On a borderline between the NP-hard and polynomial-time solvable cases of the flow shop with job-dependent storage requirements

Alexander Kononov · Julia Memar · Yakov Zinder

Received: 15 January 2020 / Accepted: 17 September 2021 / Published online: 13 October 2021
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract
The paper is concerned with the two-machine flow shop, where each job requires an additional resource (referred to as storage space) from the start of its first operation till the end of its second operation. The storage requirement of a job is determined by the processing time of its first operation. At any point in time, the total consumption of this additional resource cannot exceed a given limit (referred to as the storage capacity). The goal is to minimise the makespan, i.e. to minimise the time needed for the completion of all jobs. This problem is NP-hard in the strong sense. The paper analyses how the parameter - a lower bound on the storage capacity specified in terms of the processing times, affects the computational complexity.

Keywords Flow shop · Computational complexity · Makespan · Job-dependent storage requirements

1 Introduction

This paper considers the two-machine flow shop that is comprised of the first-stage machine $M_1$, the second-stage machine $M_2$, and an additional resource referred to as a buffer or storage space. The machines must process a set of jobs $N = \{1, \ldots, n\}$. In order to be processed a job $j \in N$ must be processed on the machine $M_1$ for $a_j$ units of time and, after that, on the machine $M_2$ for $b_j$ units of time. Once a machine starts to process a job, it continues processing till the completion of this operation. Each machine can process at most one job...
at a time. For each job, the processing on $M_2$ can commence only after the completion of the processing of this job on $M_1$. Each job $j$ seizes $\omega(j)$ units of storage space (buffer) at the beginning of its processing on $M_1$ and releases this portion of storage space only at the completion of the job’s processing on $M_2$. Similar to [14–16,20,21] it is assumed that, for each $j \in N$, $\omega(j) = a_j$. At any point in time, the total consumption of the storage space cannot exceed $\Omega$ - the storage capacity.

The processing of jobs commences at time $t = 0$. A schedule $\sigma$ specifies for each $j \in N$ and each machine $M_i$, the point in time $S^i_j(\sigma)$ when this job commences its processing on $M_i$. Since each operation is processed without interruptions, for any $j \in N$, the completion times of job $j$ on $M_1$ and $M_2$ are

$$C^1_j(\sigma) = S^1_j(\sigma) + a_j \quad \text{and} \quad C^2_j(\sigma) = S^2_j(\sigma) + b_j,$$

respectively. The goal is to minimise the makespan

$$C_{\text{max}}(\sigma) = \max_{j \in N} C^2_j(\sigma).$$

Following [14], the considered problem will be referred to as PP-problem.

The flow shop problems with a buffer have been extensively studied in the literature on scheduling, but most of these publications consider flow shops with an intermediate buffer between stages and assume that this buffer limits only the number of jobs that have completed one operation and are waiting for the commencement of the next one [5–7,23,24]. In contrast, in the scheduling problem, considered in this paper, the storage (buffer) requirement varies from job to job and each job seizes the required storage space during its processing on the machines as well as during its waiting time between the operations. This also differentiates this paper from the publications on resource constrained scheduling that assumes that the additional resource is consumed by a job only during its processing on the machines but it is released between the operations [4,6].

Although the practical significance of the flow shops with job-dependent storage requirements where a job uses this resource not only during its processing on the machines but also between the operations was acknowledged as early as in [25], the active research in this field of scheduling theory was triggered by the publications [21] and [19] which were concerned with controlling the lag during a multimedia presentation of media objects, where each object has a loading time and a play time. An object’s presentation cannot be started until it has finished loading. To avoid the potential presentation delay that occurs while waiting for loading, the player is designed to “prefetch” the objects prior to their intended presentation time. For applications that use portable devices such as PDAs to play multimedia objects, one must consider the memory size allowed for storing the objects. Under a network environment with a rather stable transmission rate, the download time of an object is proportional to its file size. Since, we can scale the file size in such a way that a unit of size is loaded per a unit of time. So, without loss of generality, we can assume that the download time of an object is equal to the size of its file. This leads to the model where the duration of the operation on the first-stage machine is equal to the storage space requirement.

The two-machine flow shop with job-dependent storage requirements, considered in this paper, was used in the study on data gathering in networks with star topology [1,2]. Such a network is comprised of a number of worker nodes, each transmitting its data set to the base station where this data set is to be processed. The base station can communicate with at most one worker node at a time and allocates to a data set the entire required memory from the beginning of its transmission till the end of its processing by the station.
The practical applications of the considered flow shop with job-dependent storage requirements can be found far beyond the confines of the systems where storage is the system computer memory. Thus, the interest of one of the coauthors of this paper in the PP-problem was originally inspired by the scheduling situations in supply chains where the change of the means of transportation involves unloading and loading, using certain storage space. Such situations arise, for example, in a supply chain of mineral resources [9].

One of the key postulates of the PP-problem is the assumption that the storage space, required by a job, is determined by the processing time of its first operation. As it has been discussed above, this assumption is justified by several applications. On the other hand, it is known that, in manufacturing, the condition that the storage requirement is determined by the duration of one of the operations may be inadequate [25]. The generalisation of the PP-problem where the storage requirement of a job is not necessarily determined by the processing time of one of its operations was considered in [26]. Other variations and generalisations of the PP-problem, considered in the literature, include the relaxation of the assumption that the storage capacity is a constant [3]; the replacement of the assumption that there exists only one pair of machines by the assumption that each job can be processed by one of several disjoint pairs of machines, each pair being assigned a storage capacity, which varies from pair to pair [8]; a flexible flow shop with batch processing [12]; as well as different objective functions.

