$ for batches

m: $1,\ldots,M$ for machines

Parameters:

w_j: size of job j

r_j: release date of job j

N: number of jobs

B: machine capacity

p: job processing times

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}: completion time

Mathematical formulation:

$$Minimize \quad C_{\max} \tag{0}$$

subject to

$$\sum_{k=1}^{N}\sum_{m=1}^{M} x_{jkm} = 1 \qquad \forall j \in [1; N] \tag{1}$$

$$\sum_{j=1}^{N} w_j * x_{jkm} \le B * b_{km} \qquad \forall k \in [1; N]; \forall m \in [1; M] \tag{2}$$

$$\sum_{m=1}^{M} b_{km} \le 1 \qquad \forall k \in [1; N] \tag{3}$$

$$S_{km} \ge x_{jkm} * r_j \qquad \forall j \in [1; N]; \forall k \in [1; N]; \forall m \in [1; M] \tag{4}$$

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

$$C_{\max} \ge S_{N,m} + p * b_{N,m} \qquad \forall m \in [1; M] \tag{6}$$

$$b_{k;(k \bmod M)+1} \ge 1 \qquad \forall k \in [1; nb] \tag{7}$$

$$b_{k;m'} = 0 \qquad \forall k \in [nb+1; N], \forall m' \neq (k \bmod M) + 1 \tag{8}$$

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

Our objective is to minimize the makespan. At most *N* batches can be created in a schedule. So the index of batches vary from 1 to *N*. Constraint (1) ensures the assignment of all jobs to a batch and to a machine. Constraint (2) is the capacity constraint in case batch *k* is created on machine *m*. Constraint (3) assigns a batch at most on one machine. Constraint (4) sets the ready time of a batch as the greatest release date of jobs in that batch. In case more than one batch is assigned to a machine, (5) ensures a difference at least equal to the execution duration *p*, between the processing of these batches. Another functionality of constraint (5) is to assign a ready time to all batches, 1 to *N*, even if they are not created, *i.e.* dummy batches get also a ready time with (5). If $b_{k,m}$ is null, then the batch *k* on the machine *m* is a dummy batch and its ready time is directly given to the next indexed batch on machine *m*. Constraint (6) defines C_{max}. Constraints (7) and (8) aim at improving the performance of the MILP model by decreasing the multiple solutions due to the batch assignments. As all batch processing times are equal, without loss of generality we schedule batches consecutively on machines. The minimum number of batches is expressed by *nb*, so, at least *nb* batches should be created. By constraint (7), the first *nb* batches are placed consecutively on machines. *"k* mod *M"* determines the machine on which batch *k* will be executed. We add "1" to "(*k* mod *M*)" in order to prevent from having 0 as a

machine index and without loss of generality, we place the first batch on machine number 2.) Reasoning the same way as constraint (7), constraint (8), enables to place the rest of the batches, indexed from $nb+1$ to N, consecutively on machines. We do not know if the binary variables of these batches, namely $b_{k;(k \bmod M)+1}$, equal to 0 or 1. But as any batch can be assigned at most to one machine, we can force the complement binary variables of $b_{k;(k \bmod M)+1}$ (*i.e.* binary variables $b_{km'}$ where $m' \neq (k \bmod M)+1$) to be 0 as constraint (8) does. Note that this constraint is specific and is just for more than one machine cases. The model contains $N^2M+2NM+1$ variables and the number of constraints is $N^2M+3NM+N-nbM+2nb$.

6.3. A 2-approximation algorithm

Now we can start to construct an approximation algorithm for our problem. 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 in a batch, the job release date must be smaller than the ready time of the batch). Afterwards, we search machine availabilities, *i.e.* the periods in which a machine is idle. If there is machine availability greater than the processing time, p, between two batches, the algorithm searches the jobs whose release dates are up to p units of time earlier than the end of the time interval. Then, a new batch is opened to be filled with these jobs. If the batch can be executed in the time interval without modifying the processing starting time of other batches, then the batch is created. Finally, all un-batched jobs are batched using the "first fit decreasing" algorithm and scheduled after the previously formed batches.

Algorithm Combine Job

Step 1. Apply the "Split Job" algorithm.

Step 2. Remove split jobs from the batches created in step 1.

