Optimization Letters
https://doi.org/10.1007/s11590-020-01645-5

ORIGINAL PAPER

Two-machine flow shop with dynamic storage space

Joanna Berlińska[1] · Alexander Kononov[2,3] · Yakov Zinder[4]

Received: 31 March 2020 / Accepted: 8 September 2020

Abstract
The publications on two-machine flow shop scheduling problems with job dependent storage requirements, where a job seizes a portion of the storage space for the entire duration of its processing, were motivated by various applications ranging from supply chains of mineral resources to multimedia systems. In contrast to the previous publications that assumed that the availability of the storage space remains unchanged, this paper is concerned with a more general case when the availability is a function of time. It strengthens the previously published result concerning the existence of an optimal permutation schedule, shows that the variable storage space availability leads to the NP-hardness in the strong sense even for unit processing times, and presents a polynomial-time approximation scheme together with several heuristic algorithms. The heuristics are evaluated by means of computational experiments.

Keywords Two-machine flow shop · Makespan · Dynamic storage · Computational complexity · Polynomial-time approximation scheme

1 Introduction

This paper is concerned with the two-machine flow shop scheduling problem. This problem is widely used for modelling various real-world situations which can be viewed as a problem of scheduling a set of jobs on two machines, the first-stage

✉ Joanna Berlińska
Joanna.Berlinska@amu.edu.pl

Alexander Kononov
alvenko@math.nsc.ru

Yakov Zinder
Yakov.Zinder@uts.edu.au

1 Faculty of Mathematics and Computer Science, Adam Mickiewicz University, Poznan, Poland

2 Sobolev Institute of Mathematics, Novosibirsk, Russia

3 Novosibirsk State University, Novosibirsk, Russia

4 School of Mathematical and Physical Sciences, University of Technology Sydney, Sydney, Australia

machine and the second-stage machine [6]. According to the two-machine flow shop model, each job should be processed on the first-stage machine and, after the completion of this first operation, on the second-stage machine. The considered objective function is the total time needed for the completion of all jobs. In the literature on scheduling, the problems with this objective function are normally referred to as the makespan minimisation problems or simply makespan problems.

A number of applications, ranging from multimedia systems [14] and star data gathering networks [1] to supply chains of mineral resources [7], motivated the recent interest in the two-machine flow shop with limited storage space, also referred to as the two-machine flow shop with job dependent buffer requirements. This direction of research focuses on the situations with two distinct characteristics: (1) each job requires some storage space and the storage requirements vary from job to job; and (2) each job seizes the required portion of the storage space (buffer) from the start of its first operation till the completion of its second operation. Such use of the storage space differentiates the flow shop with limited storage from the two-machine flow shop with an additional resource, where the additional resource is used only during the processing on the machines [4,5].

It is known that the two-machine flow shop problem with limited storage and the objective of makespan is NP-hard in the strong sense [14]. Moreover, it remains NP-hard in the strong sense even under the restriction that the order in which the jobs should be processed on one of the machines is given [12]. According to [13] the makespan minimisation problem is also NP-hard in the following two cases: when all jobs have the same processing time on the second-stage machine and the buffer requirement of a job is proportional to its processing time on the first-stage machine, and in the case when all jobs have the same processing time on the second-stage machine and the same buffer requirements.

The case where all jobs have the same processing time on one of the machines, or the same processing time on the first-stage machine and the same processing time on the second-stage machine was considered in a number of publications, including [2,9] and the mentioned above [13]. Besides the theoretical interest, this case has applications in star data gathering networks where different workstations are allocated dataset-independent time slots for data transfer, and in unloading and loading involving a crane.

Furthermore, as is shown in [8], there are instances of the makespan minimisation problem where, in any optimal schedule, the order in which the jobs are processed on one of the machines differs from the order on the other machine. The existence of such instances significantly complicates the development of optimisation algorithms.

