J Sched (2007) 10: 353–364
DOI 10.1007/s10951-007-0041-9

Batching and scheduling in a multi-machine flow shop

C.T. Ng · Mikhail Y. Kovalyov

Published online: 20 October 2007

Abstract We study the problem of batching and scheduling n jobs in a flow shop comprising m, $m \geq 2$, machines. Each job has to be processed on machines $1, \ldots, m$ in this order. Batches are formed on each machine. A machine dependent setup time precedes the processing of each batch. Jobs of the same batch are processed on each machine sequentially so that the processing time of a batch is equal to the sum of the processing times of the jobs contained in it. Jobs of the same batch formed on machine l become available for a downstream operation on machine $l+1$ at the same time when the processing of the last job of the batch on machine l has been finished. The objective is to minimize maximum job completion time. We establish several properties of an optimal schedule and develop polynomial time algorithms for important special cases. They are improvements over the existing methods with regard to their generality and time efficiency.

Keywords Scheduling · Batching · Flow shop · Dynamic programming

C.T. Ng
Department of Logistics, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
e-mail: lgtctng@polyu.edu.hk

M.Y. Kovalyov (✉)
Belarusian State University, and United Institute of Informatics Problems, National Academy of Sciences of Belarus, Nezavisimosti 4, 220030, Minsk, Belarus
e-mail: koval@newman.bas-net.by

1 Introduction

We consider the following problem of batching and scheduling n non-preemptive independent jobs in a flow shop environment. Each job is ready for processing at time zero and it has to be processed on machines $1, 2, \ldots, m$ in this order. The processing time of job j on machine l, that is, the duration of operation (j, l) is equal to p_{jl}. Batches are formed on each machine. They may include any number of jobs. Jobs of the same batch are processed on each machine sequentially so that the processing time of a batch is equal to the sum of the processing times of the jobs contained in it. A batch of jobs formed on machine l can start its processing only after all its jobs have been completed on machine $l-1$. Furthermore, jobs of the same batch formed on machine l complete on this machine and become available for a downstream operation on machine $l+1$ at the same time when the processing of the last job of this batch on machine l has been finished. This type of processing corresponds to the *batch availability model* in the literature on batch scheduling. A traditional type of processing in which each job becomes available for a downstream operation immediately when the current machine has finished its processing corresponds to the *job (or item) availability model*, see Webster and Baker (1995), Tanaev et al. (1998), and Potts and Kovalyov (2000) for more information on batch scheduling models. A machine dependent setup time s_l immediately precedes the processing of each batch on machine l, $l = 1, \ldots, m$. No machine can process a job while performing a setup. In other words, all setups are performed on-line. We distinguish the cases when all setups are *anticipatory* and all of them are *non-anticipatory*. An anticipatory setup can be performed before the batch arrives at the corresponding machine. A non-anticipatory setup can start only after the batch becomes available at the machine. All machine

setup times and job processing times are assumed to be non-negative integer numbers. A batch schedule is characterized by the batch sequences on machines $1, \ldots, m$. A batch sequence is an ordered collection of batches. Since the batch availability model is applied, the order of jobs in each batch does not affect job completion times. We distinguish *permutation* and *non-permutation* schedules, and schedules with *consistent* and *inconsistent* batches. A schedule is a permutation schedule if job sequences are the same on all machines. The batches are consistent if batch partitions are the same on all machines. Opposite statements define a non-permutation schedule and inconsistent batches. Note that a permutation schedule with consistent batches is totally determined by the batch sequence on machine 1.

The problem is to find a feasible batch schedule that minimizes the completion time of the last job on machine m, that is, the makespan $C_{\max}$. We denote this problem as $Fm|sum\text{–}batch, \nu, s_l^{\mu}|C_{\max}$, where $\nu = consi$ if the batches are restricted to be consistent and ν is omitted otherwise, and $\mu = A$ if setup times are anticipatory, $\mu = NA$ if they are non-anticipatory and μ is omitted if setup times are of any of the two types. A special case in which the schedules are restricted to be permutation ones is denoted as $PFm|sum\text{–}batch, \nu, s_l^{\mu}|C_{\max}$. Notations $p_{jl} = p$ and $s_l = s$ in the second field mean that all job processing times are equal to p and all setup times are equal to s, respectively.

Batch scheduling problems have been attracting much attention of the scheduling community lately. Reviews of the results in this area can be found in Potts and Van Wassenhove (1992), Webster and Baker (1995), Tanaev et al. (1998), and Potts and Kovalyov (2000). Problem $Fm|sum\text{–}batch, s_l|C_{\max}$ with identical jobs is closely related to lot streaming and lot splitting models. An overview of such basic models was given by Trietsch and Baker (1993). Recent results in this area can be found in Kalir and Sarin (2000), Bukchin et al. (2002), Bukchin and Masin (2004), Huq et al. (2004), Li and Xiao (2004), and Chin and Chang (2005).

Motivation for problem $Fm|sum\text{–}batch, s_l|C_{\max}$ comes from batch production in a multi-stage flexible manufacturing cell. The part-types to be produced are mounted on pallets for machining. The capacity of a pallet is usually quite large and it can be reasonably assumed to be unbounded. The part-types mounted on the same pallet are processed together. Cheng et al. (2000) observed such a situation in a manufacturer of pneumatic valves.

The studies on the considered problem are limited. Cheng et al. (2000) proved that problem $PF2|sum\text{–}batch, consi, s_l^{NA} = s|C_{\max}$ is NP-hard in the strong sense, derived several properties of an optimal solution and developed $O(n)$, $O(n^2)$, and $O(n^3)$ time algorithms for the cases (1) $p_{jl} = p,\ j = 1, \ldots, n,\ l = 1, 2$, (2) $p_{j1} = p,\ j = 1, \ldots, n$, or $p_{j2} = p,\ j = 1, \ldots, n$, and (3) $p_{11} \le p_{21} \le \cdots \le p_{n1}$ and $p_{12} \ge p_{22} \ge \cdots \ge p_{n2}$, respectively. They also suggested several heuristics for the general case of their problem. Glass et al. (2001) studied the problem $F2|sum\text{–}batch, s_l^A|C_{\max}$. They proved that the batches are consistent in an optimal schedule, the problem is strongly NP-hard and derived an heuristic with a tight worst-case performance bound of 4/3. The heuristic constructs a schedule with at most three consistent batches. We became recently aware of the papers by Mosheiov and Oron (2004) and Mosheiov et al. (2004) on m-machine flow shop batch scheduling with identical processing time jobs. Manuscript (Mosheiov and Oron 2004) generalizes the result of Cheng et al. (2000) for identical processing time jobs and common setup time to the case of m machines. In Mosheiov et al. (2004), the total job completion time criterion is considered and formulas for calculating batch sizes are given for a relaxed problem, in which batch sizes are allowed to be non-integer.