Step 3. Create a list, L_1, of jobs in non-increasing order of sizes with jobs found in step2.

Step 4. For all the batches created in step 1, starting from the batch with the smallest ready time and the first element of L_1, if the release date of the job is smaller than the ready time of the batch and if there is enough space for the job in that batch, place the job in the batch. Update the size of the batch and erase the job from L_1.

Step 5. For all machines, search for the time intervals between batches during which a machine is idle for more time than the processing time, *p*. If there is no such time interval, go to step 7.
Step 6. For all time intervals, search the un-batched jobs whose release dates are up to *p* units of time earlier than the end of the time interval and create a list, L_2, containing these jobs in non-decreasing order of release dates. If L_2 is not empty, open a batch for the corresponding time interval. While the batch can be processed in the time interval without modifying the processing starting times of other batches on the same machine, apply first fit algorithm on L_2 in order to fill the batch.
Step 7. For all the un-batched jobs, apply the first fit decreasing algorithm to create batches and schedule them consecutively on machines after the previously formed batches.

The time complexity of the algorithm depends on the step 6. For a problem with *n* jobs, the upper bound for the number of batches is *n* and hence, the maximum number of time intervals is *n*-1. The maximum number of un-batched jobs is equal to the number of split jobs which, also is at most; *n*-1,. Thus, for each time interval, *n*-1 jobs are scanned at most. Then, all jobs are sorted in non-decreasing order of release dates and the first fit algorithm is applied on *n*-1 jobs which leads to a total complexity of $0(n^3 logn)$.
Theorem 5. Algorithm Combine Job is a 2-approximation algorithm for the problem
$P \mid p\text{-}batch, r_j, p_j = p, w_j, B \mid C_{max}$
Proof. Let us denote by $C_{\max}^{LB}$ the makespan found by the split job algorithm (which is in fact the lower bound of the problem) and $C_{\max}^{*}$ the optimal solution. It is obvious that the relation between the optimal solution and the one found by the lower bound is $C_{\max}^{LB} \leq C_{\max}^{*}$. Let $C_{\max}$ be the completion time found by the algorithm combine job. Note that the algorithm finds firstly 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 either scheduled in the time intervals where the machines are idle more than the execution time, *p*, or, if there is no machine availability, the jobs are batched by FFD and scheduled after the batches formed by the lower bound 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. We can still suppose that in the worst case

scenario after removing the split jobs, there is no machine idle times more than a processing time between the batches (batches that are created and scheduled by the lower bound algorithm in the step 1), and the sizes of the previously split jobs are so large that, they can neither be inserted in any existing batch nor batched together. Hence, we are to schedule these *nb-1* jobs one by one after the batches created in step 1.

Let us suppose that there are *M* machines, and in the lower bound solution each machine contains nb_m batches, where $1 \le m \le M$. Then, $\max_{m=1}^{M} \{ nb_m \} * p \le C_{\max}^{LB}$. 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: $C_{\max} \le C_{\max}^{LB} + \max_{m=1}^{M} \{ nb_m \} * p$

Moreover, $C_{\max}^{LB} + \max_{m=1}^{M} \{ nb_m \} * p \le 2 C_{\max}^{LB} \le 2 C_{\max}^{*}$

Thus, we get, $C_{\max} \le 2 C_{\max}^{*}$ ■

7. Experimental design

In this section, we test the effectiveness of the proposed MILP model and the 2-approximation algorithm. Because we consider equal processing times, the problem we treat in this paper is a special case of the problem treated by Chung *et al.* (2009) and Damodaran and Velez-Gallego (2009). We solve our MILP model with the commercial program CPLEX 10.2 and compare it to the MILP model proposed by Chung *et al.* (2009). As reported in Damodaran and Velez-Gallego (2009), their heuristic outperforms other heuristics in the literature. So, we compare the 2-approximation algorithm to their heuristic.

