Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure

Lionel Dupont[a,*], Clarisse Dhaenens-Flipo[b]

[a]*GILCO – 46, Avenue Félix Viallet, 38031 Grenoble Cedex, France*
[b]*LIFL – Batiment M3, Cité scientifique, 59655 Villeneuve d'Ascq cedex, France*

Abstract

A batch processing machine can simultaneously process several jobs forming a batch. This paper considers the problem of scheduling jobs with non-identical capacity requirements, on a single-batch processing machine of a given capacity, to minimize the makespan. The processing time of a batch is equal to the largest processing time of any job in the batch. We present some dominance properties for a general enumeration scheme and for the makespan criterion, and provide a branch and bound method. For large-scale problems, we use this enumeration scheme as a heuristic method.

Scope and purpose

Usually in classical scheduling problems, a machine can perform only one job at a time. Although, one can find machines that can process several jobs simultaneously as a batch. All jobs of a same batch have common starting and ending times. Batch processing machines are encountered in many different environments, such as burn-in operations in semiconductor industries or heat treatment operations in metalworking industries. In the first case, the capacity of the machine is defined by the number of jobs it can hold. In the second case, each job has a certain capacity requirement and the total size of a batch cannot exceed the capacity of the machine. Hence, the number of jobs contained in each batch may be different. In this paper, we consider this second case (which is more difficult) and we provide an exact method for the makespan criterion (minimizing the last ending time). © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Batching machine; Non-identical jobs; Branch-and-bound

* Corresponding author. Tel.: + 33-4-76-57-48-44; fax: + 33-4-76-57-47-93.
E-mail addresses: lionel.dupont@gilco.inpg.fr (L. Dupont), clarisse.dhaenens@lifl.fr (C. Dhaenens-Flipo).

1. Presentation

We consider the problem of minimizing the total completion time on a single-batch processing machine. A batch processing machine (BPM) or batching machine is a machine that can process several jobs simultaneously, as a batch, with common starting and ending times. Many manufacturing systems, such as those encountered in the semiconductor and metalworking industries, contain batch processors. In the semiconductor industry, products (integrated circuit) of different characteristics can be processed simultaneously, in the same batch, without restriction. In the metalworking industry, where steel coils have to be heated, for example, technical constraints may require grouping similar products in the same furnace. Therefore, several families of products are defined, according to the two principal characteristics: size and heating time. A batch can only be composed of products of a same family, i.e. products with similar characteristics.

In this paper we make the following assumptions:

(1) A set $N = \{1, \ldots, n\}$ of jobs have to be processed. The minimum processing time required for job i is denoted by p_i. Each job i is available at time $r_i = 0$.
(2) The machine has a capacity B and each job has a size s_i. The sum of the sizes of the jobs contained in a batch cannot exceed B. We assume that the size of a job cannot exceed the machine capacity, i.e., $s_i \leqslant B$ for all job i.
(3) Once the processing of a batch is started, it cannot be interrupted and other jobs cannot be introduced into the machine until the processing is completed. The processing time of a batch is given by the largest processing time of jobs contained in the batch.
(4) The objective is to minimize the makespan, that is the maximum completion time of jobs, $C_{\max}$.

The organization of this paper is as follows. In Section 2, we review previous related work on batch processing models. In Section 3, we present some previous results related to the problem under study. In Section 4, we develop dominance properties of optimal schedules for general regular criteria and for $C_{\max}$. We present a branch and bound method to find an optimal solution in Section 5. In Section 6, we report the results of the computational experiments. We conclude the paper with a summary in Section 7.

2. Related works

Problems of scheduling BPM have been recently studied in the literature. First, we review works that address problems with identical job sizes, where B is the maximum number of jobs that can be processed simultaneously. Given identical job processing times, Ikura and Gimple [1] and Lee et al. [2] examine problems with agreeable release times and due dates (i.e. $r_i \leqslant r_j$ implies $d_i \leqslant d_j$) for several criteria. Li and Lee [3] show that these problems are both strongly NP-hard if release times and due dates are not agreeable. Ahmadi et al. [4] present the only paper where a system of two machines, with at least one BPM, is modeled.