In Sect. 2, we prove that, for problems $Fm|sum\text{–}batch, s_l|C_{\max}$ and $Fm|sum\text{–}batch, consi, s_l|C_{\max}$ there exists an optimal schedule for which job sequences are the same on machines 1 and 2, and they are the same on machines $m-1$ and m, $m \ge 2$. Therefore, a permutation schedule is optimal for problems $F3|sum\text{–}batch, s_l|C_{\max}$ and $F3|sum\text{–}batch, consi, s_l|C_{\max}$. This property is important for applying local search techniques to solve the three-machine problem. We give examples where all optimal schedules are non-permutation ones for $m \ge 4$. Then we show that there exists an optimal schedule for the problem $F2|sum\text{–}batch, s_l|C_{\max}$, which is a permutation one with consistent batches. We give examples where all optimal schedules have inconsistent batches for $m \ge 3$. We also demonstrate that the problem $F2|sum\text{–}batch, consi, s_l|C_{\max}$ with k consistent and given batches can be solved in $O(k \log k)$ time by known algorithms. In Sect. 3, we study the problem $PFm|sum\text{–}batch, s_l|C_{\max}$ under the assumption that the job sequence is given. Investigation of this problem is important because its solution algorithm can be used to evaluate job sequences in a local search method for the problem with the same job sequence on all machines. For such a problem, an optimal batch partition has to be found on each machine. In Sect. 3.1, we propose a dynamic programming algorithm for this problem with $O(n^{5m-7})$ running time, if $m \ge 3$ is a constant. For $m = 2$, it can be modified to run in $O(n^3)$ time. In Sect. 3.2, we give an example of a problem, in which an optimal job sequence, the same for all machines, can easily be found. In this problem, job processing times are the same on each machine $l = 2, \ldots, m-1$ and they are oppositely ordered on machines 1 and m. In Sect. 3.3, we develop an iterative $O(n^2 \log n \log L)$ time algorithm for the problem $PF2|sum\text{–}batch, s_l|C_{\max}$ with a given job sequence, where $L = \max_j\{n, s_1, s_2, p_{j1}, p_{j2}\}$. In Sect. 4, we assume that m is variable, job processing times are all equal and setup times are all equal. For this case,

$Fm|sum\text{–}batch, s_l = s, p_j = p|C_{\max}$, an $O(\sqrt{n})$ time algorithm is presented, which is better than the $O(n)$ algorithm of Cheng, Lin and Toker suggested for a more restrictive problem with $m = 2$ and non-anticipatory setups. This paper concludes with a summary of the results, some observations and suggestions for future research.

2 Properties of an optimal schedule

Theorem 1 *There exists an optimal schedule for the problem* $Fm|sum\text{–}batch, s_l|C_{\max}$, *in which job sequences are the same on machines* 1 *and* 2, *and they are the same on machines* $m-1$ *and* m.

Proof Let S be an optimal schedule. In this schedule, let job i precede job j on machine 1 and job j precede job i on machine 2. If i and j are in the same batch on machine 1 or machine 2, then they can be interchanged without changing job completion times on any machine. Assume that the jobs are in different batches on machines 1 and 2. On machine 1, shift job i to the batch of job j. Only completion time of job i on machine 1 can increase. The makespan value will not increase. Repeat the described shifting operation a finite number of times for all pairs of jobs that are not in the same order on machines 1 and 2. The resulting schedule will be optimal and it will have the same job sequences on machines 1 and 2. Thus, theorem is proved for $m = 2$. Note that we did not change the batch sequence on machine 2.

Consider an optimal schedule with the same batch sequence on machines 1 and 2. Assume that job k precedes job r on machine $m-1$, job r precedes job k on machine m and they are in different batches on machines $m-1$ and m. On machine m, shift job k to the batch of job r. Only start time of job k on machine m can decrease. The makespan value will not increase. Therefore, the schedule will remain optimal. Repeat the described shifting operation a finite number of times for all pairs of jobs that are not in the same order on machines $m-1$ and m. The resulting schedule will evidently satisfy the statement of the theorem. □

Statement 1 *There exists an optimal schedule for the problem* $F3|sum\text{–}batch, s_l|C_{\max}$, *which is a permutation one.*

This statement is a re-wording of the above theorem for $m = 3$.

Statement 2 *There exists an optimal schedule for the problem* $Fm|sum\text{–}batch, consi, s_l|C_{\max}$, *in which job sequences are the same on machines* 1 *and* 2, *and they are the same on machines* $m-1$ *and* m.

Proof Since the batches are consistent, they can be considered as composite jobs and the result for classical m-machine flow shop scheduling problem can be applied, which states that there exists an optimal schedule, in which the sequence of composite jobs is the same on machines 1 and 2, and it is the same on machines $m-1$ and m. □

Theorem 2 *There exists an optimal schedule for the problem* $F2|sum\text{–}batch, s_l|C_{\max}$, *which is a permutation one with consistent batches.*

Proof Glass et al. (2001) proved this statement for the case when setups are anticipatory. Based on Theorem 1, it can easily be proved for both anticipatory or non-anticipatory setups as follows. Let S be an optimal permutation schedule. Assume that some jobs i and j are scheduled in the same batch on one machine and they are scheduled in different batches on the other machine. On the other machine, combine the batch of job i, the batch of job j and the batches sequenced between these two batches to form a single batch. The makespan will not increase. Repeat this operation a finite number of times for all pairs of jobs that are in the same batch on one machine and they are in different batches on the other machine. The resulting schedule will be optimal and will have consistent batches on machines 1 and 2. □

Statement 3 *There exist examples of problems* (a) $F3|sum\text{–}batch, s_l, p_{jl} = 1|C_{\max}$ *and* (b) $F3|sum\text{–}batch, s_l = 1|C_{\max}$ *in which all optimal schedules have inconsistent batches.*

Proof Due to Theorem 1, we limit our considerations to permutation schedules.

(a) Consider an example in which there are two jobs with unit processing times on each machine. Setup times are $s_1 = 2$ and $s_2 = s_3 = 0$. Since setup times on machines 2 and 3 are equal to zero, all setups are anticipatory and non-anticipatory at the same time. The only optimal batch schedule with one batch on machine 1 and two batches on machines 2 and 3 has value $C_{\max} = 7$. If two batches are formed on machine 1 or only one batch is formed on machine 2 or machine 3, then $C_{\max} \geq 8$.

(b) Consider an example in which there are three jobs with processing times p_{lj} given in Table 1. The only optimal batch schedule has value $C_{\max} = 10$ for anticipatory setups and $C_{\max} = 12$ for non-anticipatory setups. Optimal batch sequence on machine 1 is (1, 2), (3) and optimal batch sequences on machines 2 and 3 are the same: (1), (2), (3).

Observe that for any job sequence, if there is only one batch on each machine, then $C_{\max}$ is greater than the total

Table 1 Processing times p_{lj}

$l \backslash j$	1	2	3
1	1	1	4
2	1	2	1
3	2	1	1

Table 2 $C_{\max}$ values for permutation schedules with two consistent batches

Batch sequence	(1), (2, 3)	(1, 2), (3)	(2), (1, 3)	(2, 1), (3)
$C^A_{\max}(C^{NA}_{\max})$	13(15)	11(13)	13(15)	11(13)

job processing time, which is 14. If there are three consistent batches, then machine 1 completes at time 9. After this time instant the last job can start processing on machines 2 and 3, thus adding its processing times on machines 2 and 3 to the $C_{\max}$ value. If setup times are non-anticipatory, then setup times preceding this last job on machines 2 and 3 must be added to the $C_{\max}$ value as well. This observation implies that $C_{\max} \geq 11$ for anticipatory setups and $C_{\max} \geq 13$ for non-anticipatory setups if there are three consistent batches.

Assume that there are two consistent batches. Then machine 1 completes at time 8. Similarly to the previous case, after this time instant the last job can start processing on machines 2 and 3, thus adding its processing times on machines 2 and 3 to the $C_{\max}$ value. If setup times are non-anticipatory, then setup times preceding this last job on machines 2 and 3 must be added to the $C_{\max}$ value as well. We deduce that if there are two consistent batches, then the total processing time of the last job on machines 2 and 3 should not exceed 2 in an optimal schedule. This condition is satisfied only for the job sequences $(1, 2, 3)$ and $(2, 1, 3)$.

Table 2 gives values $C_{\max} = C^A_{\max}(C^{NA}_{\max})$ for feasible solutions with job sequences $(1, 2, 3)$ and $(2, 1, 3)$ and two consistent batches, where $C^A_{\max}$ is the makespan value for anticipatory setups and $C^{NA}_{\max}$ is the makespan value for non-anticipatory setups. All of them are not optimal. □

Statement 4 *There exist examples of the problem* $F4|sum\text{–}batch, s_l|C_{\max}$ *in which all optimal schedules are non-permutation ones.*

Proof The validity of this statement follows from the fact that in the case when all setup times are equal to zero, it is optimal that each batch includes a single job and the batch scheduling problem is equivalent to the classical flow shop problem for which this statement holds. □

Consider the case where there are k consistent and given batches. Index them arbitrarily $B_1, \ldots, B_k$. Consider batch B_i as a composite job i with processing time $P_{il} = s_l + \sum_{j \in B_i} p_{jl}$ on machine l, $l = 1, \ldots, m$.