The PP-problem is NP-hard in the strong sense [19]. The problem remains NP-hard in the strong sense even under the restriction that, on one of the machines, the jobs are to be processed in a given sequence [11]. Furthermore, it has been proven in [10] that there are instances for which the set of all optimal schedules does not contain a permutation schedule, that is, a schedule in which both machines process the jobs in the same order. Even the decision problem, requiring an answer to the question of whether or not the given instance is one of the instances that do not have an optimal schedule that is a permutation one, is NP-complete.

The computational complexity of the PP-problem motivated interest in branch-and-bound algorithms [14,19,20] as well as in integer programming-based and metaheuristic optimisation methods [16,18]. Another direction of research, triggered by the computational complexity of the PP-problem, was the study of its various particular cases [2,15,17,22,26]. Thus, [15] presents a polynomial-time algorithm and a proof that this algorithm constructs an optimal schedule for any instance of the PP-problem where the storage capacity is not less than five times the maximal processing time. This result raised the question of how the computational complexity changes with the variation of a lower bound on the storage capacity computed as a function of the processing times.

The remainder of this paper addresses this question. Section 2 presents a proof that the PP-problem remains NP-hard after the introduction of the additional assumption that, for any \( \delta > 0 \),

\[
\frac{4}{1 + \delta} \max_{i \in N} \{a_i, b_i\} \leq \Omega. \tag{1}
\]

Section 3 presents a polynomial-time algorithm and a proof that this algorithm constructs an optimal schedule if

\[
3.5 \max_{i \in N} a_i + \max_{i \in N} \left\{0.5 \max_{i \in N} a_i, \max_{i \in N} b_i\right\} \leq \Omega.
\]
Observe that these results not only contribute to the efforts aimed at establishing the borderline between the NP-hard and polynomial-time solvable, but also significantly strengthen the result in [15] by replacing the factor 5 in the lower bound in [15] by 4.5.

### 2 NP-hardness

We show that the PP-problem remains NP-hard even if (1) holds. We use the following NP-hard variant of the partition problem. The input consists of $2m$ positive integers $e_1, e_2, \ldots, e_{2m}$ such that

$$\sum_{k=1}^{2m} e_k = 2E \text{ and } E \frac{m+1}{m+1} < e_k < E \frac{m-1}{m-1}, \quad k = 1, \ldots, 2m.$$ 

It is necessary to answer the following question: does there exist an index set $K$, such that $\sum_{k \in K} e_k = E$ holds? Without lost of generality we assume that

$$m \geq \max\left\{2\left(1 + \frac{1}{\delta}\right), 8\right\}. \quad (2)$$

Consider the following instance of the PP-problem with $2m+4$ jobs. We have $2m$ regular jobs. For $j = 1, \ldots, 2m$ a regular job $j$ has processing times $a_j = E$ and $b_j = E + e_j$, and $a_0 = 0$ and $b_0 = E$ for the job 0. There are three special jobs $2m+1, 2m+2, \text{ and } 2m+3$. For each special job $j$, let $a_j = E$ and $b_j = 0$. The buffer capacity $\Omega = 4E(1 - \frac{1}{(m-1)}) < 4E$. We are going to check whether or not there exists a feasible schedule with makespan $C_{\text{max}} \leq (2m+3)E$.

It is easy to see that no four jobs fit in the buffer unless the job 0 is one of the jobs. On the other hand, any three jobs can be in the buffer at the same time. Indeed, taking into account (2), we have

$$a_i + a_j + a_k < 3E + \frac{3E}{m-1} = 3E + \frac{7E}{m-1} - \frac{4E}{m-1} \leq 4E \left(1 - \frac{1}{(m-1)}\right) = \Omega.$$ 

Furthermore, all jobs $0 \leq i \leq 2m+3$, satisfy the condition (1):

$$\max_{i \in N} \{a_i, b_i\} \leq E + \frac{E}{m-1} = E \left(\frac{m}{m-1} \times \frac{m-2}{m-2}\right) = E \left(\frac{m-2}{m-1} \times \frac{m}{m-2}\right)$$

$$= E \left(1 - \frac{1}{m-1}\right) \left(1 + \frac{2}{m-2}\right) \leq \frac{\Omega}{4} \left(1 + \delta\right).$$

Let $\sigma$ be a schedule with makespan $C_{\text{max}} \leq (2m+3)E$. We note that the load on both machines is equal to $(2m+3)E$. Thus, both machines are not idle during the interval $[0, (2m+3)E]$. It follows that 0 is the first job in $\sigma$, and the job sequences on the both machines are the same. Observe that the machine $M_2$ works without idle time if and only if for each job $i$ its completion time $C_i^1(\sigma)$ does not exceed the completion time of the job which precedes it on $M_2$. We note that $a_j = E$ for all jobs except the job 0, and it follows that for all jobs in $\sigma$ except for the last job, $C_i^2(\sigma) - C_i^1(\sigma) \geq E$.

Without loss of generality we assume that the job $2m+1$ is the first special job in $\sigma$. Assume that job $i > 0$ immediately precedes the job $2m+1$.

If $C_i^2(\sigma) - C_i^1(\sigma) < 2E$, then $C_{2m+1}^2(\sigma) - C_{2m+1}^1(\sigma) < E$, which implies that the machine $M_2$ is idle and hence $C_{\text{max}}(\sigma) > (2m+3)E$. Suppose that $C_i^2(\sigma) - C_i^1(\sigma) > 2E$. The buffer capacity $\Omega = 4E(1 - \frac{1}{(m-1)}) < 4E$. Without loss of generality we assume that the job $2m+1$ is the first special job in $\sigma$. Assume that job $i > 0$ immediately precedes the job $2m+1$.