Brucker et al. [5] give several exact algorithms and complexity results for the special case $n \leqslant B$. When $n \geqslant B$, they show that problems of minimizing $L_{\max}, \sum U_i$, and the total tardiness, $(\sum T_i)$, are strongly NP-hard and provide an $O(n^{B(B-1)})$ exact algorithm for minimizing $\sum C_i$. For this problem, Chandru et al. [6] and Dupont and Jolai Ghazvini [7] provide branch and bound

methods, that are able to solve problems with up to 35 and 100 jobs, respectively. The weighted case, $\sum w_i C_i$, has been addressed by Uzsoy and Yang [8]. Lee et al. [2] give efficient algorithms for minimizing $L_{\max}$ and $\sum U_i$ when processing times and due dates are agreeable. Lee and Uzsoy [9] study the problem of minimizing $C_{\max}$ with dynamic job arrivals. Scheduling a single BPM with a secondary resource constraints has been considered by Kempf et al. [10].

Scheduling BPMs with incompatible job families and identical job sizes has been widely studied. Uzsoy [11] develops efficient algorithms for minimizing $L_{\max}$ and $\sum w_i C_i$ on a single BPM with incompatible job families. He also considers the problems of minimizing $C_{\max}$ and $L_{\max}$ on a single BPM with dynamic job arrivals. Mehta and Uzsoy [12] provide exact and heuristic algorithms for minimizing $\sum T_i$. The problem of minimizing $\sum C_i$ on a BPM with compatible job families has been considered by Chandru et al. [13], Hochbaum and Landy [14] and Brucker et al. [5]. They have presented dynamic programming algorithms with complexity time of order $O(m^3 B^{m+1})$, $O(3^m m^2)$ and $O(B^2 2^m m^2)$, respectively. Kubiak and Jolai Ghazvini [15] consider the problem of minimizing earliness/tardiness criteria on a BPM for job families.

Relatively few works have studied problems with non-identical job sizes. Dobson and Nambinadom [16] provide a mathematical formulation and some heuristics for the problem of minimizing $\sum w_i C_i$ with incompatible job families. Uzsoy [17] examines the $\sum C_i$ and the $C_{\max}$ criteria and provides branch and bound algorithms and heuristics. Jolai Ghazvini and Dupont [18] present two new heuristics to minimize $\sum C_i$.

3. Previous results

In this section, we present the results provided by Uzsoy [17] and Dupont and Jolai Ghazvini [19] in their study of the problem of minimizing the makespan on a single-batch processing machine with non-identical job sizes.

Complexity results. When each job requires identical processing time, minimizing $C_{\max}$ is equivalent to a bin-packing problem with a bin capacity of B and item sizes equal to s_i. As the bin-packing problem is strongly NP-hard, the problem of minimizing makespan on a single-batch processing machine with non-identical jobs is also strongly NP-hard.

Lower bound. A lower bound on the optimal $C_{\max}$ can be calculated by relaxing the problem and allowing jobs to be split and processed in different batches. This is done by constructing an instance of $C_{\max}$ where each job i, of the original problem, is replaced with s_i jobs of unit size and processing time p_i. This can easily be solved by sorting jobs in the LPT order (decreasing order of their processing time), successively grouping the B non-added jobs with longest processing times into the same batch, and then processing batches in any order.

Heuristic. Uzsoy [17] uses the First-Fit heuristic, developed for the bin-packing problem [20], to construct a set of heuristics. Among them, he shows that the following algorithm, named FFLPT, has the best performance on average and in the worse-case.