Statement 5 *The problem* $Fm|sum\text{–}batch, consi, s_l|C_{\max}$ *with* k *consistent and given batches can be solved in* $O(k \log k)$ *time.*

Proof For non-anticipatory setups, the problem $Fm|sum\text{–}batch, consi, s_l|C_{\max}$ with k consistent and given batches is evidently equivalent to the classical m-machine flow shop problem with k composite jobs. Therefore, for $m = 2$ an optimal solution can be found by applying Johnson's algorithm (Johnson 1954) to the corresponding composite jobs. For anticipatory setups, the problem $Fm|sum\text{–}batch, consi, s_l|C_{\max}$ with k consistent and given batches is equivalent to the m-machine flow shop problem with k composite jobs such that an execution of a composite job on machine $l + 1$ can start s_{l+1} time units before its completion on machine l, $l = 1, \ldots, m - 1$. For $m = 2$, this problem can be solved in $O(k \log k)$ time by determining the sequencing priorities for the composite jobs, see Tanaev et al. (1994). □

3 Given job sequence

For the problem $F2|sum\text{–}batch, s_l|C_{\max}$, there exists an optimal schedule that is a permutation one with consistent batches. However, the problem is NP-hard in the strong sense. Therefore, the indicated properties are not enough to find an optimal schedule in a polynomial time and further assumptions should be imposed in order to solve the problem in a polynomial time, unless $\mathcal{P} = \mathcal{NP}$.

In this section, we consider the problem $PFm|sum\text{–}batch, s_l|C_{\max}$, $m \geq 2$, in which job sequence is the same on all the machines and given. For such a problem, an optimal batching decision has to be made for each machine. This problem is important for two reasons. First, it can be applied in situations when the same optimal job sequence is given. Second, it can be used to evaluate the quality of a trial job sequence in the local search techniques for permutation flow shops. Let the given job sequence be $(1, \ldots, n)$.

3.1 Dynamic programming algorithm

We first assume that setups are non-anticipatory and $m \geq 3$. In our dynamic programming algorithm, denoted as DP, we construct partial feasible schedules forwards. For such a schedule, we assume that the machines complete their processing as early as possible.

Given a schedule, let T_l denote the completion time of machine l, $l = 1, \ldots, m$. Consider partial feasible schedules for jobs $1, \ldots, j$, in which the $(m-1)$-tuple $(T_1, \ldots, T_{m-1})$ is the same and the last batches on the machines $2, 3, \ldots, m$ start with the same jobs. If all such schedules are extended in the same way by adding job $j + 1$ to the same batches on all the machines, then the new machine completion times T^{new}_l, $l = 1, \ldots, m - 1$, will be the same for these schedules and a schedule with the minimum value T_m will have the minimum value T^{new}_m. Therefore, a schedule with minimum completion time of machine m is *dominant* among

these schedules such that it can be extended to a complete feasible schedule with minimum makespan value among all complete feasible schedules extended from any mentioned partial schedule.

We associate a *state* $(A_1, \ldots, A_m, j)$ with each partial feasible schedule. Here j is the number of jobs assigned to the schedule and $A_1, \ldots, A_m$ are sets of parameters associated with machines $1, \ldots, m$, respectively. We have

$$A_1 = \{a_1, k_1, j_1\},$$

$$A_l = \{a_l, k_l, h_l, j_l, v_l\}, \quad l = 2, \ldots, m-2,$$

$$A_{m-1} = \{d_{m-1}, h_{m-1}, v_{m-1}\}, \qquad A_m = \{v_m\},$$

where the parameters of the above sets are defined as follows. Set $h_1 = 1$.

- j_l is the last job of a batch on machine l where the *critical path* in the network corresponding to the considered schedule, which determines $C_{\max}$ value, switches from machine l to machine $l+1$ (definition of the critical path see, for example, in Tanaev et al. 1995).
- h_l, $h_l \le j_{l-1}$, is the first job in a batch on machine l where the critical path switches to machine l from machine $l-1$.
- a_l is the number of batches on machine l created for the jobs $h_l, h_l+1, \ldots, j_l$.
- k_l is the number of batches on machine l created for the jobs $j_l+1, j_l+2, \ldots, j$.
- v_l, $2 \le l \le m$, is the first job in the last batch on machine l.
- d_{m-1} is the number of batches on machine $m-1$ created for the jobs $h_{m-1}, h_{m-1}+1, \ldots, j$.

Denote $P_{kr}^l = \sum_{i=k}^{r} p_{il}$. The values P_{kr}^l, $l = 1, \ldots, m$, $1 \le k \le r \le n$, can be calculated in $O(mn^2)$ time. For a given partial schedule in the state $(A_1, \ldots, A_m, j)$, we can use the critical path method and calculate auxiliary values

$$H_0 := 0,$$

$$H_l := H_{l-1} + a_l s_l + P_{h_l j_l}^l, \quad l = 1, \ldots, m-2,$$

and machine completion times

$$T_l = H_l + k_l s_l + P_{j_l+1, j}^l, \quad l = 1, \ldots, m-2,$$

$$T_{m-1} = H_{m-2} + d_{m-1} s_{m-1} + P_{h_{m-1} j}^{m-1}.$$

Therefore, partial schedules in the same state $(A_1, \ldots, A_m, j)$ have the same completion times of machines $1, \ldots, m-1$, and a dominant schedule with the minimum makespan value can be chosen among them for further extension.

Denote by $C(A_1, \ldots, A_m, j)$ the minimum makespan value for all partial feasible schedules in the same state $(A_1, \ldots, A_m, j)$. In Algorithm DP, we recursively compute values $C(A_1, \ldots, A_m, j)$. In iteration j, we assign job j to the last batch on machine l or start a new batch for $l = 1, \ldots, m$. Such an assignment can be described by a 0-1 vector $(x_1, \ldots, x_m)$, where $x_l = 0$ if j is added to the last batch and $x_l = 1$ if it starts a new batch on machine l. The number of different assignments is 2^m. In the beginning, we set $C(A_1, \ldots, A_m, j) = 0$ if $A_1 = \cdots = A_m = \{0, \ldots, 0\}$ and $j = 0$, and $C(A_1, \ldots, A_m, j) = \infty$ for the remaining tuples $(A_1, \ldots, A_m, j)$.

Let us describe iteration $j+1$, $0 \le j \le n-1$, of Algorithm DP. Assume that we are given a state $A = (A_1, \ldots, A_m, j)$ with the associated minimum makespan value $C(A)$ and an assignment vector $x = (x_1, \ldots, x_m)$ for job $j+1$. We show how to calculate a new state $A' = (A'_1, \ldots, A'_m, j+1)$ corresponding to the schedule obtained from a schedule in the state A by assigning job $j+1$ according to x. We denote the makespan value of the new schedule by $C(A, x)$. Note that $C(A, x)$ may not be equal to the minimum makespan value $C(A')$ corresponding to A' because there may be several ways of obtaining A' from a "previous" state. Consider a diagram of a partial schedule corresponding to the state A. An example is given Fig. 1, which is also used for further discussion.