The instances we test are inspired from a real case. The data given by a private French hospital are used to create these instances. More precisely, the machine capacity, batch processing time, job sizes and release dates are inspired from the real case. The machine capacity is assumed to be 6 and the processing time is 60. In the real case, there are nearly 36 different RMD set sizes and these sizes are assumed to be multiple of one thirty sixth of the machine capacity. We observed the frequencies of different RMD set sizes in 5 days data and created different job sizes respecting the proportionality of these frequencies. For the release dates, it is

observed that there are on average 5 RMD set arrivals to the sterilization service per hour and there may be 0 to 40 minutes of difference between different RMD set arrivals. Besides, in some sterilization services, there is a regular collecting of RMD sets among operating blocs. Hence, we define 2 more different types of job release dates. We consider 20 minutes and 40 minutes of regular job release dates. Note that there may be more than one job released at the same. .

We group our experiments into sets according to the number of jobs and the number of machines. The number of machines varies from 1 to 4, while numbers of jobs are 10, 15, 20, 25, 30 and 50. For each job number/machine number combination, we test 90 different instances. Thus, for each different release date type, 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. The solution approaches are coded in C++ language, and CPLEX version 10.2 is used to implement the MILP models.

7.1 Performance of the proposed MILP model

In table 2, we show the average resolution times and the proportion of the optimally solved instances in 3600 seconds. The processing time of batches is equal to one hour and so we fixed a resolution time limit of 3600 seconds for the implementation of the MILP models. 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 in order to calculate the batch processing times. As all job/batch processing times are equal in our problem, we could remove this constraint from their MILP model.

For the two MILP models, if 4 machines are considered, the resolution limit is 20 jobs. However for 20 jobs, our MILP model can also resolve instances with 1, 2 and 3 machines while the other MILP model cannot reach the optimal solution in the desired time limit. Moreover, our MILP model can still go beyond 20 jobs/4 machines and solve until 25 jobs/2 machines instances. Beside, the resolution limits, we see from table 2 that the resolution times are faster with our MILP model.

7.2 Quality of the lower bound algorithm compared with optimally solved instances

In section 6.1, we showed that the optimal algorithm for the case with the job splitting could be used as a lower bound algorithm for the general problem. Therefore, we can test the

efficiency of the proposed lower bound algorithm with respect to the optimally solved instances. Therefore, in the following table, we show the average difference between the solutions found by the lower bound algorithm and the optimal solutions. The reported gap is calculated by $(C_{max}^{*} - C_{max}^{LB})*100 / C_{max}^{LB}$ where C_{max}^{LB} is the makespan found by the lower bound algorithm and C_{max}^{*} is the optimal makespan.

Table 2. Resolution limits for the MILP models

		Proposed MILP		*MILP of Chung et al. (2009)*	
Number of jobs	*Number of machines*	*Resolution time on avg.*	*% of optimally solved instances*	*Resolution time on avg.*	*% of optimally solved instances*
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 sec.	100%	>3600 sec.	0%
20	3	≈450 sec.	100%	>3600 sec.	0%
20	4	≈341 sec.	100%	≈1349 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

Number of jobs	*Number of machines*	*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	≈ 9.13 %
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 %

In table 3, we show the average gap between the lower bound algorithm and optimal solutions in terms of solution quality. We tested 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, especially for 2, 3 and 4 machine instances.

7.3 Performance of the 2-approximation algorithm

In this section, we test the efficiency of the 2-approximation algorithm. It is reported in Damodaran and Velez-Gallego (2009) that their heuristic, namely PSKP (progressive successive knapsack heuristic), outperforms other heuristics for the problem: $P \mid p\text{-}batch, r_j, p_j, w_j, B \mid C_{max}$. The heuristic they propose sets first a time window which is defined by [0; r_k] where r_k is the k^{th} earliest job release date among the un-batched jobs. For jobs in that interval of time, they solve a 0-1 knapsack problem using the dynamic algorithm proposed by Martello and Toth (1990). They execute the heuristic for varying values of the parameter k from 1 to n where n is the total number of jobs. Finally, the best makespan value is chosen. In this part, we compare the 2-approximation algorithm to the PSKP heuristic.

We test the efficiency of the 2-approximation algorithm and the heuristic of Damodaran and Velez-Gallego (2009) in terms of solution quality. With this purpose, the average gap is calculated by the following formula: $(C_{\max}^{sol} - C_{\max}^{*}) * 100 / C_{\max}^{*}$ where $C_{\max}^{*}$ is the optimal makespan and $C_{\max}^{sol}$ is the makespan found by the other solution methods. Note that for the instances which are not optimally solved, we make a comparison to the lower bound value. Therefore, for instances which are not optimally solved, $C_{\max}^{*}$ presents the value of the lower bound.