If $C_i^2(\sigma) - C_i^1(\sigma) < 2E$, then $C_{2m+1}^2(\sigma) - C_{2m+1}^1(\sigma) < E$, which implies that the machine $M_2$ is idle and hence $C_{\text{max}}(\sigma) > (2m+3)E$. Suppose that $C_i^2(\sigma) - C_i^1(\sigma) > 2E$. The buffer capacity $\Omega = 4E(1 - \frac{1}{(m-1)}) < 4E$. Without loss of generality we assume that the job $2m+1$ is the first special job in $\sigma$. Assume that job $i > 0$ immediately precedes the job $2m+1$.

If $C_i^2(\sigma) - C_i^1(\sigma) < 2E$, then $C_{2m+1}^2(\sigma) - C_{2m+1}^1(\sigma) < E$, which implies that the machine $M_2$ is idle and hence $C_{\text{max}}(\sigma) > (2m+3)E$.
Since the machine $M_1$ processes the jobs without idle time, then there are at least four jobs $j$ with $a_j = E$ in the buffer during the interval $[C^1_1(\sigma) + 2E, C^2_1(\sigma)]$, which violates the buffer capacity. Thus, $C^2_1(\sigma) - C^1_1(\sigma) = 2E$.

Denote by $K'$ the set of jobs that precede the job $2m + 1$ in $\sigma$. We have

$$2E = \sum_{i \in K'} b_i - \sum_{i \in K'} a_i = b_0 + \sum_{i \in K} e_i = E + \sum_{i \in K} e_i.$$ 

Therefore, a feasible schedule with makespan $C_{max} \leq (2m + 3)E$ exists if and only if for $2m$ positive integers $e_1, e_2, \ldots, e_{2m}$ such that

$$\sum_{k=1}^{2m} e_k = 2E \text{ and } \frac{E}{m+1} < e_k < \frac{E}{m-1}, \quad k = 1, \ldots, 2m,$$

there exists an index set $K$ such that $\sum_{i \in K} e_i = E$.

### 3 Polynomial-time algorithm

The PP-problem is strongly NP-hard [21]. However, the computational complexity of the problem depends on the relationship between the size of jobs and the size of the buffer. Indeed, the problem is easily solvable if the buffer size is large enough, for example, when all jobs can be simultaneously placed in the buffer. In this case, the problem is equivalent to the two-machine flow shop problem without buffer and it can be solved in $O(n \log n)$ time by Johnson’s algorithm [13]. Johnson’s rule can be stated as follows:

- partition $N$ into two sets:
  
  \[ L_1 = \{ i \in N : a_i < b_i \} \quad \text{and} \quad L_2 = \{ i \in N : a_i \geq b_i \}; \]

- first schedule the jobs from $L_1$ in non-decreasing order of $a_i$, and then schedule the jobs from $L_2$ in non-increasing order of $b_i$.

Denote by $\sigma^J$ the schedule constructed by Johnson’s rule. The makespan $C_{max}(\sigma^J)$ is a lower bound on the objective value of the PP-problem. Let

$$a_{max} = \max_{i \in N} a_i \quad \text{and} \quad b_{max} = \max_{i \in N} b_i.$$ 

The polynomial-time algorithm below constructs a schedule for an extended set of jobs obtained by adding some auxiliary jobs. The set of these auxiliary jobs is determined by schedule $\sigma^J$. It is shown that if

$$3.5a_{max} + \max\{0.5a_{max}, b_{max}\} \leq \Omega,$$

then the makespan of this schedule is equal to $C_{max}(\sigma^J)$ which implies its optimality.

Number the jobs according to the sequence constructed by Johnson’s rule, then

$$C_{max}(\sigma^J) = \max_k \left( \sum_{i=1}^{k} a_i + \sum_{i=k}^{n} b_i \right).$$ 