Construct a network that can be used to calculate the earliest possible machine completion times in a schedule corresponding to $(A'_1, \ldots, A'_m, j+1)$. For these purposes, introduce the following *(aggregate) operations* O_{il} with processing times t_{il}:

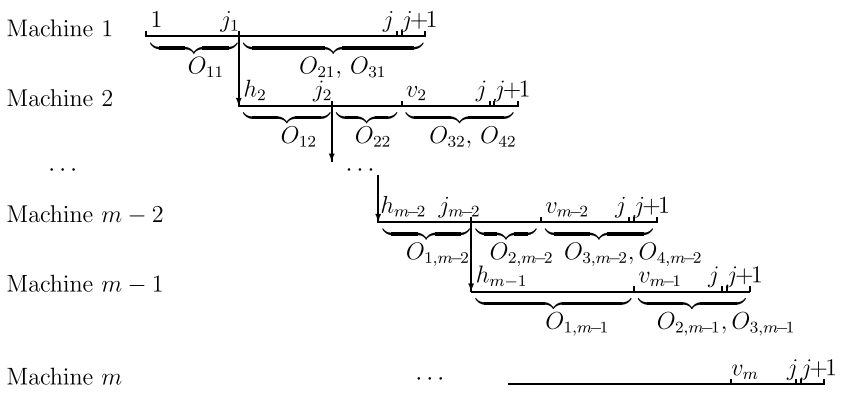

Fig. 1 A diagram of a partial schedule

– operations O_{il}, $i = 1, 2, 3$, if $x_l = 0$, and $i = 1, 2, 3, 4$, if $x_l = 1$, for $l = 2, \ldots, m-2$
– operations O_{il}, $i = 1, 2$, if $x_l = 0$, and $i = 1, 2, 3$, if $x_l = 1$, for $l \in \{1, m-1\}$.

Calculate operation processing times.
For $l = 2, \ldots, m-2$:

$$t_{1l} = a_l s_l + P^l_{h_l j_l},$$

$$t_{2l} = (k_l - 1)s_l + P^l_{j_l+1, v_l-1},$$

$$t_{3l} = \begin{cases} s_l + P^l_{v_l, j+1}, & \text{if } x_l = 0, \\ s_l + P^l_{v_l j}, & \text{if } x_l = 1, \end{cases}$$

$$t_{4l} = s_l + p_{j+1,l}, \qquad \text{if } x_l = 1.$$

For $l = 1$:

$$t_{11} = a_1 s_1 + P^1_{1 j_1},$$

$$t_{21} = \begin{cases} k_1 s_1 + P^1_{j_1+1, j+1}, & \text{if } x_1 = 0, \\ k_1 s_1 + P^1_{j_1+1, j}, & \text{if } x_1 = 1, \end{cases}$$

$$t_{31} = s_1 + p_{j+1,1}, \quad \text{if } x_1 = 1.$$

For $l = m-1$:

$$t_{1,m-1} = (d_{m-1} - 1)s_{m-1} + P^{m-1}_{h_{m-1}, v_{m-1}-1},$$

$$t_{2,m-1} = \begin{cases} s_{m-1} + P^{m-1}_{v_{m-1}, j+1}, & \text{if } x_{m-1} = 0, \\ s_{m-1} + P^{m-1}_{v_{m-1} j}, & \text{if } x_{m-1} = 1, \end{cases}$$

$$t_{3,m-1} = s_{m-1} + p_{j+1,m-1}, \quad \text{if } x_{m-1} = 1.$$

In the network, there is a chain (O_{1l}, O_{2l}, O_{3l}) if $x_l = 0$ and a chain $(O_{1l}, O_{2l}, O_{3l}, O_{4l})$ if $x_l = 1$ for $l = 2, \ldots, m-2$. There are chains (O_{1l}, O_{2l}) if $x_l = 0$ and (O_{1l}, O_{2l}, O_{3l}) if $x_l = 1$ for $l \in \{1, m-1\}$. Furthermore, there are two chains $(O_{11}, O_{12}, \ldots, O_{1,m-1})$ and $(O_{21}, O_{32}, O_{33}, \ldots, O_{3,m-2}, O_{2,m-1})$. For $l = 2, \ldots, m-3$, if $x_l = 1$ and $x_{l+1} = 0$, then there is an arc $(O_{4l}, O_{3,l+1})$. If $x_l = 1$ and $x_{l+1} = 1$, then there is an arc $(O_{4l}, O_{4,l+1})$. If $x_1 = 1$ and $x_2 = 0$, then there is an arc (O_{31}, O_{32}). If $x_1 = 1$ and $x_2 = 1$, then there is an arc (O_{31}, O_{42}). If $x_{m-2} = 1$ and $x_{m-1} = 0$, then there is an arc $(O_{4,m-2}, O_{2,m-1})$. If $x_{m-2} = 1$ and $x_{m-1} = 1$, then there is an arc $(O_{4,m-2}, O_{3,m-1})$.

Define the length of an oriented path in the constructed network as a summation of the processing times of the operations in this path. It is easy to see that the longest path connecting operations O_{11} and O_{3l} if $x_l = 0$, or operations O_{11} and O_{4l} if $x_l = 1$, determines the earliest possible completion time T'_l of machine $l = 2, \ldots, m-2$, and the longest path connecting operations O_{11} and O_{2l} if $x_l = 0$ or operations O_{11} and O_{3l} if $x_l = 1$ determines the earliest possible completion time T'_l of machine $l \in \{1, m-1\}$, in a schedule corresponding to $(A'_1, \ldots, A'_m, j+1)$. Parameters to be included in the sets A'_l, $l = 1, \ldots, m-1$, can easily be found from these pathes. Parameter $v'_m = v_m$ if $x_m = 0$, and $v'_m = j+1$ if $x_m = 1$.

Furthermore, calculate makespan value of the constructed schedule:

$$C(A, x) = \max\{C(A) + p_{j+1,m} + s_m x_m, T'_{m-1} + \delta\},$$

where

$$\delta = \begin{cases} P^m_{v_m, j+1} + s_m + s_m x_m, \\ \quad \text{if } x_{m-1} = 0, \text{ or } x_{m-1} = 1 \text{ and } x_m = 0, \\ s_m + p_{j+1,m}, \\ \quad \text{if } x_{m-1} = 1 \text{ and } x_m = 1. \end{cases}$$

Perform the above computations for all combinations (A, x) of a state $A = (A_1, \ldots, A_m, j)$ and an assignment vector $x = (x_1, \ldots, x_m)$ for job $j+1$.

Let $Y(A')$ denote the set of pairs (A, x) that lead to the same state $A' = (A'_1, \ldots, A'_m, j+1)$. Using the principle of optimality, we determine the minimum makespan value $C(A')$ corresponding to the state A' as

$$C(A') = \min\{C(A, x) \mid (A, x) \in Y(A')\}.$$

An optimal schedule for the problem $PFm|sum\text{–}batch, s_l|C_{\max}$, $m \geq 3$, in which job sequence $(1, \ldots, n)$ is the same for all the machines, corresponds to a state $(A_1, \ldots, A_m, n)$ with minimal value $C(A_1, \ldots, A_m, n)$ and can be found by backtracking.

Algorithm DP can be represented as an algorithm of finding a shortest path in the $n+1$-layer network, denoted as LN, where states $(A_1, \ldots, A_m, j)$ represent the vertices of the layer j and there is an arc connecting vertices A and A' of layers j and $j+1$, respectively, if state A' is obtained from the state A by a certain assignment of job $j+1$. Since every single parameter (state variable) in a state $(A_1, \ldots, A_m, j)$ can take one of the values $0, 1, \ldots, n$ and the total number of parameters is equal to $5m-7$, the number of vertices in the network LN is at most $O(n^{5m-7})$. Since the number of different assignments of job j is 2^m, the number of arcs in LN is at most $O(2^m n^{5m-7})$, which is the time complexity of Algorithm DP. For given m, this time complexity reduces to $O(n^{5m-7})$.

If the batches are restricted to be consistent, then parameters $v_2, v_3, \ldots, v_m$ that determine the batch sizes can be substituted by a single parameter v. In this case, Algorithm DP will run in $O(n^{4m-5})$ time.

Recall that Algorithm DP is designed for $m \geq 3$. If $m = 2$, then only the number of batches d on machine 1, job v that starts the last batch on machine 2 and the number of jobs j assigned to the schedule so far can be taken as state variables. Algorithm DP can be modified to solve the problem $F2|sum\text{–}batch, s_l|C_{\max}$ with a given job sequence in $O(n^3)$ time.