A schedule where the order in which the jobs are processed is the same for both machines, is called a permutation schedule. For two particular cases, the existence of an optimal schedule that is also a permutation one is proved in [15]. In both cases, the buffer requirement of a job is proportional to its processing time on the first-stage machine. One of these two cases is the case where the smallest processing time on the second-stage machine is greater than or equal to the largest processing time on the first-stage machine. It is shown that in this case an optimal schedule can be constructed in polynomial time. The publication [14] can be viewed as a source of motivation for studying this particular case as well as its mirror reflection where the smallest processing time on the first-stage machine is greater than or equal to the largest processing time on the second-stage machine.

There are many situations where the available storage space (buffer capacity) is a function of time. For example, the storage space is often shared by several clients of the same transportation facility, and the computer memory is often used by several processes simultaneously. Despite this, to the best of the authors' knowledge, there are no publications except [2] that study such systems. The paper addresses this gap in the literature on scheduling by considering the two-machine flow shop with variable storage capacity. The paper

(a) proves that the introduction of variable storage availability makes the problem NP-hard in the strong sense even for unit processing times (in contrast, the same problem with constant availability of the storage space is solvable in polynomial time [9]) (Sect. 3);
(b) strengthens the result in [15] by establishing the existence of an optimal permutation schedule even for arbitrary storage requirements which are not necessarily proportional to the duration of the first operation (Sect. 4);
(c) establishes the existence of an optimal permutation schedule for the case of arbitrary storage requirements when the smallest processing time on the first-stage machine is greater than or equal to the largest processing time on the second-stage machine (Sect. 4);
(d) shows that in the case of variable resource availability, even for unit processing times, the two-machine flow shop with an additional resource may not have an optimal permutation schedule (Sect. 4);
(e) presents a polynomial-time approximation scheme for the case where all jobs have the same processing time on the first-stage machine and the same processing time on the second-stage machine (Sect. 5);
(f) presents several heuristics for the case where all jobs have the same processing time on the first-stage machine and the same processing time on the second-stage machine, and compares them by means of computational experiments (Sects. 6, 7).

2 Problem formulation

The considered scheduling problem can be stated as follows. The jobs, comprising the set $N = \{1, \ldots, n\}$, are to be processed on two machines, machine M_1 and machine M_2. Each job should be processed first on M_1 (the first operation of the job) and then, from some point in time after the completion of the first operation, on M_2 (the second operation of the job). Each machine can process at most one job at a time (except the points in time when one job completes processing and another job starts processing on this machine), and each job can be processed on at most one machine at a time (except the situations when the completion time of the first operation coincides with the start of the second operation). If a machine starts processing a job, it continues its processing until the completion of the corresponding operation, i.e. no preemptions are allowed.

All processing times are positive integers. The processing time of job j on machine M_i will be denoted $p_{i,j}$ and the total processing time will be denoted by T, i.e.

$$T = \sum_{i \in \{1,2\}} \sum_{j \in N} p_{i,j}.$$

The processing of jobs commences at time $t = 0$. A schedule σ specifies for each $j \in N$ the point in time $S_j^1(\sigma)$ when job j starts its processing on M_1 and the point in time $C_j^2(\sigma)$ when job j completes its processing on M_2. Since preemptions are not allowed, for each job j, a schedule σ also specifies the completion time on machine M_1

$$C_j^1(\sigma) = S_j^1(\sigma) + p_{1,j}$$

and the starting time on machine M_2

$$S_j^2(\sigma) = C_j^2(\sigma) - p_{2,j}.$$

The notation S_j^1, C_j^1, S_j^2, C_j^2 will be used instead of $S_j^1(\sigma)$, $C_j^1(\sigma)$, $S_j^2(\sigma)$, $C_j^2(\sigma)$ if it is clear what schedule is considered.