Algorithm FFLPT (First-Fit Longest Processing Time)
Step 1. Arrange the jobs in decreasing order of their processing times p_i (LPT).
Step 2. Select the job at the top of the list and place it in the first batch with enough space to accommodate it. If the job fits in no existing batch, create a new batch. Repeat Step 2 until all jobs have been assigned to a batch.

Step 3. Sequence the batches on the machine in any arbitrary order.

Dupont and Jolai Ghazvini [19] present two other heuristics. The first one is based on the Best-Fit algorithm, developed for the bin-packing problem, and can be stated as follows:

Algorithm BFLPT (Best-Fit Longest Processing Time)

Step 1. Arrange the jobs in decreasing order of their processing times p_i.

Step 2. Select the job at the top of the list and place it in the feasible batch with the smallest residual capacity (breaking ties in favor of the lowest indexed batch). If the job fits in no existing batch, create a new batch. Repeat Step 2 until all jobs have been assigned to a batch.

Step 3. Sequence the batches on the machine in any arbitrary order.

This heuristic attempts to address both the need for minimizing the number of batches and that for minimizing the residual capacity of batches.

The second heuristic (SKP: Successive Knapsack Problem) constructs a schedule batch per batch. The first job f in the current batch is assumed to be the job with the longest processing time among non-scheduled jobs. The batch is represented as a rectangle of dimension (p_f, B) and an unscheduled job i to a rectangular piece of dimension (p_i, s_i). The objective is to maximize the occupied surface. To determine the other jobs in the current batch, the algorithm solves a 0–1 knapsack problem defined by:

$$\max \sum_i (s_i.p_i).x_i$$
$$\text{s.t.} \sum_i s_i.x_i \leqslant B - s_f.$$

The proposed heuristic attempts to take both processing times and job sizes into account while developing a schedule.

Over the set of instances solved, BFLPT and SKP dominate FFLPT and SKP dominates BFLPT. However, the difference is very small and SKP needs more computational time requirements.

4. Enumeration scheme

In this section, we present enumeration schemes for different problems: the general problem, problems with regular criteria and problems with the $C_{\max}$ criterion. All enumeration schemes follow the same main framework. They construct the schedule batch per batch until every job has been added. We will call the current batch, the batch to which we are currently adding jobs, CB. We will note J the set of non-scheduled jobs. At each step st, one job of J is added to the partial schedule. In order to build the enumeration tree, two decisions have to be made at each step:

1. Whether the job will be added to the current batch CB or to the next batch $CB + 1$, which will become the current batch. In the latter case, the new job will be the first job of this new batch.

2. Which unscheduled job has to be added to the selected batch.

4.1. Generic enumeration

For the rest of this paper, we shall assume jobs to be indexed in non-increasing order of their processing times p_i (LPT order).

Remark 1. All the jobs of a given batch are processed simultaneously, and have the same starting and ending time. The order in which those jobs are added to the batch has no influence on the value of the solution. In order to get a single description of a batch, we will assume jobs of a given batch to be added in increasing order of their index. As jobs are sorted by decreasing order of their processing times, the processing time pb_k of the batch k is given by the processing time of the first added job. We call this first job $f(k)$.

Let BR_k be the remaining capacity of the batch k. Based on Remark 1, a job i can be added in the current batch CB if:

- $BR_{CB} \geqslant s_i$,
- $i = f(CB)$ or its index is greater than the index of the last job added to CB.

All the jobs contained in J are able to become the first job of the following batch.
Algorithm 1 (detailed in Appendix A) generates all the feasible schedules.
The procedure AddNewJob(ja, CB, BR):

- adds the job ja in the current batch CB of remaining capacity BR,
- selects the next batch and the next job jn to be added.

The main program adds the first job in batch 1.