Algorithm DP was described for the case of non-anticipatory setups. It can easily be modified for the case of anticipatory setups. Its time complexity estimation will not change. An open question is the computational complexity of the problem with a given job sequence, when the number of machines m is variable.

3.2 An example of a problem with easy to find optimal job sequence

Consider the case when job processing times satisfy the following two assumptions: (a) they are the same on the same machine l, $2 \le l \le m-1$: $p_{jl} = p_l$, $j = 1, \ldots, n$, $l = 2, \ldots, m-1$, and (b) the jobs can be ordered such that $p_{11} \le \cdots \le p_{n1}$ and $p_{1m} \ge \cdots \ge p_{nm}$.

Statement 6 *There exists an optimal schedule for the problem* $Fm|sum\text{–}batch, s_l|C_{\max}$ *under assumptions* (a) *and* (b), *in which job sequence is the same on all the machines and it is* $(1, \ldots, n)$.

Proof Consider an optimal schedule that does not satisfy the statement of the theorem. Then there exist two jobs i and j such that job i precedes job j on some machines and $i > j$, that is, $p_{i1} \ge p_{j1}$, $p_{im} \le p_{jm}$. Interchange jobs i and j on all machines where i precedes j. The resulting schedule will be optimal. Repeat this interchange a finite number of times for all pairs of jobs that violate the statement of the theorem. □

It follows that the corresponding problem $Fm|sum\text{–}batch, s_l|C_{\max}$, $m \ge 3$, can be solved in $O(n^{5m-7})$ time by Algorithm DP in the previous section.

Statement 7 *For the problem* $PFm|sum\text{–}batch, consi, s_l|C_{\max}$ *under assumptions* (a) *and* (b), $(1, \ldots, n)$ *is an optimal job sequence.*

Proof The same interchange argument as in the proof of the previous statement can be applied to prove this statement. □

It follows from the above statement that a modification of Algorithm DP can be used to solve the problem $PFm|sum\text{–}batch, consi, s_l|C_{\max}$ under assumptions (a) and (b) in $O(n^{4m-5})$ time if $m \ge 3$. For $m = 2$, a modification of Algorithm DP will run in $O(n^3)$ time for this problem, which coincides with the running time of the algorithm of Cheng et al. (2000) derived for non-anticipatory setups $s_l = s$, $l = 1, 2$.

3.3 An iterative algorithm for $m = 2$

We now describe another polynomial time algorithm to solve the problem $PF2|sum\text{–}batch, s_l|C_{\max}$ with a given job sequence. For this problem, there exists an optimal schedule with consistent batches, see Theorem 2. Our algorithm constructs such a schedule. We first assume that setups are non-anticipatory. Let S^* and $C^*_{\max}$ denote an optimal solution and its value for this problem, respectively. We can find lower and upper bounds such that $LB \le C^*_{\max} \le UB$, where $LB = \max_{1 \le i \le n}\{s_1 + \sum_{j=1}^{i} p_{j1} + s_2 + \sum_{j=i}^{n} p_{j2}\}$ and $UB = s_1 + s_2 + \sum_{j=1}^{n}(p_{j1} + p_{j2})$.

At the upper level of our algorithm, a bisection search procedure over the range $[LB, UB]$ is performed. In the beginning, the question "Is there a batch sequence S such that $C_{\max}(S) \le C$?" is answered for $C = LB$. If the answer is "yes", then we find S, set $S^* = S$ and the procedure terminates. Otherwise, we set $V = LB$, $U = UB$ and conduct a general iteration of the bisection search by answering the above question for $C = \lfloor (V + U)/2 \rfloor$, where $\lfloor y \rfloor$ denotes the integral part of y. If the answer is "yes", then we find and keep the corresponding batch sequence, reset $U = C$, retain V unchanged and go to the next iteration of the bisection search. If the answer is "no", then we reset $V = C$, retain U unchanged and go to the next iteration of the bisection search. The procedure terminates when the length of the current interval $[V, U]$ to be partitioned is equal to zero. In this case, S^* is the last found batch sequence. It remains to provide an algorithm for solving the problem of finding a batch sequence S such that $C_{\max}(S) \le C$, $C \in [LB, UB]$. Let S be a batch sequence with minimum number of batches k among batch sequences satisfying the above inequality. It can be found by enumerating $k = 1, \ldots, n$. Let it be $S = (B_1, B_2, \ldots, B_k)$. Denote by b_i the size of the batch B_i : $|B_i| = b_i$, $i = 1, \ldots, k$.

The following algorithm ANSWER either finds the above batch sizes and, hence, the corresponding batch sequence S with k batches or determines that such a sequence does not exist. The main idea of this algorithm is to construct a feasible batch sequence by assigning as many jobs to the current batch B_i, $i = 1, \ldots, k$, as permitted by its *latest possible completion time on machine 1*. Completing batch B_i after this time on machine 1 implies completing the last job n after the threshold value C.

Assume that the batches $B_1, \ldots, B_{i-1}$ have been formed. Furthermore, assume without loss of generality that the first setup on machine 1 starts at time zero and that jobs and setups are performed with no idle time between them on machines 1 and 2. It is convenient to introduce the following notations.

- a_{i-1} is the number of jobs in the batches $B_1, \ldots, B_{i-1}$.
- $T^{(i-1)}$ is the completion time of job a_{i-1} on machine 1, i.e., $T^{(i-1)} = (i-1)s_1 + \sum_{j=1}^{a_{i-1}} p_{j1}$.
- b is the unknown number of jobs in batch B_i.

- $f(b)$ is the total setup and processing time of the batch B_i on machine 1, i.e., $f(b) = s_1 + \sum_{j=a_{i-1}+1}^{a_{i-1}+b} p_{j1}$.
- $T^{(i)}(b)$ is the completion time of job $a_{i-1} + b$ on machine 1, i.e., $T^{(i)}(b) = T^{(i-1)} + f(b)$.
- Δ_i is the latest possible completion time of batch B_i on machine 1 provided that job n completes at time C, i.e., $\Delta_i = C - (k - i + 1)s_2 - \sum_{j=a_{i-1}+1}^{n} p_{j2}$.

Note that $1 \le b \le n - (a_{i-1} + k - i)$ must be satisfied in order to ensure that each of the batches $B_i, \ldots, B_k$ includes at least one job. Given $B_1, \ldots, B_{i-1}$, the maximum possible size of the current batch B_i can be calculated from $b_i = \max\{b \mid T^{(i)}(b) \le \Delta_i\}$. Algorithm ANSWER calculates values $b_1, \ldots, b_k$. Its formal description is given below.

Algorithm ANSWER
(for given job sequence $(1, \ldots, n)$, k consistent batches and non-anticipatory setups)

Step 1. (Initialization) Set $T^{(0)} = 0$, $a_0 = 0$.

Step 2. (Recursive computation of batch sizes) For $i = 1, \ldots, k$, compute the following.

For $b = 1, \ldots, n - (a_{i-1} + k - i)$, calculate $f(b) = s_1 + \sum_{j=a_{i-1}+1}^{a_{i-1}+b} p_{j1}$ and $T^{(i)}(b) = T^{(i-1)} + f(b)$. Find $b^* = \max\{b \mid T^{(i)}(b) \le \Delta_i, b = 1, \ldots, n - (a_{i-1} + k - i)\}$.

It is easy to see that function $T^{(i)}(b)$ is increasing in b. Therefore, we can apply a bisection search in the range $1, \ldots, n - (a_{i-1} + k - i)$ of b to find b^* or determine that it does not exist in $O(\log n)$ time.

If b^* does not exist, then stop: the required batch sequence S with k batches such that $C_{\max}(S) \le C$ does not exist.

Otherwise, set $a = a_{i-1} + b^*$. Note that $a \le n - (k - i)$.

If $i < k$ and $a < n - (k - i)$, then set $b_i = b^*$, $a_i = a$ and repeat Step 2 for $i = i + 1$.