Denote by $Idle_1$ and $Idle_2$ the total idle time in the interval $[0, C_{max}(\sigma^J)]$ on machines $M_1$ and $M_2$, respectively. If the maximum in (4) is obtained for $k = k'$, then $Idle_1 = C_{max}(\sigma^J) - \sum_{i=1}^{n} a_i \leq \sum_{i=k'}^{n} b_i$ and $Idle_2 = C_{max}(\sigma^J) - \sum_{i=1}^{n} b_i \leq \sum_{i=1}^{k'} a_i$. 
Let $X$ and $Y$ be two sets of auxiliary jobs of cardinality
\[ |X| = \left\lceil \frac{\text{Idle}_2}{b_{\text{max}}} \right\rceil \quad \text{and} \quad |Y| = \left\lceil \frac{\text{Idle}_1}{a_{\text{max}}} \right\rceil \]
and such that
\[ a_i = \begin{cases} 0, & \text{if } i \in X \\ \frac{\text{Idle}_1}{|Y|}, & \text{if } i \in Y \end{cases} \quad \text{and} \quad b_i = \begin{cases} \text{Idle}_2, & \text{if } i \in X \\ \frac{|X|}{|Y|}, & \text{if } i \in Y \\ 0, & \text{if } i \in Y \end{cases} . \]
Observe that
\[ \sum_{i \in N \cup X \cup Y} a_i = \sum_{i \in N \cup X \cup Y} b_i, \tag{5} \]
and that $a_i \leq a_{\text{max}}$ and $b_i \leq b_{\text{max}}$ for any job $i \in N \cup X \cup Y$. Consider a permutation schedule (a permutation schedule is a schedule with the same order of jobs on both machines) $\sigma'$ where the jobs of the set $N' = N \cup X \cup Y$ are scheduled as follows. The first $|X|$ jobs are the jobs constituting $X$, sequenced in arbitrary order; these jobs are followed by all jobs in $N$, scheduled according to Johnson’s rule; the jobs from $N$ are followed by the arbitrary ordered remaining jobs, i.e. the jobs constituting $Y$. It is easy to see that
\[ C_{\text{max}}(\sigma') = C_{\text{max}}(\sigma_J) \]
and that $\sigma'$ can be viewed as a result of the application of Johnson rule to $N'$. Let $\pi'$ be the permutation of all jobs in $N'$ induced by the order in which these jobs are processed in $\sigma'$ and let $\pi_0$, $\pi_1$ and $\pi_2$ be the permutations of the jobs in
\[ \begin{align*}
L_0' &= X \cup \left\{ i \in L_1 \mid a_i \leq \frac{1}{2} a_{\text{max}} \right\}, \\
L_1' &= \left\{ i \in L_1 \mid a_i > \frac{1}{2} a_{\text{max}} \right\}, \\
L_2' &= L_2 \cup Y,
\end{align*} \]
respectively, induced by $\pi'$. The algorithm below constructs a permutation $\pi$ that specifies the order in which the jobs are processed in an optimal permutation schedule. It is convenient to use the following notation:
\[ \Delta_j = b_j - a_j, \quad \text{for all } j \in N'; \]
\[ \mu(k) = \sum_{j=1}^{\lfloor k \rfloor \lfloor L_0' \rfloor} \Delta_{\pi_0(j)}, \quad \text{for all } 1 \leq k \leq |L_0'|; \]
\[ n' = n + |X| + |Y|; \]
\[ l_{k,1}(\pi) = \sum_{j=1}^{\lfloor k \rfloor \lfloor L_0' \rfloor} a_{\pi(j)}, \quad \text{for all } 1 \leq k \leq n'; \]
\[ l_{k,2}(\pi) = \sum_{j=1}^{\lfloor k \rfloor \lfloor L_0' \rfloor} b_{\pi(j)}, \quad \text{for all } 1 \leq k \leq n'; \]
\[ R_k(\pi) = l_{k,2}(\pi) - l_{k,1}(\pi), \quad \text{for all } 1 \leq k \leq n'. \]
Let $\pi = \emptyset$ indicate that the permutation $\pi$ is not specified.
Algorithm 1

1: Set \(i = 1, i_0 = 1, i_1 = 1, i_2 = 1, \pi = \emptyset, R_0(\pi) = 0, l_{0,1} = 0, l_{0,2} = 0\).
2: while \(i_0 \leq |L'_0|\) do
3: \(\text{if } (R_{i-1}(\pi) < 2a_{\text{max}} \text{ and } R_{i-1}(\pi) + \Delta_{\pi_2(i_2)} + \mu(i_0) < a_{\text{max}}) \text{ or } R_{i-1}(\pi) < \frac{3}{2}a_{\text{max}} \) then
4: set \(\pi(i) = \pi_0(i_0), i_0 = i_0 + 1;\)
else
6: set \(\pi(i) = \pi_2(i_2), i_2 = i_2 + 1;\)
end if
8: set \(S^1_{\pi(i)}(\sigma) = l_{i-1,1}(\pi)\) and \(S^2_{\pi(i)}(\sigma) = l_{i-1,2}(\pi);\) set \(i = i + 1;\)
end while
10: while \(i \leq n'\) do
11: \(\text{if } R_{i-1}(\pi) < 2a_{\text{max}} \text{ and } i_1 \leq |L'_1| \) then
12: set \(\pi(i) = \pi_1(i_1), i_1 = i_1 + 1;\)
else
14: set \(\pi(i) = \pi_2(i_2), i_2 = i_2 + 1;\)
end if
16: set \(S^1_{\pi(i)}(\sigma) = l_{i-1,1}(\pi)\) and \(S^2_{\pi(i)}(\sigma) = l_{i-1,2}(\pi);\) set \(i = i + 1;\)
end while
18: return schedule \(\sigma.\)

Lemma 1 Algorithm 1 schedules all jobs in \(N'.\)

Proof Taking into account the condition in line 2, the first while loop (line 2–9) schedules all jobs in \(L'_0\) (and probably some jobs in \(L'_2\)) if, each time when line 4 is to be executed, there exists an unscheduled job from \(L'_0\) and, each time when line 6 is to be executed, there exists an unscheduled job from \(L'_2\). Observe that the condition in line 2 guarantees this for line 4.

As far as the remaining jobs are concerned, by virtue of the condition in line 10, they all will be scheduled by the second while loop (lines 10–17) if, each time when line 12 is to be executed, there exists an unscheduled job from \(L'_1\) and, each time when line 14 is to be executed, there exists an unscheduled job from \(L'_2\). Observe that the condition in line 11 guarantees this for line 12 whereas the condition in line 10 together with the equality

\[i_1 = |L'_1| + 1\]  

(6)
guarantees this for line 14.

If the execution of line 14 is needed and (6) does not hold, then, for the corresponding \(i, R_{i-1}(\pi) > 0.\) Such inequality also holds each time when the execution of line 6 is needed. Since, for any \(1 \leq i \leq n',\)

\[R_{i-1}(\pi) < R_i(\pi), \text{ if } \pi(i) \in L'_0 \cup L'_1\]  

(7)
\[R_{i-1}(\pi) \geq R_i(\pi), \text{ if } \pi(i) \in L'_2,\]  

(8)

and since (5) implies that \(R_{n'}(\pi) = 0,\) the inequality \(R_{i-1}(\pi) > 0\) implies the existence of an unscheduled job from \(L'_2.\)

Theorem 1 If (3) holds, then Algorithm 1 constructs an optimal schedule.