4.2. Regular criteria

Any criteria that is non-decreasing in the job completion times is called *Regular*. We can easily verify that criteria $C_{\max}$, $\sum C_i$, $\sum w_i C_i$, $T_{\max}$, $\bar{T} \sum w_i T_i$ are regular.
We say that a schedule is left-shifted if:

1. There is no job i in a batch $h > k$ such that $BR_k \geqslant s_i$ and $pb_k \geqslant p_i$ (we cannot shift i from h to k),
2. the starting time of batch $k + 1$ is equal to the ending time of batch k.

Lemma. *Left-shifted schedules are dominant for any regular criterion.*

Remark 2. At step st, let F be the set of jobs i of J, such that $s_i \leqslant BR_{CB}$ and $p_i \leqslant pb_{CB}$. If F is a non-empty set, we cannot add any job to the next batch. Moreover, in the current batch, we can only add jobs in F whose index is greater than that of the last added job.

Otherwise, if F is an empty set, all jobs in J are candidate to become the first job of the next batch $CB + 1$.

4.3. $C_{\max}$ criterion

Remark 3. Let O and O' be two schedules constituted by the same batches sequenced in a different order. For the $C_{\max}$ criteria, this two schedules have equal value. To avoid solutions which would be equivalent by permutation of the batches, we will impose the condition $f(k) < f(k + 1)$.

Corollary 1. *Due to Remark* 1, $f(k)$ *is unique and is defined by the smallest job that has not been scheduled in batches* $1, \ldots, k-1$. *In particular, job* 1 *can only be the first job of the first batch. Moreover, if* $F = \emptyset$, *the only job jn that can be added in the next batch must satisfy* $jn = \min(i/i \in J)$.

Property 1. *If we have two jobs* i *and* j *such that* $s_i = s_j$ *and* $p_i \geqslant p_j$, *then there exists an optimal schedule in which job* i *is scheduled before job* j.

Proof. Let i and j be two jobs of equal size such that $i < j$ (i.e. $p_i \geqslant p_j$). Assume i has been added to a batch k and j to a batch h such that $h < k$ (i.e. $pb_h \geqslant pb_k$).

Job i is in batch k, so $p_i \leqslant pb_k$. As $p_j \leqslant p_i$, $p_j \leqslant pb_k$ and j can be placed into batch k without increasing $C_{\max}$. Moreover, as $pb_k \leqslant pb_h$, $p_i \leqslant pb_h$ and i can be placed in batch h without increasing $C_{\max}$. So the two jobs i and j can be switched.

Consequence: Given a job i, let a_i denote the job such that: $a_i = \max(j < i, \ s_j = s_i)$. We set $a_i = 0$ if such a job does not exist. Then if $a_i > 0$, we have to add the job a_i before adding i.

Enumeration scheme. Finally, for $C_{\max}$ criterion, the enumeration scheme will be as follows:

At step st, let F be the set of jobs i of J such that $s_i \leqslant BR_{CB}$ and $a_i \notin J$. If F is non-empty, we cannot add any job to the next batch. Moreover, in the current batch, we can only add jobs of F whose index is above the one of the last added job. If F is an empty set, only the smallest job of J can be added.

Corollary 2. *Let us consider the special case where all jobs have the same size* $s_i = s$. *The previous algorithm builds a single solution within n steps. This solution is optimal and the problem is polynomial. An easier way to find this optimal solution is given by the following algorithm:*

1. *Order jobs by non-increasing value of* p_i.
2. *Let* $m = \lfloor B/s \rfloor$.
3. *Move the first m jobs to the batch* 1, *the following m ones to the batch* 2, *and repeat until all jobs have been assigned.*

5. Branch and bound

In this section, we present the Branch and Bound algorithm developed to deal with minimizing the makespan on a batching machine with non-identical job sizes. This algorithm integrates properties and remarks of the previous sections.

We use the enumeration scheme presented in Section 4. To compute a starting value, we use both FFLPT and BFLPT heuristics (see Section 3).