If $i \le k$ and $a = n - (k - i)$, then stop: the required batch sequence is determined by the batch sizes $(b_1, \ldots, b_{i-1}, b^*, 1, \ldots, 1)$, where $b_r = 1$, $r = i + 1, \ldots, k$.

If $i = k$ and $a < n - (k - i) = n$, then stop: the required batch sequence S with k batches such that $C_{\max}(S) \le C$ does not exist.

Theorem 3 *Algorithm* ANSWER *either finds a batch sequence S with k batches such that $C_{\max}(S) \le C$ or establishes that such a sequence does not exist.*

Proof In each iteration i of algorithm ANSWER, a partial batch schedule $(B_1, \ldots, B_i)$, $|B_r| = b_r$, $r = 1, \ldots, i$, for the jobs $1, \ldots, a_i$ is constructed such that the completion time of batch B_i on machine 2 does not exceed C minus the minimal time needed to process the remaining jobs $a_i + 1, \ldots, n$ in the remaining $k - i$ batches on machine 2 so that each of the remaining batches contains at least one job. Therefore, if algorithm ANSWER finds a batch sequence S, then $C_{\max}(S) \le C$.

Let us now consider an arbitrary batch sequence S' with minimum number of batches k such that $C_{\max}(S') \le C$. We show that S' can be transformed into the batch sequence S constructed by the algorithm ANSWER. Let $S' = (G_1, \ldots, G_k)$, where $|G_i| = g_i$, $i = 1, \ldots, k$.

Introduce empty batches $B_0 = G_0 = \phi$, which are scheduled at time zero on all the machines. Assume that $G_r = B_r$, $r = 0, 1, \ldots, i$, for some $0 \le i \le k - 2$. If $G_r = B_r$ for $r = 0, 1, \ldots, k - 1$, then $S = S'$.

Observe that $g_{i+1} \le b_{i+1}$ because b_{i+1} is the maximum possible size for the batch sequenced $(i + 1)$th in any batch sequence with k batches and batches $B_1, \ldots, B_i$ sequenced first. Assume that $g_{i+1} < b_{i+1}$. Let a_i be the number of jobs scheduled in the batches $B_1, \ldots, B_i$. In batch sequence S', move jobs $a_i + g_{i+1} + 1, a_i + g_{i+1} + 2, \ldots, \min\{a_i + b_{i+1}, a_i + g_{i+1} + g_{i+2}\}$ from batch G_{i+2} to batch G_{i+1}. The new batch sequence will be feasible because b_{i+1} is the maximum size for the batch sequenced $(i + 1)$th in any batch sequence with k batches and batches $B_1, \ldots, B_i$ sequenced first. Observe that $b_{i+1} < g_{i+1} + g_{i+2}$ because otherwise the new batch sequence will contain $k - 1$ batches and S' will not be a feasible batch sequence with minimum number of batches. Therefore, there exists a feasible batch sequence with k batches such that $G_r = B_r$, $r = 1, \ldots, i + 1$. Repeating this argument for $i = 0, 1, \ldots, k - 2$ completes the proof. □

It is easy to see that algorithm ANSWER runs in $O(k \log n)$ time. Application of this algorithm for $k = 1, \ldots, n$, will answer the question "Is there a batch sequence S such that $C_{\max}(S) \le C$?" in $O(n^2 \log n)$ time. Since this question will be asked by the bisection search procedure $O(\log(UB - LB))$ times and UB is bounded by a polynomial in $L = \max_j\{n, s_1, s_2, p_{j1}, p_{j2}\}$, the problem $PF2|sum\text{–}batch, s_l^{NA}|C_{\max}$ with a given job sequence can be solved in $O(n^2 \log n \log L)$ time.

For problem with anticipatory setups, a similar solution algorithm with the same time complexity estimation can be applied. The differences are the following. The lower bound is $LB = \max_{1 \le i \le n}\{\max\{s_1 + \sum_{j=1}^{i} p_{j1}, s_2\} + \sum_{j=i}^{n} p_{j2}\}$ and the upper bound is $UB = \max\{s_1 + \sum_{j=1}^{n} p_{j1}, s_2\} + \sum_{j=1}^{n} p_{j2}$. In algorithm ANSWER, value b^* should be found from the equation

$$b^* = \max\left\{ b \;\middle|\; T^{(i)}(b) \le \max\left\{ C - (k-i)s_2 - \sum_{j=a_{i-1}+1}^{n} p_{j2}, s_2 \right\}, b = 1, \ldots, n - (a_{i-1} + k - i) \right\}.$$

4 Consistent batches, equal processing times and equal setup times

In this section, we consider a problem in which schedules are restricted to have consistent batches, job processing times are the same: $p_{jl} = p$, $j = 1, \ldots, n$, $l = 1, \ldots, m$, and setup times are the same: $s_l = s$, $l = 1, \ldots, m$. This problem is important as itself. Let us denote it as P.

Theorem 4 *There exists an optimal schedule for the problem P that is a permutation one.*

Proof We first assume that setups are non-anticipatory. Consider an arbitrary schedule S for the problem P. Assume that $\{B_1, \ldots, B_k\}$ is the set of consistent batches in this schedule. Let $b_i = |B_i|$, $i = 1, \ldots, k$, and $b_{i^*} = \max\{b_i \mid i = 1, \ldots, k\}$. Assume without loss of generality that the batch sequence on machine 1 is $(B_1, \ldots, B_k)$. We first show that

$$C_{\max}(S) \ge \sum_{i=1}^{k}(s + pb_i) + (m-1)(s + pb_{i^*}) := L. \qquad (1)$$

Since for a schedule, in which the batches are sequenced in the order $(B_1, \ldots, B_k)$ on each machine we have $C_{\max} = L$, the correctness of inequality (1) is sufficient for the proof.

Let us represent schedule S as a network, in which a vertex (i, l) corresponds to the processing of batch B_i on machine l. With vertex (i, l), we associate its processing time $s + b_i p$. In the network, there are *l-machine chains* $((j_1^l, l), (j_2^l, l), \ldots, (j_k^l, l))$, $l = 1, \ldots, m$, where $B_{j_i^l}$ is the batch sequenced ith on machine l, and *i-batch chains* $((i, 1), (i, 2), \ldots, (i, m))$, $i = 1, \ldots, k$. Similarly to Sect. 3.1, we define the length of an oriented path in the constructed network as a summation of the processing times of the vertices in this path. The value of $C_{\max}(S)$ is determined by the length of a longest path connecting vertices $(1, 1)$ and (j_k^m, m) in the described network.

Consider a path that goes from vertex $(1, 1)$ to vertex (j_k^m, m) and includes all vertices of the i^*-batch chain. We call such a path an *i^*-path*. Note that any such path includes vertices $(1, 1), (2, 1), \ldots, (i^* - 1, 1)$. In this path, contribution of the i^*-batch chain to the $C_{\max}$ value is equal to $x := m(s + pb_{i^*})$. Contribution of the vertices $(1, 1), (2, 1), \ldots, (i^* - 1, 1)$ to the $C_{\max}$ value is equal to $y := \sum_{i=1}^{i^*-1}(s + pb_i)$. We will show that there exists an i^*-path that includes a vertex of every i-batch chain for $i = i^* + 1, \ldots, k$. Since contribution of these vertices to the $C_{\max}$ value is equal to $z := \sum_{i=i^*+1}^{k}(s + pb_i)$ and $x + y + z = L$, it is sufficient for the proof. Introduce the set of batch indices $Z = \{i \mid i = i^* + 1, \ldots, k\}$.

Let l_0 be the smallest machine index such that at least one vertex (i, l_0), $i \in Z$, precedes vertex (i^*, l_0) in the l_0-machine chain. If this index does not exist, then vertices (i, m), $i \in Z$, follow vertex (i^*, m) in the m-machine chain and we are done.