In order to be processed, each job j requires ω_j units of an additional resource, referred to as storage space or a buffer, which it seizes during the time interval $[S_j^1, C_j^2)$ and releases at the point in time C_j^2. All ω_j are nonnegative integers. For any point in time $0 \le t < T$ the permissible consumption of the storage space is determined by a piecewise constant function $\Omega(t)$. In other words, at any point in time $0 \le t < T$, any schedule must satisfy the condition

$$\sum_{\{j:\, S_j^1 \le t < C_j^2\}} \omega_j \le \Omega(t).$$

The function $\Omega(t)$ satisfies the condition

$$\max_{j \in N} \omega_j \le \min_{0 \le t < T} \Omega(t), \tag{1}$$

changes its value only at integer points and is given as the sequence $\Omega(0), \ldots, \Omega(T-1)$ where, for any integer $0 \le \tau \le T-1$, the value of $\Omega(t)$ in the time interval $[\tau, \tau+1)$ is $\Omega(\tau)$.

The goal is to find a schedule that minimises the makespan

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

In what follows, the problem stated above will be referred to as the two-machine flow shop with job dependent storage requirements and dynamic storage availability. In the standard three-field notation [11] this problem can be denoted $F2|storage, \omega_j, \Omega(t)|C_{max}$.

Since all processing times are integer and all points of discontinuity of $\Omega(t)$ are also integer, for any instance of $F2|storage, \omega_j, \Omega(t)|C_{max}$, there exists an optimal schedule σ such that all starting times $S_j^1(\sigma)$ and $S_j^2(\sigma)$ are integer. Therefore, in what follows, without loss of generality, it is assumed that the starting times of all operations on both machines should be integer.

3 Computational complexity

Denote by $F2|storage, \omega_j, \Omega(t), p_{i,j} = 1|C_{max}$ the restricted version of the $F2|storage, \omega_j, \Omega(t)|C_{max}$ problem where all processing times are equal to one unit of time. In contrast to the case of constant storage availability, which is polynomially solvable when all processing times are equal to one unit of time [9], the $F2|storage, \omega_j, \Omega(t), p_{i,j} = 1|C_{max}$ problem is NP-hard in the strong sense. This will be proved below by a reduction from the Numerical Matching with Target Sums (NMTS) decision problem, which is NP-complete in the strong sense [10]. The NMTS decision problem is stated as follows:
INPUT: three sets $\{x_1, \ldots, x_r\}$, $\{y_1, \ldots, y_r\}$ and $\{z_1, \ldots, z_r\}$ of positive integers, where

$$\sum_{i=1}^{r} x_i + \sum_{i=1}^{r} y_i = \sum_{i=1}^{r} z_i. \tag{2}$$

QUESTION: do there exist permutations $(i_1, \ldots, i_r)$ and $(j_1, \ldots, j_r)$ of the indices $1, \ldots, r$ such that $z_k = x_{i_k} + y_{j_k}$ for all $k \in \{1, \ldots, r\}$?

Theorem 1 *The $F2|storage, \omega_j, \Omega(t), p_{i,j} = 1|C_{max}$ problem is NP-hard in the strong sense.*

Proof Let $\{x_1, \ldots, x_r\}$, $\{y_1, \ldots, y_r\}$ and $\{z_1, \ldots, z_r\}$ be an arbitrary instance of the NMTS problem, and let

$$x = \max_{1 \le i \le r} x_i \quad \text{and} \quad Z = x + \max_{1 \le i \le r} y_i + \max_{1 \le i \le r} z_i.$$

Consider the following instance of the decision version of the makespan minimisation problem $F2|storage, \omega_j, \Omega(t), p_{i,j} = 1|C_{max}$:

INPUT: $N = \{1, \ldots, 2r\}$,
$p_{1,i} = p_{2,i} = 1$ for $1 \le i \le 2r$,
$\omega_i = 2Z + x_i$ for $1 \le i \le r$,
$\omega_i = Z + y_{i-r} + x$ for $r + 1 \le i \le 2r$,

$$\Omega(t) = \begin{cases} 2Z + x & \text{for } t = 0 \\ 3Z + x + z_k & \text{for } t = 3k - 2 \text{ and } k \in \{1, \ldots, r\} \\ 2Z + x & \text{for } t = 3k - 1 \text{ and } k \in \{1, \ldots, r - 1\} \\ 2Z + x & \text{for } t = 3k \text{ and } k \in \{1, \ldots, r - 1\} \\ 2Z + x & \text{for } t = k \text{ and } k \in \{3r - 1, \ldots, 4r - 1\}. \end{cases}$$

QUESTION: does there exist a schedule σ such that $C_{max}(\sigma) \le 3r$?