The cut is made before considering the next batch. We use the lower bound defined in Section 3 to get a lower bound $BInf(J)$ on the set of non-scheduled jobs J (in particular, $CMin$ represents the lower bound for the whole set of jobs N). Let $ValBest$ be the value of the best-known solution and $ValSol$ be the value of the partial schedule on batches 1 to CB. If $ValSol + BInf(J) \geqslant ValBest$, we can cut at this node. Otherwise, we determine a feasible solution on J by a greedy algorithm with solution value $ValGreedy(J)$ (in our experiment, we use the FFLPT method as the

greedy algorithm, because it is the quickest of the three heuristics presented in Section 3). If $ValSol + ValGreedy(J) \leqslant ValBest$, we have a new best solution. Moreover, if $CMin = ValSol + ValGreedy(J)$, we get the optimal solution on J and we can cut the enumeration at this node.

We stop the program, either if we can prove that the optimal solution is found, or if the maximum allowed time is reached.

The detailed algorithm is given in Appendix B.

6. Computational experiments

In order to determine the performance of these methods (exact and heuristic), we tested them on a wide range of problems. We generated four categories of problems. In order to be close to industrial problems, where products of similar processing times have to be grouped into batches, the capacity of the batching machine has been normalized to 100 and processing times are integers uniformly generated in [80, 120]. The categories differ by the job sizes. The first two categories have relatively small jobs with job sizes uniformly generated in [1, 10] (category 1) and [5, 10] (category 2). The other categories have larger jobs. Sizes have been generated according to the uniform distribution over the intervals [5, 20] and [5, 30]. For each of these categories we considered the number of jobs n equal to 40, 60, 80, 100, 200, 300 and 400. For each set, we created 20 instances.

Algorithms were implemented in PASCAL language and run on a PC (Pentium II, 266 MHz). The maximum time limit has been fixed to 15 min. The performance of the heuristic and the branch-and-bound algorithm are given in Table 1. The first two columns describe problems (job sizes and number of jobs). Column 3 indicates the number of problems for which the optimal solution has been proven. Then the four following columns give performances of the initial heuristic (column 4) and of the branch-and-bound after 1 min (Column 5), after 5 min (Column 6) and after 15 min (Column 7). Two results are reported: average over the 20 instances, and in parentheses, the worst case, when it is different to 0. Performances are calculated in comparison with the optimal solution (when found) or with the lower bound ($Perf = (C^H_{\max} - C^*_{\max})/C^*_{\max}$).

This table shows that the first solution given by the best of the two heuristics FFLPT and BFLPT is close to optimum ($< 4\%$, in average, above the optimum, when found or the lower bound).

For small jobs, with sizes varying from 1 to 10, the heuristic always finds the optimum. When job sizes are less dispersed ([5, 10]), problems are more difficult to solve. The heuristic gives solutions up to 3% above the optimum, but the branch-and-bound manages to improve this first solution very quickly.

For larger jobs, it becomes more difficult to prove optimality, even for reasonable problem sizes. But even for these categories of problems, solutions found by the branch-and-bound are of good quality, as the performance indicated in the table are calculated relative to lower bounds.

So, this table shows that the proposed branch-and-bound is able to improve the initial solution significantly in 1 min, and can then be used as an improvement procedure. In particular, we may remark that when the heuristic produce bad solutions, the branch and bound improves them quickly. For example, the worst initial solution for the problem 5–10–60, is 19.8% over the optimum and is solved to optimality, thanks to the branch and bound, within 1 min.