Let (i_0, l_0), $i_0 \in Z$, be a vertex which precedes vertex (i^*, l_0) and all other vertices (i, l_0), $i \in Z$, in the l_0-machine chain. Construct a partial i^*-path, denoted as P_0, that includes vertices $(i_0, l_0 - 1)$ and (i_0, l_0), all vertices between $(i^*, l_0 - 1)$ and $(i_0, l_0 - 1)$ in the $(l_0 - 1)$-machine chain and all vertices between (i_0, l_0) and (i^*, l_0) in the l_0-machine chain. Remove from the set Z all batch indices i such that $(i, l') \in P_0$ for some $l' \in \{1, \ldots, l_0\}$. If $Z = \phi$, then the path P_0 can evidently be extended to the required i^*-path.

If $Z \ne \phi$, then extend the path P_0 as follows. Let l_1 be the smallest machine index such that at least one vertex (i, l_1), $i \in Z$, precedes vertex (i^*, l_1) in the l_1-machine chain, $l_1 \ge l_0 + 1$. If this index does not exists, then vertices (i, m), $i \in Z$, follow vertex (i^*, m) in the m-machine chain and we are done.

Let $(i_1, l_1) \in Z$, $i_1 \in Z$, be a vertex which precedes vertex (i^*, l_1) and all other vertices (i, l_1), $i \in Z$, in the l_1-machine chain. Construct a partial i^*-path, denoted as P_1, that includes vertices of the path P_0, vertices $(i_1, l_1 - 1)$ and (i_1, l_1), all vertices between $(i^*, l_1 - 1)$ and $(i_1, l_1 - 1)$ in the $(l_1 - 1)$-machine chain and all vertices between (i_1, l_1) and (i^*, l_1) in the l_1-machine chain. Remove from the set Z all batch indices i such that $(i, l') \in P_1$ for some $l' \in \{l_0 + 1, \ldots, l_1\}$. If $Z = \phi$, then path P_1 can evidently be extended to the required i^*-path.

The described procedure of a i^*-path construction can be repeated until a machine index l_k, $k = 0, 1, \ldots$, exists and $Z \ne \phi$. After at most m iterations one of these two conditions will be violated and the required i^*-path will be constructed. A similar proof can be provided for the case of anticipatory setups. □

Since the jobs are identical, it follows from the above theorem that an arbitrary job sequence is optimal for the problem P. Let it be $(1, \ldots, n)$. Consider an arbitrary schedule S with job sequence $(1, \ldots, n)$ on each machine. Since the batches are consistent, this schedule is completely characterized by the number of batches k and batch sizes $b_1, \ldots, b_k$, where b_i is the size of the batch sequenced ith. Using the critical path method, calculate the makespan value of this schedule:

$$C_{\max}(S) = \max_{1\le i_1\le i_2\le\cdots\le i_{m-1}\le k}\left\{\sum_{i=1}^{i_1}(s+pb_i)+\sum_{i=i_1}^{i_2}(s+pb_i) +\cdots+\sum_{i=i_{m-1}}^{k}(s+pb_i)\right\}$$
$$= \max_{1\le i_1\le i_2\le\cdots\le i_{m-1}\le k}\left\{(k+m-1)s+pn + p(b_{i_1}+b_{i_2}+\cdots+b_{i_{m-1}})\right\}$$

for non-anticipatory setups and

$$C_{\max}(S) = \max_{1\le i_1\le i_2\le\cdots\le i_{m-1}\le k}\left\{\sum_{i=1}^{i_1}(s+pb_i)+pb_{i_1} +\sum_{i=i_1+1}^{i_2}(s+pb_i)+\cdots+pb_{i_{m-1}} +\sum_{i=i_{m-1}+1}^{k}(s+pb_i)\right\}$$
$$= \max_{1\le i_1\le i_2\le\cdots\le i_{m-1}\le k}\left\{ks+pn + p(b_{i_1}+b_{i_2}+\cdots+b_{i_{m-1}})\right\}$$

for anticipatory setups. Thus, for either non-anticipatory or anticipatory setups, the problem reduces to finding the number of batches k and corresponding batch sizes b_i, $i = 1,\ldots,k$, such that

$$\max_{1\le i_1\le i_2\le\cdots\le i_{m-1}\le k}\left\{ks+p(b_{i_1}+b_{i_2}+\cdots+b_{i_{m-1}})\right\} = ks+p\max_{1\le i_1\le i_2\le\cdots\le i_{m-1}\le k}\{b_{i_1}+b_{i_2}+\cdots+b_{i_{m-1}}\} \quad (2)$$

is minimized.

Let $b_{i^*} = b_{\max} := \max\{b_i | i = 1,\ldots,k\}$. It is easy to see that the maximum in (2) is achieved when $i_j = i^*$, $j = 1,\ldots,m-1$. Since $\sum_{i=1}^{k} b_i = n$, value $b_{\max}$ is minimized when it reaches its lower bound $\lceil n/k\rceil$. Therefore, a schedule with k batches is optimal if its batch sizes b_i^*, $i = 1,\ldots,k$, satisfy the following conditions.

$$1 \le b_i^* \le \lceil n/k\rceil, \quad i = 1,\ldots,k, \quad (3)$$

$$\sum_{i=1}^{k} b_i^* = n. \quad (4)$$

If n/k is integer, then trivially $b_i^* = n/k$, $i = 1,\ldots,k$. Otherwise, there exists an optimal schedule in which x, $x \in \{1,\ldots,k-1\}$, batches have the same size $\lfloor n/k\rfloor := a$ and $k-x$ batches have the same size $\lceil n/k\rceil = a+1$. For this schedule, condition (3) is evidently satisfied. Condition (4) can be written as $ax + (k-x)(a+1) = n$, from where we obtain $x = k(a+1) - n$. Since n/k is not integer, we have $x = k\lceil n/k\rceil - n \ge 1$ and $x \le k-1$. The latter inequality is equivalent to $(\lceil n/k\rceil - 1)k = \lfloor n/k\rfloor k \le n-1$ which is correct. Thus, we have proved

Theorem 5 *An arbitrary permutation schedule in which there are $x = k\lceil n/k\rceil - n$ batches with size $\lfloor n/k\rfloor$ and $k-x$ batches with size $\lceil n/k\rceil$ is optimal for the problem P with given number k of consistent batches.*

Mosheiov and Oron (2004) independently obtained a similar result for a special case of problem P, in which all setups are non-anticipatory and schedules are restricted to be permutation ones with consistent batches. However, they did not give the formula for calculating x.

To solve problem P, it remains to find an optimal number of batches k. It is an optimal solution to the following problem, denoted as P1:

Minimize $f(k) = ks + p(m-1)\lceil n/k\rceil$,

subject to $k \in \{1,\ldots,n\}$.

Let k^* be an optimal solution to this problem and $y^* = \lceil n/k^*\rceil$. Observe that (k^*, y^*) is one of the pairs (k_y, y) such that $y \in M_0 := \{\lceil n/k\rceil \mid k = 1,\ldots,n\}$ and $k_y = \min\{k \mid \lceil n/k\rceil = y\}$. We have $M_0 = M_1 \cup M_2$ where $M_1 = \{y \mid y \in M_0, y \ge \sqrt{n}\}$ and $M_2 = \{y \mid y \in M_0, y < \sqrt{n}\}$. Since y is an integer, $M_1 = \{y \mid y \in M_0, y \ge \lceil\sqrt{n}\rceil\}$ and $M_2 = \{y \mid y \in M_0, y \le \lceil\sqrt{n}\rceil - 1\}$.