Proof The schedule \(\sigma,\) constructed by Algorithm 1, is feasible if

(a) the schedule \(\sigma\) has no overlapping jobs on the same machine;
(b) the schedule \(\sigma\) has no overlapping operations of the same job;
(c) the schedule \(\sigma\) does not violate the buffer constraint.  

\[\square\]
(a) This condition holds because, for each \( j \geq 2 \),
\[
S_{\pi(j)}^{1}(\sigma) = l_{j-1,1}(\pi) = l_{j-2,1}(\pi) + a_{\pi(j)} = S_{\pi(j-1)}^{1}(\sigma) + a_{\pi(j-1)} = C_{\pi(j-1)}^{1}(\sigma),
\]
\[
S_{\pi(j)}^{2}(\sigma) = l_{j-1,2}(\sigma) = l_{j-2,2}(\sigma) + b_{\pi(j-1)} = S_{\pi(j-1)}^{2}(\sigma) + b_{\pi(j-1)} = C_{\pi(j-1)}^{2}(\sigma).
\]

(b) In what follows, we will consider a few cases to show that there are no operations overlapping in schedule \( \sigma \). According to the algorithm any job from \( L_{0}' \) precedes any job from \( L_{1}' \) in the permutation \( \pi \). Assume that
\[
\pi(|L_{0}'| + |L_{1}'| + 1) = \pi(1).
\]
The equality above implies that all jobs from \( L_{0}' \) succeed any job from \( L_{1}' \) in the permutation \( \pi \). Then it follows that the permutation \( \pi \) is Johnson’s permutation and the schedule \( \sigma \) coincides with the schedule \( \sigma' \). The feasibility \( \sigma' \) implies that the schedule \( \sigma \) has no overlapping operations of the same job.

If (9) does not hold, then in the permutation \( \pi \) the job \( \pi_{2}(1) \) is before the job \( \pi_{1}(|L_{1}'|) \), i.e. \( \pi(i) = \pi_{2}(1) \) for some \( i < |L_{0}'| + |L_{1}'| + 1 \). Thus either there exists \( k \leq |L_{0}'| \) such that
\[
R_{k-1}(\pi) \geq \frac{3}{2}a_{\max} \quad \text{and} \quad R_{k-1}(\pi) + \Delta_{\pi_{2}(1)} + \mu(k) \geq a_{\max},
\]
or there exists \( k ' , k \leq |L_{0}'| + |L_{1}'| \) such that \( R_{k-1}(\pi) > 2a_{\max} \).

Let \( \pi(h_{0}) \) be the last job from \( L_{0}' \) and \( \pi(h_{1}) \) be the last job from \( L_{1}' \) in the permutation \( \pi \), i.e., \( \pi(h_{0}) = \pi_{0}(|L_{0}'|) \) and \( \pi(h_{1}) = \pi_{1}(|L_{1}'|) \). Note that \( k < h_{1} \), as otherwise we would have (9) satisfied. Partition the permutation \( \pi \) into three subsequences: \( (\pi(1), \ldots, \pi(k)) \), \( (\pi(k), \ldots, \pi(h_{1})) \), and \( (\pi(h_{1} + 1), \ldots, \pi(n')) \).

Case 1(b) \( 1 \leq j \leq k - 1 \). The first \( k - 1 \) jobs in the \( \pi \) are from the set \( L_{0}' \cup L_{1}' \), hence \( \sigma \) and \( \sigma' \) are the same for the first \( k - 1 \) jobs. Consequently, for \( j = 1, \ldots, k - 1 \), the operations of job \( \pi(j) \) do not overlap.

Case 2(b) \( k \leq j \leq h_{0} \). Observe that this case is possible only if \( k \leq |L_{0}'| \). Consider two sub-cases, which are defined by the set the job \( \pi(j) \) belongs to: either \( \pi(j) \in L_{2}' \) or \( \pi(j) \in L_{0}' \).

Case 2.1(b) \( \pi(j) \in L_{2}' \). According to the algorithm either \( R_{j-1}(\pi) > 2a_{\max} \) or \( R_{j-1}(\pi) \geq \frac{3}{2}a_{\max} \) and \( R_{j-1}(\pi) + \Delta_{\pi(j)} + \mu(i_{0}) \geq a_{\max} \), where \( i_{0} \) is specified according to Algorithm 1. Thus, we have
\[
R_{j-1}(\pi) \geq \min\{2a_{\max}, a_{\max} - \Delta_{\pi(j)} - \mu(i_{0})\}.
\]
We have
\[
C_{\pi(j)}^{1}(\sigma) = S_{\pi(j)}^{1}(\sigma) + a_{\pi(j)} = l_{j-1,1}(\pi) + a_{\pi(j)} = l_{j-1,2}(\pi) + a_{\pi(j)} - R_{j-1}(\pi)
\]
\[
\leq l_{j-1,2}(\pi) + a_{\pi(j)} - \frac{3}{2}a_{\max} \leq l_{j-1,2}(\pi) = S_{\pi(j)}^{2}(\sigma).
\]
Thus, the operations of the job \( \pi(j) \) do not overlap. Moreover, for the \( \pi(j) \in L_{2}' \) the following inequality holds:
\[
R_{j}(\pi) = R_{j-1}(\pi) + b_{j} - a_{j} \geq \frac{3}{2}a_{\max} - a_{\max} \geq \frac{1}{2}a_{\max}.
\]
Taking into account (7), we have that \( R_{j}(\pi) \geq \frac{1}{2}a_{\max} \) for all \( k + 1 \leq j \leq h_{0} \).
Case 2.2(b) \( \pi(j) \in L'_0 \). We have
\[
C^1_{\pi(j)}(\sigma) = S^1_{\pi(j)}(\sigma) + a_{\pi(j)} = l_{j-1,1}(\pi) + a_{\pi(j)} = l_{j-1,2}(\pi) + a_{\pi(j)} - R_{j-1}(\pi) \\
\leq l_{j-1,2}(\pi) + a_{\pi(j)} - \frac{1}{2}a_{\text{max}} \leq l_{j-1,2}(\pi) = S^2_{\pi(j)}(\sigma).
\]
The last inequality follows from the fact that \( a_i \leq \frac{1}{2}a_{\text{max}} \) for any \( i \in L'_0 \). Thus, the operations of the job \( \pi(j) \) do not overlap.