Table 1
Performances of the branch-and-bound (in %)

Job sizes	No. of jobs	Optimum found	Heuristic perf.	Perf. after 1 min	Perf. after 5 min	Perf. after 15 min
1–10	40	100	0.00			
	60	100	0.00			
	80	100	0.00			
	100	100	0.00			
	200	100	0.00			
	300	100	0.00			
	400	100	0.00			
5–10	40	100	0.00			
	60	100	2.00 (19.8)	0.00		
	80	100	0.06 (0.2)	0.00		
	100	90	2.97 (11.7)	0.05 (0.5)	0.05 (0.5)	0.04 (0.4)
	200	65	1.91 (6.1)	0.67 (6.1)	0.36 (5.9)	0.35 (5.9)
	300	15	2.71 (4.3)	0.51 (4.1)	0.31 (4.0)	0.31 (4.0)
	400	10	1.68 (3.2)	0.71 (3.1)	0.70 (3.1)	0.70 (3.1)
5–20	40	100	2.49 (16.3)	0.00		
	60	90	0.12 (1.0)	0.03 (0.4)	0.03 (0.4)	0.03 (0.4)
	80	60	0.19 (9.3)	0.09 (0.3)	0.08 (0.3)	0.07 (0.3)
	100	45	0.88 (6.8)	0.14 (0.5)	0.12 (0.4)	0.11 (0.4)
	200	15	1.56 (3.9)	0.85 (3.9)	0.66 (3.9)	0.64 (3.9)
	300	5	1.15 (2.6)	0.57 (2.5)	0.45 (2.5)	0.44 (2.5)
	400	5	0.99 (2.1)	0.56 (2.0)	0.47 (1.9)	0.46 (1.9)
5–30	40	95	3.70 (12.8)	0.03 (0.9)	0.03 (0.9)	0.03 (0.9)
	60	85	2.87 (8.3)	0.12 (0.6)	0.06 (0.4)	0.05 (0.3)
	80	35	0.87 (6.2)	0.19 (0.6)	0.15 (0.4)	0.13 (0.4)
	100	40	1.46 (5.2)	0.66 (5.0)	0.62 (4.9)	0.61 (4.9)
	200	0	1.55 (2.8)	1.17 (2.8)	1.16 (2.8)	0.95 (2.8)
	300	0	1.45 (2.0)	0.89 (1.9)	0.79 (1.9)	0.78 (1.9)
	400	0	1.14 (1.6)	0.89 (1.5)	0.77 (1.5)	0.77 (1.5)

Table 2
Performances with smaller scale problems ($B = 40$) in %

Job sizes	No. of jobs	Optimum found	Heuristic perf.	Perf. after 1 min	Perf. after 5 min	Perf. after 15 min
2–8	40	100	0.06 (0.8)	0.00		
	60	100	0.09 (0.3)	0.00		
	80	100	0.98 (8.8)	0.00		
	100	85	0.46 (7.2)	0.01 (0.1)	0.01 (0.1)	0.01 (0.1)
	200	50	0.65 (3.6)	0.06 (0.3)	0.06 (0.3)	0.06 (0.3)
	300	35	0.48 (2.6)	0.05 (0.1)	0.04 (0.1)	0.04 (0.1)
	400	35	0.39 (2.0)	0.05 (0.2)	0.05 (0.2)	0.05 (0.2)

Table 3
Performances with smaller scale problems ($B = 20$) in %

Job sizes	No. of jobs	Optimum found	Heuristic perf.	Perf. after 1 min	Perf. after 5 min	Perf. after 15 min
1–4	40	100	0.00			
	60	100	0.66 (13.0)	0.00		
	80	100	0.00			
	100	100	0.00			
	200	100	0.00			
	300	100	0.00			
	400	90	0.10 (2.0)	0.002 (0.02)	0.002 (0.02)	0.002 (0.02)

Moreover, one must take the influence of the scale of data into account. For example, we compared problems with a batch capacity of 100 and job sizes in [5, 20] with equivalent problems of less precision: problems with a batch capacity of 40 and job sizes in [2, 8] (scale 2/5) and problems with a batch capacity of 20 and job sizes in [1, 4] (scale 1/5). Tables 2 and 3 report these new results and show that problems with less precision (smaller scale) are far easier to solve.