Observe that this instance satisfies the condition (1) and suppose that there exists a schedule σ with the makespan that does not exceed $3r$, i.e. assume that the answer to the scheduling problem is YES.

The function $\Omega(t)$ induces the partition of the time interval $[0, 3r]$ into $2r$ unit intervals with the buffer capacity $2Z + x$, referred to as 1-intervals because at most one job can use the storage space at any point in such an interval, and r unit intervals with the buffer capacity $3Z + x + z_k$ where $k \in \{1, \ldots, r\}$, referred to as 2-intervals. Taking into account that the total processing time is $T = 4r$, both machines are busy in σ during each 2-interval and one machine is busy in σ during each 1-interval.

The same job cannot be processed in any two 2-intervals because in this case it consumes the storage space during each 1-interval that separates these 2-intervals (there are at least two such 1-intervals), leaving not enough storage space for the jobs that must be processed in the 1-intervals. Since the number of jobs is $2r$ and since the number of jobs which have an operation, processed in the 2-intervals, is $2r$, one operation of each job is processed in a 1-interval whereas its another operation is processed in a 2-interval. Taking into account that no two jobs from the set $\{1, \ldots, r\}$ can be processed concurrently due to the insufficient storage space, in every 2-interval $[3k-2, 3k-1)$, where $k \in \{1, \ldots, r\}$, an operation of some job from the set $\{1, \ldots, r\}$, denoted i_k, is processed concurrently with an operation of some job from the set $\{r + 1, \ldots, 2r\}$, denoted g_k. For each such pair of jobs,

$$\omega_{i_k} + \omega_{g_k} \leq \Omega(3k - 2).$$

Let $j_k = g_k - r$. Then, by virtue of $\omega_{i_k} = 2Z + x_{i_k}$ and $\omega_{g_k} = Z + y_{g_k - r} + x$,

$$x_{i_k} + y_{j_k} \leq z_k,$$

which by (2) implies $z_k = x_{i_k} + y_{j_k}$ for all $k \in \{1, \ldots, r\}$.

Now suppose that there exist permutations $(i_1, \ldots, i_r)$ and $(j_1, \ldots, j_r)$ of the indices $1, \ldots, r$ such that $z_k = x_{i_k} + y_{j_k}$ for all $k \in \{1, \ldots, r\}$, i.e. assume that the answer to the considered instance of NMTS is YES. Then, the schedule where, for each job g, $S_g^1 + 1 = S_g^2$ and where, for each $1 \leq k \leq r$, $S_{i_k}^1 = 3(k - 1)$ and $S_{j_k+r}^1 = 3k - 2$, has the required makespan of $3r$. □

4 Permutation schedules

A schedule for the $F2|storage, \omega_j, \Omega(t)|C_{max}$ problem is a no-wait schedule if, for every $j \in N$, $S_j^2 = C_j^1$.