Consider pairs (k_y, y) for $y \in M_2$. Calculate $k_1 = n$ and

$$k_y = \min\{k \mid \lceil n/k\rceil = y\} = \min\{k \mid n/y \le k < n/(y-1)\},$$
$$y = 2,\ldots,\lceil\sqrt{n}\rceil - 1.$$

Note that value k_y may not exist for some $y \in \{2,\ldots,\lceil\sqrt{n}\rceil - 1\}$. However, if it exists, then $k_y = \lceil n/y\rceil$, which follows from the above equation. Therefore,

$$(k_y, y) \in \{(\lceil n/y\rceil, y) \mid y = 1,\ldots,\lceil\sqrt{n}\rceil - 1\} \quad \text{for } y \in M_2.$$

Similarly,

$$(k_y, y) \in \left\{(k, \lceil n/k\rceil) \,\middle|\, k = 1,\ldots,\left\lceil \frac{n}{\lceil\sqrt{n}\rceil - 1}\right\rceil - 1\right\} \quad \text{for } y \in M_1,$$

because inequalities $\lceil n/k\rceil \ge \lceil\sqrt{n}\rceil$, $n/k > \lceil n/k\rceil - 1$ and the integrality of k imply $k \le \lceil \frac{n}{\lceil\sqrt{n}\rceil - 1}\rceil - 1$. From the above discussion, we deduce that

$$(k^*, y^*) \in M := \left\{\left(k, \left\lceil\frac{n}{k}\right\rceil\right) \,\middle|\, k = 1,\ldots,\left\lceil \frac{n}{\lceil\sqrt{n}\rceil - 1}\right\rceil - 1\right\} \cup \left\{\left(\left\lceil\frac{n}{y}\right\rceil, y\right) \,\middle|\, y = 1,\ldots,\lceil\sqrt{n}\rceil - 1\right\}.$$

The problem P1 can be solved by enumerating all pairs in the set M. The cardinality $|M|$ of this set is at most $\lceil \frac{n}{\lceil\sqrt{n}\rceil-1} \rceil + \lceil\sqrt{n}\rceil - 2$. Since

$$\left\lceil \frac{n}{\lceil\sqrt{n}\rceil - 1} \right\rceil \le \left\lceil \frac{n}{\sqrt{n}-1} \right\rceil \le \left\lceil \frac{n-1}{\sqrt{n}-1} + \frac{1}{\sqrt{n}-1} \right\rceil = \left\lceil \sqrt{n} + 1 + \frac{1}{\sqrt{n}-1} \right\rceil \le \lceil \sqrt{n} \rceil + 2,$$

we obtain $|M| \le 2\lceil\sqrt{n}\rceil$.

We have shown that the problem $Fm|sum\text{–}batch, consi, s_l = s, p_{jl} = p|C_{\max}$ can be solved in $O(\sqrt{n})$ time. It is an improvement over the result of Cheng et al. (2000) who derived an $O(n)$ time algorithm for the case $m = 2$ and non-anticipatory setups. Note that both algorithms are not polynomial for the considered problem because its input includes only four parameters n, m, s, and p. Finding an algorithm polynomial in $\log(\max\{n, m, s, p\})$ appears to be a non-trivial task because the function $f(k)$ may have several local minima.

5 Conclusions

We have proved that for the problem $Fm|sum\text{–}batch, s_l|C_{\max}$ with anticipatory or non-anticipatory setups there exists an optimal schedule for which job sequences are the same on machines 1 and 2 and they are the same on machines $m-1$ and m, $m \ge 3$, irrespectively of whether the batches are restricted to be consistent or not. It follows that a permutation schedule is optimal for this problem when $m = 3$. For $m \ge 4$, there are examples where all optimal schedules are non-permutation ones. We have shown that there exists an optimal schedule for the problem $F2|sum\text{–}batch, s_l|C_{\max}$, which is a permutation one with consistent batches. Examples have been derived where all optimal schedules have inconsistent batches for $m \ge 3$. The problem $F2|sum\text{–}batch, consi, s_l|C_{\max}$ with k consistent and given batches has been shown to be solvable in $O(k \log k)$ time by known algorithms. A dynamic programming algorithm with $O(n^{5m-7})$ running time has been developed for the problem $PFm|sum\text{–}batch, s_l|C_{\max}$, in which the same job sequence is given for all the machines and $m \ge 3$. For $m = 2$, it can be modified to run in $O(n^3)$ time. The algorithm can be applied to solve a special case when job processing times are the same on each machine $l = 2, \ldots, m-1$ and they are oppositely ordered on machines 1 and m. An iterative algorithm with $O(n^2 \log n \log L)$ running time, where L is the maximum numerical parameter, has been developed for the problem $PF2|sum\text{–}batch, s_l|C_{\max}$ with the same given job sequence on both machines. An $O(\sqrt{n})$ time algorithm has been presented to solve the problem with arbitrary number of machines and consistent batches, when job processing times are all equal and setup times are all equal. Our algorithms improve the existing algorithms of Cheng et al. (2000) suggested for more restrictive cases.

Below we pose three open questions that are interesting for future research.

(1) Is there an optimal schedule which is the one with consistent batches for the problem with m machines, equal setup times and equal job processing times? In Sect. 2, we gave a positive answer to this question for the case $m = 2$.
(2) What is the computational complexity of the problem $F|sum\text{–}batch, consi, s_l = s, p_{jl} = p|C_{\max}$? Recall that our $O(\sqrt{n})$ algorithm is pseudopolynomial for this problem.
(3) What is the computational complexity of the problem with the same given job sequence for variable $m \ge 3$?

Acknowledgements This research was supported by The Hong Kong Polytechnic University for Area of Strategic Development under grant number A628. M.Y. Kovalyov was additionally supported by INTAS (Project 03-51-5501).

References

Bukchin, J., & Masin, M. (2004). Multi-objective lot splitting for a single product m-machine flow shop line. *IIE Transactions*, *36*, 191–202.

Bukchin, J., Tzur, M., & Jaffe, M. (2002). Lot splitting to minimize average flow-time in a two machine flow shop. *IIE Transactions*, *34*, 953–970.

Cheng, T. C. E., Lin, B. M. T., & Toker, A. (2000). Makespan minimization in the two-machine flowshop batch scheduling problem. *Naval Research Logistics*, *47*, 128–144.

Chin, H. N., & Chang, J. H. (2005, to appear). Cost models for lot streaming in a multistage flow shop, *Omega*.

Glass, C. A., Potts, C. N., & Strusevich, V. A. (2001). Scheduling batches with sequential job processing for two-machine flow and open shops. *INFORMS Journal on Computing*, *13*, 120–137.

Huq, F., Cutright, K., & Martin, C. (2004). Employee scheduling and makespan minimization in a flow shop with multiprocessor work stations: a case study. *Omega*, *32*, 121–129.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, *1*, 61–68.

Kalir, A. A., & Sarin, S. C. (2000). Evaluation of potential benefits of lot streaming in flow shop systems. *International Journal of Production Economics*, *66*, 131–142.

Li, C.-L., & Xiao, W.-Q. (2004). Lot streaming with supplier-manufacturer coordination. *Naval Research Logistics*, *51*, 522–542.

Mosheiov, G., & Oron, D. (2004). A note on flow-shop and job-shop batch scheduling with identical processing-time jobs. *European Journal of Operational Research*, *161*, 285–291.

Mosheiov, G., Oron, D., & Ritov, Y. (2004). Flow-shop batch scheduling with identical processing-time jobs. *Naval Research Logistics*, *51*, 783–799.

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

Potts, C. N., & Van Wassenhove, L. N. (1992). Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of the Operational Research Society*, *43*, 395–406.

Tanaev, V. S., Kovalyov, M. Y., & Shafransky, Y. M. (1998). *Scheduling theory. Group technologies*. Minsk: IEC NASB (in Russian).

Tanaev, V. S., Gordon, V. S., & Shafransky, Y. M. (1994). *Scheduling theory. One-stage systems*. Dordrect: Kluwer Academic.

Tanaev, V. S., Sotskov, Y. N., & Strusevich, V. A. (1995). *Scheduling theory. Multi-stage systems*. Dordrecht: Kluwer Academic.

Trietsch, D., & Baker, K. R. (1993). Basic techniques for lot streaming. *Operations Research*, *41*, 1065–1076.

Webster, S., & Baker, K. R. (1995). Scheduling groups of jobs on a single machine. *Operations Research*, *43*, 692–703.