7. Conclusion

This paper provides a branch-and-bound method for the problem of minimizing the maximum completion time, on a single-batch processing machine. Thanks to dominance properties, the enumeration scheme has been reduced and allows to solve exactly problems with a large number of jobs, more than with other exact methods. The experiment shows that the more the job sizes are different the more problems are difficult to solve. But even for such difficult problems, the enumeration used as a heuristic method is able to find good-quality solutions. Hence, the exact method can be used in lots of industrial applications, where products to group in batches are often of similar sizes. Moreover, this enumeration method can be seen as a great tool to evaluate heuristics developed for such problems.

Appendix A. Algorithm 1

Generic enumeration.
Job_{st} and $Batch_{st}$ represent, respectively, the added job at step st and its batch.

```
Main Program
| st ← 0
| J = {1, ..., n}
| for jn from 1 to n do
| | AddNewJob (jn, 1, B)
| endfor
End Main Program
```

```
AddNewJob (ja, CB, BR)
| st ← st + 1
| J ← J − {ja}
| Jobₛₜ ← ja
| Batchₛₜ ← CB
| BR ← BR − s_ja
| if J ≠ ∅ then
            // Add a new job jn > ja in the current batch
| | for jn from ja + 1 to n do
| | | if jn ∈ J then
| | | | if BR ⩾ s_jn then AddNewJob (jn, CB, BR)
| | endfor
            // Add a new job in the following batch
| | for jn from 1 to n do
| | | if jn ∈ J then AddNewJob (jn, CB + 1, B)
| | endfor
| else
| | Write Solution
| endif
|
| J ← J + {ja}
| st ← st − 1
End AddNewJob
```

Appendix B. Algorithm 2

```
Main Program
| J = {1, …, n}
| st ← 0
| Determine a feasible solution on J
| Let ValBest be its value
| Determine BInf(J) a Lower Bound on J
| CMin ← BInf(J)
| Optimum ← (ValBest = CMin)
| if not Optimum then
| | ValSol ← p₁
| | AjoutJob(1, 1, B)
| endif
End Main Program
```

```
AddNewJob (ja, CB, BR)
| Stop ← (Time ⩾ MaximumTime)
| st ← st + 1
| J ← J − {ja}
| Job_st ← ja
| Batch_st ← CB
| BR ← BR − s_ja
| if J ≠ ∅ then
| | let F = {j ∈ J, s_j ⩽ BR, a_j ∉ J}
| endif
            // Add a new job jn > ja in the current batch
| if J ≠ ∅ and F ≠ ∅ then
| | for jn from ja + 1 to n do
| | | if jn ∈ F then
| | | | if not Stop do AddNewJob (jn, CB, BR)
| | endfor
| endif
            // Add the first job in the next batch
| if J ≠ ∅ and F = ∅ then
| | Determine BInf(J) a Lower bound on J
| | if ValSol + BInf(J) < ValBest then
| | | Determine a feasible solution on J by a greedy algorithm
| | | Let ValGreedy(J) be its value
| | | if ValSol + ValGreedy(J) < ValBest then
| | | | ValBest ← ValSol + ValGreedy(J)
| | | | Optimum ← (ValBest = CMin)
| | | | if Optimum then Stop ← true
| | | endif
| | | if not Stop then
| | | | if BInf(J) < ValGreedy(J) then
| | | | | | jn = min(j ∈ J)
| | | | | | ValSol ← ValSol + p_jn
| | | | | | AddNewJob(jn, CB + 1, B)
| | | | | | ValSol ← ValSol − p_jn
| | | | endif
| | | endif
| | endif
| endif

| if J = ∅
| | if ValBest > ValSol then
| | | ValBest ← ValSol
| | | Optimum ← (ValBest = CMin)
| | | if Optimum then Stop ← true
```

```
| | endif
| endif
|
| J ← J + {ja}
| st ← st − 1
endif
```