Lemma 1 *Let $j_1, \ldots, j_n$ be the sequence in which the jobs are processed on M_1 in some schedule σ. If, for all $1 \leq k < n$, the processing times satisfy the condition $p_{2,j_k} \leq p_{1,j_{k+1}}$, then there exists a no-wait schedule σ' such that $C_{max}(\sigma) \geq C_{max}(\sigma')$.*

Proof Consider the no-wait schedule σ' where $S_j^1(\sigma') = S_j^1(\sigma)$ and $S_j^2(\sigma') = C_j^1(\sigma)$ for all $j \in N$. Since, for any job j, $S_j^2(\sigma) \geq C_j^1(\sigma)$ and therefore $S_j^2(\sigma') \leq S_j^2(\sigma)$, the schedule σ' satisfies the inequality $C_{max}(\sigma) \geq C_{max}(\sigma')$. It also satisfies the storage restrictions because, for each job j, the choice of $S_j^1(\sigma')$ and $S_j^2(\sigma')$ implies

$$[S_j^1(\sigma'), C_j^2(\sigma')) \subseteq [S_j^1(\sigma), C_j^2(\sigma)).$$

Furthermore, for each job j and each $1 \leq k < n$,

$$\begin{aligned} C_{j_k}^2(\sigma') &= C_{j_k}^1(\sigma') + p_{2,j_k} = C_{j_k}^1(\sigma) + p_{2,j_k} \leq S_{j_{k+1}}^1(\sigma) + p_{2,j_k} \\ &= S_{j_{k+1}}^1(\sigma') + p_{2,j_k} \leq S_{j_{k+1}}^1(\sigma') + p_{1,j_{k+1}} = S_{j_{k+1}}^2(\sigma') \end{aligned}$$

which shows that the schedule σ' satisfies the restrictions imposed by the processing times. □

The proof of the lemma below is similar to the proof of Lemma 1. The main difference is the choice of the schedule σ'. Now this schedule is defined as follows: for each $j \in N$, $S_j^2(\sigma') = S_j^2(\sigma)$ and $C_j^1(\sigma') = S_j^2(\sigma)$.

Lemma 2 *Let $j_1, \ldots, j_n$ be the sequence in which the jobs are processed on M_2 in some schedule σ. If, for all $1 \leq k < n$, the processing times satisfy the condition $p_{2,j_k} \geq p_{1,j_{k+1}}$, then there exists a no-wait schedule σ' such that $C_{max}(\sigma) \geq C_{max}(\sigma')$.*

If a schedule is a no-wait schedule it obviously is a permutation schedule. This observation leads to the following theorem.

Theorem 2 *For any instance of the $F2|storage, \omega_j, \Omega(t)|C_{max}$ problem such that either $\min_{j\in N} p_{1,j} \geq \max_{j\in N} p_{2,j}$ or $\min_{j\in N} p_{2,j} \geq \max_{j\in N} p_{1,j}$, there exists an optimal schedule which is a permutation one.*

Consider the restricted version of the $F2|storage, \omega_j, \Omega(t)|C_{max}$ problem where all $p_{1,j}$ are equal and all $p_{2,j}$ are also equal. Let p_1 be the common value of all $p_{1,j}$ and let p_2 be the common value of all $p_{2,j}$. Of course p_1 and p_2 vary from instance to instance. This restricted makespan minimisation problem will be denoted by $F2|storage, \omega_j, \Omega(t), p_i|C_{max}$. The instance of the $F2|storage, \omega_j, \Omega(t), p_i|C_{max}$ problem with the set of jobs N', function $\Omega'(t)$, storage requirements ω'_j, and processing times p'_1 and p'_2 is conjugate to the instance of the $F2|storage, \omega_j, \Omega(t), p_i|C_{max}$ problem with the set of jobs N'', function $\Omega''(t)$, storage requirements ω''_j, and processing times p''_1 and p''_2 if

$$N' = N'', \quad \Omega'(t) = \Omega''(t), \quad \omega'_j = \omega''_j \text{ for all jobs } j, \quad p'_1 = p''_2, \quad p'_2 = p''_1.$$

Lemma 3 *Any two conjugate instances of the $F2|storage, \omega_j, \Omega(t), p_i|C_{max}$ problem have the same optimal makespan.*

Proof Consider a pair of conjugate instances I' and I'' with the set of jobs $N = \{1, \ldots, n\}$, function $\Omega(t)$, and storage requirements ω_j. Let the first of these instances have processing times p_1 and p_2 and let σ' be an optimal schedule for this instance.