Figure 2 shows for all numbers of jobs the average performance of the 2-approximation algorithm and the PSKP heuristic for different machine numbers. We see that the solution quality found by the PSKP heuristic is much better than the performance of the 2-approximation algorithm for small numbers of machines. However, with the increasing machine number, the performance of the 2-approximation algorithm increases considerably. Especially, for 4 machine cases, the performance of the 2-approximation algorithm is quite well.

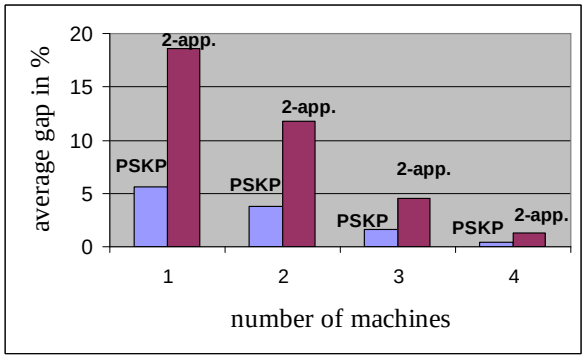


Figure 2. Performance of the PSKP heuristic and the 2-approximation algorithm in terms of solution quality

Although the performance of the PSKP heuristic proposed by Damodaran and Velez-Gallego (2009) gives better results in terms of solution quality, it requires excessive computation time for our problem (figure 3). The PSKP uses a 0-1 knapsack problem dynamic programming resolution procedure which is pseudo polynomial in time. Moreover, Damodaran and Velez-Gallego (2009) consider integer job sizes. Since we consider fractional job sizes, we are to multiply the job sizes and the machine capacity with a proper "product factor" in order to have integer values for the 0-1 knapsack resolution procedure. As explained before, job sizes are a multiple of one thirty sixth of the machine capacity. We considered 2 digits after decimal comma while creating job sizes. This fact raises 100 as the "product factor" in order to have integer values for the job sizes. In figure 3, we give the average resolution times for the implementation of the PSKP heuristic. Note that the resolution for the 2-approximation algorithm is some milliseconds for all instances.

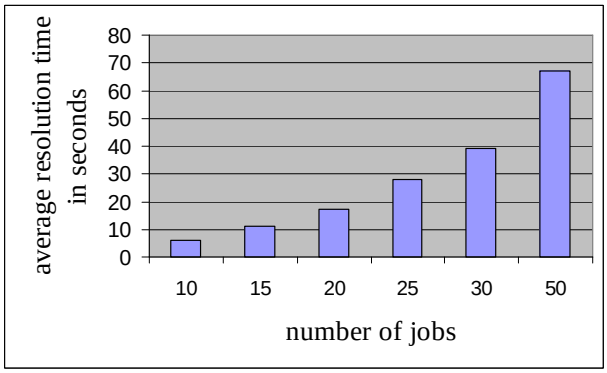


Figure 3. Average resolution times with the PSKP heuristic for different numbers of machines

The main reason for the poor performance for the 2-approximation algorithm for small numbers of machines is that, after the first step of the algorithm, generally every batch contains split jobs. Therefore, there is a big amount of jobs which are to be batched after the fist step. Moreover, it is not evident that placing these jobs in any already existing batches is possible. Hence, initially split jobs are generally to be batched apart which causes a big number of batches in total. However, with the increasing number of machines, the effect of the number of batches on the makespan decreases and thus, the more machines we have, the closer we get to the optimal solution with the 2-approximation algorithm.

8. Conclusion and further issues

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 at minimizing the total duration time for the washing step. Besides, completing the washing operations as soon as possible also helps to profit from the versatility of washing operators by assigning them to other posts.