Let \( \pi(h_2) < \pi(h_0) \) be the last job from \( L'_2 \) that precedes the job \( \pi(h_0) \) in the permutation \( \pi \). Thus, the jobs \( \pi(h_2 + 1), \ldots, \pi(h_0) \) belong to \( L'_0 \). We have
\[
R_{h_0}(\pi) = R_{h_2-1}(\pi) + \Delta(\pi) + \sum_{i=h_2+1}^{h_0} \Delta_{\pi(i)} \\
= R_{h_2-1}(\pi) + \Delta(\pi) + \mu(h'),
\]
where \( h' = |L'_0| + 1 - h_0 + h_2 \) is the position of the job \( \pi(h_2 + 1) \) in the permutation \( \pi_0 \).

Taking into account (10) we obtain
\[
R_{h_0}(\pi) \geq \min\{2a_{\text{max}} + \Delta(\pi), a_{\text{max}} \} \geq a_{\text{max}}.
\]

Case 3(b) \( h_0 + 1 \leq j \leq h_1 \). Consider two sub-cases, which are defined by the set the job \( \pi(j) \) belongs to: either \( \pi(j) \in L'_2 \) or \( \pi(j) \in L'_1 \).

Case 3.1(b) \( \pi(j) \in L'_2 \). Due to step 11 of the algorithm, we have \( R_{j-1}(\pi) \geq 2a_{\text{max}} \). Thus we get
\[
C^1_{\pi(j)}(\sigma) = S^1_{\pi(j)}(\sigma) + a_{\pi(j)} = l_{j-1,1}(\pi) + a_{\pi(j)} \\
= l_{j-1,2}(\pi) + a_{\pi(j)} - R_{j-1}(\pi) \\
\leq l_{j-1,2}(\pi) + a_{\pi(j)} - 2a_{\text{max}} \leq S^2_{\pi(j)}(\sigma).
\]
Thus, the operations of the job \( \pi(j) \) do not overlap. Moreover,
\[
R_{j}(\pi) = R_{j-1}(\pi) + b_j - a_j \geq 2a_{\text{max}} - a_{\text{max}} \geq a_{\text{max}}.
\]
Taking into account (7) and (11), we have \( R_{j}(\pi) \geq a_{\text{max}} \) for all \( h_0 \leq j \leq h_1 \).

Case 3.2(b) \( \pi(j) \in L'_1 \). We have
\[
C^1_{\pi(j)}(\sigma) = S^1_{\pi(j)}(\sigma) + a_{\pi(j)} = l_{j-1,1}(\pi) + a_{\pi(j)} \\
= l_{j-1,2}(\pi) + a_{\pi(j)} - R_{j-1}(\pi) \\
\leq l_{j-1,2}(\pi) + a_{\pi(j)} - a_{\text{max}} \leq S^2_{\pi(j)}(\sigma).
\]
Thus, the operations of the job \( \pi(j) \) do not overlap.

Case 4(b) \( h_1 + 1 \leq j \leq n' \). Finally, we observe that the \( h_1 \) first jobs are the same for \( \pi \) and \( \pi' \). Moreover, the machines \( M_1 \) and \( M_2 \) work without idle time in both schedules \( \sigma \) and \( \sigma' \). Thus, \( C^1_{\pi(h_1)}(\sigma) = C^1_{\pi(h_1)}(\sigma') \) and \( C^2_{\pi(h_1)}(\sigma) = C^2_{\pi(h_1)}(\sigma') \). Moreover, \( \pi(j) = \pi'(j) \) for all \( j > h_1 \). Hence, the schedules \( \sigma \) and \( \sigma' \) are the same for the last \( n' - h_1 \) jobs and feasibility of \( \sigma' \) implies that operations of job \( \pi(j) \) do not overlap for \( j = h_1 + 1, \ldots, n' \).

(c) Firstly, we obtain an upper bound for the value of \( R_{j}(\pi) \) for all \( j, 0 \leq j \leq n' \), then we use this upper bound to show that in the schedule \( \sigma \) the buffer constraints are not violated.
Observe that \( R_0(\pi) = 0 \). Consider three sub-cases, which are defined by the set the job \( \pi(j) \) belongs to: either \( \pi(j) \in L_0' \) or \( \pi(j) \in L_1' \) or \( \pi(j) \in L_2' \).