In the light of Lemmas 1 and 2, without loss of generality, σ' is a no-wait schedule. Then, σ' defines n time intervals $[S_j^1(\sigma'), C_j^2(\sigma')]$, one for each job j in N.

For the conjugate instance I'', consider the schedule σ'' where, for each $1 \le j \le n$, $S_j^1(\sigma'') = S_j^1(\sigma')$ and $C_j^2(\sigma'') = C_j^2(\sigma')$. Since, for each $1 \le j \le n$,

$$C_j^2(\sigma') - S_j^1(\sigma') = p_1 + p_2,$$

$S_j^2(\sigma'') = C_j^1(\sigma'')$ for each job j, i.e. the operations of the same job do not overlap in σ''. Furthermore, because σ' is a no-wait schedule, for each job j there exists at most one job g such that the second operation of j is processed concurrently with the first operation of g. For any such pair of jobs j and g, these two jobs are processed concurrently in both schedules, σ' and σ'', in the same time interval, which is the intersection of the time intervals $[S_j^1(\sigma'), C_j^2(\sigma')]$ and $[S_g^1(\sigma'), C_g^2(\sigma')]$. Thus, σ'' satisfies the buffer restrictions. Finally, since σ' is a feasible no-wait schedule,

$$C_j^2(\sigma') - S_g^1(\sigma') \le p_1, \quad C_j^2(\sigma') = S_j^1(\sigma') + p_1 + p_2 \text{ and } S_j^1(\sigma') + p_1 \le S_g^1(\sigma').$$

Taking this into account,

$$S_g^1(\sigma'') = S_g^1(\sigma') = S_g^1(\sigma') + S_j^1(\sigma') + p_1 + p_2 - C_j^2(\sigma') \ge S_j^1(\sigma') + p_2 = C_j^1(\sigma'')$$

and

$$S_g^2(\sigma'') = S_g^1(\sigma') + p_2 = S_g^1(\sigma') + p_2 + C_j^2(\sigma') - S_j^1(\sigma') - p_1 - p_2 \ge C_j^2(\sigma'').$$

Hence, the operations of jobs j and g do not overlap on any of the two machines, and in consequence, σ'' is a feasible schedule. Denote by $C^*_{max}(I)$ the optimal makespan for instance I. Since all job completion times are the same in σ' and σ'', it holds that $C_{max}(\sigma') = C_{max}(\sigma'')$, and hence, $C^*_{max}(I'') \le C^*_{max}(I')$ Because the conjugation relation is symmetric, it is also true that $C^*_{max}(I') \le C^*_{max}(I'')$, and hence, $C^*_{max}(I') = C^*_{max}(I'')$. □

Problem $F2|storage, \omega_j, \Omega(t)|C_{max}$ is connected to the two-machine flow shop problem with an additional resource, where the resource is used only during the processing on the machines [3,4,16,17], because the storage space can also be viewed as an additional resource. However, the condition that the storage is used in the whole interval from the start of the first operation of a job till the completion of its second operation, differentiates the two problems, even in the case of unit processing times. By Theorem 2, for any instance of problem $F2|storage, \omega_j, \Omega(t), p_{i,j} = 1|C_{max}$, there exists a permutation schedule. Although in the case when the available amount of the resource is constant, any instance of the two-machine flow shop problem with an additional resource and unit operation execution times also has an optimal permutation schedule [16], it will be shown below that this is not true when the resource availability is a function of time.