The batch scheduling problem we tackled has the following specifications: parallel batching machines which can execute several jobs at the same time, job release dates, job sizes, limited machine capacity and equal job processing times. If unequal job processing times are considered, our problem becomes a special case for this new problem. At first, two optimal algorithms are provided, first for a special case with strongly divisible job sizes, and then for another case where job splitting is allowed. For the main problem, we developed a MILP model and compared it to the MILP model developed by Chung *et al.* (2009). Our MILP model outperformed the other model in terms of resolution time. Afterwards, a 2-approximation algorithm is provided and compared to the PSKP heuristic proposed by Damodaran and Velez-Gallego (2009). Clearly, PSKP heuristic performs better than the 2-approximation algorithm for small numbers of machines. But, for a 4 machine case, 2-approximation algorithm performs quite well. Further, running time of the 2-approximation algorithm is some milliseconds for all tested instances while the running time of the PSKP heuristic is considerably longer.

For future work, the effect of washing step optimization on the whole sterilization service may be studied. Furthermore, other performance criteria may be considered for the washing step

optimization such as minimizing the waiting of RMD sets before washing or minimizing the sum of washing completion times.

References

AFNOR, 2005, Standard AFNOR FD S98-135. Stérilisation des dispositifs médicaux, Guide pour la maîtrise des traitements appliqués aux dispositifs médicaux réutilisables

Albert F, Di Mascolo M, Marcon E. Analyse de différentes stratégies de remplissage de laveurs dans un service de stérilisation de dispositifs médicaux, 7ème Conférence de Simulation et Modélisation MOSIM'2008, Paris, France

Azizoglu M, Webster S. Scheduling a batch processing machine with non-identical job sizes, International Journal of Production Research 2000; 38; 2173-2184.

Chang PY, Damodaran P, Melouk S. Minimizing makespan on parallel batch processing machines. International Journal of Production Research 2004; 42; 4211–4220.

Chung SH, Tai YT, Pearn WL. Minimising makespan on parallel batch processing machines with non-identical ready time and arbitrary job sizes. International Journal of Production Research 2009; 47; 5109-5128.

Coffman Jr. EG, Garey MR, Johnson DS. Bin packing with divisible item sizes. Journal of Complexity 1987; 3(4), 406-428

Coffman Jr. EG, Yannakakis M, Magazine MJ, Santos C. Batch sizing and job sequencing on a single machine. Annals of Operations Research 1990; 26; 135-147

Coffman Jr. EG, Garey MR, Johnson DS. Approximation algorithms for bin packing: a survey. In: D. Hochbaum (ed.), Approximation algorithms for NP-hard problems, PWS Publishing, Boston; 1996, p. 46-93

Damodaran P, Velez-Gallego MC, Maya J. A GRASP approach for makespan minimization on parallel batch processing machines. Journal of Intelligent Manufacturing 2009. doi: 10.1007/s10845-009-0272-z

Damodaran P, Velez-Gallego MC. Heuristics for makespan minimization on parallel batch processing machines with unequal job ready times. The International Journal of Advanced Manufacturing Technology 2009. doi: 10.1007/s00170-009-2457-1

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

EESS, Enquête Electronique sur les Services de Stérilisation. 2007 (http://www.laspi.fr/ipi)

Ghazvini FJ, Dupont L. Minimizing mean flow time criteria on a single batch processing machine with non-identical job sizes, International Journal of Production Economics 1998; 55(3); 273-80.

Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics 1979; 5; 287-326.

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

Kashan AH, Karimi B, Jenabi M. A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. Computers & Operations Research 2008; 35; 1084–1098.

Lee CY, Uzsoy R, Martin-Vega LA. Efficient Algorithms for Scheduling Semiconductor Burn-In Operations. Operations Research 1992; 40(4); 764-775.

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

Martello S, Toth P. Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Chichester–New York, 1990

Mathirajan M, Sivakumar AI. A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. The International Journal of Advanced Manufacturing Technology 2006; 29; 990-1001.

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

Potts CN, Kovalyov MY. Scheduling with batching: a review. European Journal of Operational Research 2000; 120(2); 228-249

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

Zhang G, Cai X, Lee CY, Wong CK., Minimizing makespan on a single batch processing machine with nonidentical job sizes. Naval Research Logistics 2001; 48; 226–240.

Les cahiers Leibniz ont pour vocation la diffusion des rapports de recherche, des séminaires ou des projets de publication sur des problèmes liés au mathématiques discrètes.

Responsables de la publication : Nadia Brauner et András Sebő
ISSN : 1298-020X - © Laboratoire G-SCOP