End AddNewJob

Algorithm 2

References

[1] Ikura Y, Gimple M. Efficient scheduling algorithms for a single batch processing machine. Operations Research Letters 1986;5:61–5.
[2] Lee CY, Uzsoy R, Martin-Vega LA. Efficient algorithms for scheduling semiconductor burn-in operations. Operations Research 1992;40:764–75.
[3] Li CL, Lee CY. Scheduling with agreeable release times and due dates on a batch processing machine. European Journal of Operational Research 1997;96:564–9.
[4] Ahmadi JH, Ahmadi RH, Dasu S, Tang CS. Batching and scheduling jobs on batch and discrete processors. Operations Research 1992;40:750–63.
[5] Brucker P, Gladky A, Hoogeveen H, Kovalyov MY, Potts C, Tautenhahn T, Van De Velde S. Scheduling a batching machine. Journal of Scheduling 1998;1:31–55.
[6] Chandru V, Lee CY, Uzsoy R. Minimizing total completion time on batch processing machines. International Journal of Production Research 1993;31:2097–121.
[7] Dupont L, Jolai Ghazvini F. Branch and bound method for single batch processing machine with mean flow time criteria. International Journal of Industrial Engineering: Practice and Theory 1997;4:197–203.
[8] Uzsoy R, Yang Y. Minimizing total weighted completion time on a single batch processing machine. Production and Operations Management 1997;6:57–73.
[9] Lee CY, Uzsoy R. Minimizing makespan on a single batch processing machine with dynamic job arrivals. International Journal of Production Research 1999;1:219–36.
[10] Kempf KG, Uzsoy R, Wang C-S. Scheduling a single batch processing machine with secondary resource constraints. Journal of Manufacturing Systems 1998;17:37–51.
[11] Uzsoy R. Scheduling batch processing machines with incompatible job families. International Journal of Production Research 1995;33:2685–708.
[12] Mehta SV, Uzsoy R. Minimizing total tardiness on a batch processing machine with incompatible job families. IIE Transactions on Scheduling and Logistics 1998;31:165–78.
[13] Chandru V, Lee CY, Uzsoy R. Minimizing total completion time on a batch processing machine with job families. Operations Research Letters 1993;13:61–5.
[14] Hochbaum DS, Landy D. Scheduling semiconductor burn-in operations to minimize total flowtime. Operations Research 1997;45:874–85.
[15] Kubiak W, Jolai Ghazvini F. Minimizing earliness/tardiness criteria on a single batch processing machine with families of jobs. Proceeding of the 2nd Annual International Conference on Industrial Engineering, vol. 2, San Diego, California, USA, 12–15 November 1997. p. 785–90.
[16] Dobson G, Nambinadom RS. The batch loading and scheduling problem. Research Report, Simon School of Business Administration, University of Rochester, Rochester, NY, 1992.
[17] Uzsoy R. A single batch processing machine with non-identical job sizes. International Journal of Production Research 1994;32:1615–35.

[18] Jolai Ghazvini F, Dupont L. Minimizing mean flow time criteria on a single batch processing machine with non-identical job sizes. International Journal of Production Economics 1998;55:273–80.

[19] Dupont L, Jolai Ghazvini F. Minimizing makespan on a single batch processing machine with non-identical job sizes. European Journal of Automation (JESA) 1998;32:431–40.

[20] Coffman EG, Garey MR, Johnson DS. Approximation algorithms for bin–packing – an updated survey. In: Ausiello G, Lucertini M, Serafini P, editors. Algorithm design for computer system design, New York: Springer, 1984. p. 49–106.

Lionel Dupont is a Professor in the Industrial Engineering and Management School of Grenoble, France. His research interests covers areas in production planning, project management, scheduling and sequencing.

Clarisse Dhaenens-Flipo is an engineer in Computer Science. She received her Ph.D. degree in Operations Research from the Grenoble University of Technology (INPG), France. She is currently Assistant Professor in computer science and applied mathematics at the University of Lille, France. Her research interests are in scheduling and operations research approaches to production and operations management.