**Case 1(c)** \( \pi(j) \in L_0' \). According to the algorithm either \( R_{j-1}(\pi) < \frac{3}{2}a_{\text{max}} \) or \( \frac{3}{2}a_{\text{max}} \leq R_{j-1}(\pi) < 2a_{\text{max}} \) and \( R_{j-1}(\pi) + \Delta_{\pi_2(i_2)} + \mu(i_0) < a_{\text{max}} \), where \( i_0 \) and \( i_2 \) are specified according to Algorithm 1. If \( R_{j-1}(\pi) < \frac{3}{2}a_{\text{max}} \), then
\[
R_j(\pi) = R_{j-1}(\pi) + b_j - a_j \leq \frac{3}{2}a_{\text{max}} + b_{\text{max}}. \tag{12}
\]
If \( \frac{3}{2}a_{\text{max}} \leq R_{j-1}(\pi) < 2a_{\text{max}} \), we get
\[
R_j(\pi) = R_{j-1}(\pi) + b_{\pi(j)} - a_{\pi(j)} \leq R_{j-1}(\pi) + \mu(i_0)
\leq a_{\text{max}} - \Delta_{\pi_2(i_2)} = a_{\text{max}} - b_{\pi_2(i_2)} + a_{\pi_2(i_2)} \leq 2a_{\text{max}}. \tag{13}
\]

**Case 2(c)** \( \pi(j) \in L_1' \). According to the algorithm \( R_{j-1}(\pi) < 2a_{\text{max}} \). So, we obtain
\[
R_j(\pi) = R_{j-1}(\pi) + b_j - a_j \leq 2a_{\text{max}} + b_{\text{max}} - \frac{1}{2}a_{\text{max}} = \frac{3}{2}a_{\text{max}} + b_{\text{max}}. \tag{14}
\]

**Case 3(c)** \( \pi(j) \in L_2' \). From (12)–(14) we have
\[
R_i(\pi) \leq \frac{3}{2}a_{\text{max}} + \max\{\frac{1}{2}a_{\text{max}}, b_{\text{max}}\}. \tag{15}
\]
for all \( i \in L_0' \cup L_1' \). Since \( R_{i-1}(\pi) \geq R_i(\pi) \) for any \( j \) such that \( \pi(j) \in L_2' \) the inequality (15) holds for all \( j \).

For any \( j \in N' \) consider the buffer consumption at its starting time \( S_j(\sigma) \). Let \( k(j) \) be the job with the smallest completion time such that \( C_k^{j}(\sigma) \geq S_j(\sigma) \). The buffer consumption at \( S_j(\sigma) \) does not exceed the buffer requirements of job \( k(j) \) and all jobs within interval \([k(j), \sigma, C_k^{j}(\sigma)]) \) and the job \( j \). By virtue of (3) and (15) this buffer load does not exceed
\[
a_{\text{max}} + R_{k(j)}(\pi) + a_{\text{max}} \leq 3.5a_{\text{max}} + \max\{0.5a_{\text{max}}, b_{\text{max}}\} = \Omega,
\]
hence the buffer capacity is observed every time a job starts its processing.

Since, both machines proceed jobs without idle time, the makespan of \( \sigma \) coincides with the load of the machine and thus the schedule \( \sigma \) is an optimal schedule.

\[\Box\]

It remains to note, that after removing from the schedule \( \sigma \) all jobs from set \( X \cup Y \) we obtain a feasible schedule for the original instance \( I \) of the problem. Further, it is easy to check that if permutations \( \pi_0, \pi_1, \) and \( \pi_2 \), are specified, the running time of Algorithm 1 linearly depends on \( n' \). In particular, it has been shown in [15] that if \( a_{\text{max}} = b_{\text{max}} \), then \( n' \leq 2n + 1 \), and such instance of the problem can be solved in \( O(n \log n) \) time. However, if the ratio between \( a_{\text{max}} \) and \( b_{\text{max}} \) is large enough, the number of jobs in \( X \cup Y \), and hence the running time of Algorithm 1 may increase significantly. The following implementation of the algorithm will allow to avoid the increase of the running time.

Firstly, we show that instead of inserting one job from \( X \) at each iteration of the first while-loop, we will schedule several jobs from \( X \) at the same time if they are to be scheduled sequentially. Consider an iteration \( i \) of the while-loop on which \( i_0 < |L_0'| \) and \( \pi(i-1) \in L_2' \). Denote by \( b = \frac{\text{Idle}_{j_0}}{b_{\text{max}}} \), and define
\[
\kappa_1 = \min\{k | R_{i-1}(\pi) + kb \geq 2a_{\text{max}} \}.
\]
\[ \kappa_2 = \min \{ k | R_{i-1}(\pi) + kb \geq 1.5a_{max} \& R_{i-1}(\pi) + \Delta \tau_{1}(i_2) + \mu(i_0) \geq a_{max} \}, \]

where \( i_0 \) is specified according to Algorithm 1. Observe that the calculation of \( \kappa_1 \) and \( \kappa_2 \) requires \( O(1) \) time. Let \( \kappa = \min \{ \kappa_1, \kappa_2, |L_0'| - i_0 \} \). Thus, instead of inserting \( \kappa \) jobs from the set \( L_0' \) we assign to the schedule one job with the processing time 0 on the first machine and the processing time \( kb \) on the second machine. The number of such new auxiliary jobs does not exceed \( |L_2| + 1 \) and therefore does not exceed \( n \). Observe that after deleting these auxiliary jobs, the resulting schedule will coincide with the schedule \( \sigma \) after all jobs from \( X \) were deleted, and therefore the resulting schedule is optimal.

The reduction of the time required for scheduling jobs in \( Y \) can be achieved similar to the aggregation of jobs in \( X \). If some job \( j \in Y \) is to be scheduled in the \( i \)th position in \( \pi \), then the number of jobs from \( Y \) that should be scheduled between \( j \) and the next job from \( N' \setminus Y \) (or the number of all unscheduled jobs in \( Y \) when all jobs in \( N' \setminus Y \) have been already scheduled) is computed, and the parameters \( l_{i,1}(\pi) \) and \( l_{i,2}(\pi) \) are updated accordingly without assigning the starting times for the corresponding jobs from \( Y \).

So, taking into account Lemma 1 we get the following result.

Theorem 2 Any instance of PP-problem that satisfies (3) can be solved in \( O(n \log n) \) time.

Corollary 1 Any instance of PP-problem that satisfies \( \Omega \geq 4.5 \max \{ a_{max}, b_{max} \} \) can be solved in \( O(n \log n) \) time.

4 Conclusion

This paper contributes to the efforts aimed at establishing the borderline between polynomially solvable and NP-hard cases of the two-machine flow shop problem with the makespan objective function and job-dependent storage requirements by analysing how the variation of a lower bound on the storage capacity, computed as a function of processing times, affects the computational complexity. The paper also strengthens the result in [15] by significantly reducing the lower bound that guarantees the polynomial-time solvability. One of the directions of the further research could be the analysis of the worst-case behaviour of the polynomial-time algorithm, presented in this paper, when this algorithm is applied to the general case of the makespan minimisation problem. Another direction of the future research could be the development of efficient optimisation algorithms for the general case of the makespan minimisation problem.

Acknowledgements The research of the first author was supported by the program of fundamental scientific researches of the SB RAS, project 0314-2019-0014.

References

1. Berlińska, J.: Scheduling for data gathering networks with data compression. Eur. J. Oper. Res. 246(3), 744–749 (2015)
2. Berlińska, J.: Heuristics for scheduling data gathering with limited base station memory. Ann. Oper. Res. 285(1), 149–159 (2020)
3. Berlińska, J., Kononov, A., Zinder, Y.: Two-machine flow shop with dynamic storage space. Optim. Lett. 15, 2433–2454 (2020)
4. Blazewicz, J., Lenstra, J.K., Kan, A.R.: Scheduling subject to resource constraints: classification and complexity. Discrete Appl. Math. 5(1), 11–24 (1983)
5. Brucker, P., Heitmann, S., Hurink, J.: Flow-shop problems with intermediate buffers. OR Spectr. 25(4), 549–574 (2003)
6. Brucker, P., Knust, S.: Complex Scheduling. Springer, Berlin (2012)
7. Emmons, H., Vairaktarakis, G.: Flow Shop Scheduling. Springer, Berlin (2013)
8. Ernst, A., Fung, J., Singh, G., Zinder, Y.: Flexible flow shop with dedicated buffers. Discrete Appl. Math. 261, 148–163 (2018)
9. Fung, J., Singh, G., Zinder, Y.: Capacity planning in supply chains of mineral resources. Inf. Sci. 316, 397–418 (2015)
10. Fung, J., Zinder, Y.: Permutation schedules for a two-machine flow shop with storage. Oper. Res. Lett. 44(2), 153–157 (2016)
11. Gu, H., Kononov, A., Memar, J., Zinder, Y.: Efficient Lagrangian heuristics for the two-stage flow shop with job dependent buffer requirements. J. Discrete Algorithms 52–53, 143–155 (2018)
12. Gu, H., Memar, J., Zinder, Y.: Improved Lagrangian relaxation-based optimisation procedure for scheduling with storage. IFAC-PapersOnLine 52(13), 100–105 (2019)
13. Johnson, S.M.: Optimal two-and three-stage production schedules with setup times included. Naval Res. Logist. Q. 1(1), 61–68 (1954)
14. Kononov, A., Hong, J.S., Kononova, P., Lin, F.C.: Quantity-based buffer-constrained two-machine flow-shop problem: active and passive prefetch models for multimedia applications. J. Sched. 15(4), 487–497 (2012)
15. Kononov, A., Memar, J., Zinder, Y.: Flow shop with job-dependent buffer requirements—a polynomial-time algorithm and efficient heuristics. In: International Conference on Mathematical Optimisation Theory and Operations Research, pp. 342–357. Springer (2019)
16. Kononova, P., Kochetov, Y.A.: The variable neighborhood search for the two machine flow shop problem with a passive prefetch. J. Appl. Ind. Math. 7(1), 54–67 (2013)
17. Le, H.T., Geser, P., Middendorf, M.: An iterated local search algorithm for the two-machine flow shop problem with buffers and constant processing times on one machine. In: European Conference on Evolutionary Computation in Combinatorial Optimisation (Part of EvoStar), pp. 50–65. Springer (2019)
18. Le, H.T., Geser, P., Middendorf, M.: Iterated local search and other algorithms for buffered two-machine permutation flow shops with constant processing times on one machine. Evolut. Comput. 29, 415–439 (2021)
19. Lin, F.C., Hong, J.S., Lin, B.M.: A two-machine flowshop problem with processing time-dependent buffer constraints—an application in multimedia presentations. Comput. Oper. Res. 36(4), 1158–1175 (2009)
20. Lin, F.C., Hong, J.S., Lin, B.M.: Sequence optimisation for media objects with due date constraints in multimedia presentations from digital libraries. Inf. Syst. 38(1), 82–96 (2013)
21. Lin, F.C., Lai, C.Y., Hong, J.S.: Minimize presentation lag by sequencing media objects for auto-assembled presentations from digital libraries. Data Knowl. Eng. 66(3), 382–401 (2008)
22. Min, Y., Choi, B.C., Park, M.J.: Two-machine flow shops with an optimal permutation schedule under a storage constraint. J. Sched. 23, 327–336 (2019)
23. Papadimitriou, C.H., Kanellakis, P.C.: Flowshop scheduling with limited temporary storage. J. ACM 27(3), 533–549 (1980)
24. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. Springer, Berlin (2016)
25. Witt, A., Voß, S.: Simple heuristics for scheduling with limited intermediate storage. Comput. Oper. Res. 34(8), 2293–2309 (2007)
26. Zinder, Y., Kononov, A., Fung, J.: A 5-parameter complexity classification of the two-stage flow shop scheduling problem with job dependent storage requirements. J. Comb. Optim. 42, 276–309 (2021)