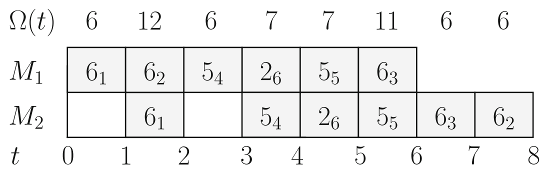
$\Omega(t)$ 6 12 6 7 7 11 6 6
M_1 6_1 6_2 5_4 2_6 5_5 6_3
M_2 6_1 5_4 2_6 5_5 6_3 6_2
t 0 1 2 3 4 5 6 7 8

Fig. 1 Optimal schedule for the proof of Theorem 3. Jobs are described by storage requirement ω_j, and job number j given in subscript

Theorem 3 *There exist instances of the two-machine flow shop scheduling problem with unit processing times, an additional resource and variable resource availability, which do not have optimal permutation schedules.*

Proof Let $n = 6$, and let the resource requirements of the jobs be $(\omega_j)_{j=1}^{6} = (6, 6, 6, 5, 5, 2)$. Let the resource availability be described by a function $\Omega(t)$ given by the sequence $(6, 12, 6, 7, 7, 11, 6, 6, 6, 6, 6, 6)$. An optimal schedule of length 8 is presented in Fig. 1. It will be shown that there exists no permutation schedule of length at most 8. Note that the total resource availability in the interval $[0, 8)$ is $\sum_{t=0}^{7} \Omega(t) = 61$, while the total requirement of all operations is $2\sum_{j=1}^{6} \omega_j = 60$. Thus, in order to construct a schedule of length 8, at most one unit of the resource can be wasted during at most one time unit.

Suppose an optimal permutation schedule σ exists. Without loss of generality, assume that if $\omega_i = \omega_j$ and $i < j$, then job i is scheduled before job j. Let j_1 be the first job in σ. If $\omega_{j_1} < 6$, then at least one unit of the resource is wasted in the interval $[0, 1)$, and at least one unit is lost in the interval $[1, 2)$. Thus, $\omega_{j_1} = 6$, $j_1 = 1$, and this job completes on M_2 at time $C_1^2 = 2$, as otherwise at least 6 units of the resource would be wasted in the interval $[1, 2)$. Similarly, let j_2 be the second job in σ. If $\omega_{j_2} < 6$, then at least one unit of the resource is wasted in the interval $[1, 2)$, and another one in the interval $[2, 3)$ or $[3, 4)$. Hence, $\omega_{j_2} = 6$, $j_2 = 2$ and $S_2^1 = 1$. Consider the following two cases: either (1) $C_2^2 = 3$, or (2) $C_2^2 > 3$.

In case (1), machine M_1 is idle in the time interval $[2, 3)$, as there is not enough available resource for starting another job. Let j_3 be the third job in σ. Note that j_3 has to start at time 3 on M_1, as 7 units of the resource would be wasted otherwise. If $\omega_{j_3} < 6$, then at least 2 units of the resource are wasted in the interval $[3, 4)$. Moreover, if $\omega_{j_3} = 6$, then one unit of the resource is lost in the interval $[3, 4)$, and another one in the interval $[4, 5)$.

In case (2), machine M_2 is idle in the interval $[2, 3)$, and job 2 has to be completed at time $C_2^2 = 4$, because at least 2 units of the resource would be wasted in the interval $[2, 4)$ otherwise. Hence, one unit of the resource is wasted in the interval $[3, 4)$, and in consequence, job 3 (whose resource requirement is 6) has to be executed on M_1 in the interval $[2, 3)$. Then, job 3 is scheduled on M_2 in the interval $[4, 5)$, which leads to losing one more unit of the resource.

Thus, in both cases at least two units of the resource are wasted, and hence, an optimal permutation schedule does not exist. $\square$

5 Polynomial-time approximation scheme

This section presents a polynomial-time approximation scheme for problem $F2|storage, \omega_j, \Omega(t), p_i|C_{max}$. By Lemmas 1 and 2, it is enough to consider no-wait schedules. For clarity of presentation, it is assumed that $p_1 \geq p_2$. However, it follows from Lemma 3 that the proposed approximation scheme can also be used for solving the problem when $p_1 < p_2$.

For any $\varepsilon > 0$, let $k = \left\lfloor \frac{n\varepsilon}{2} \right\rfloor$ and $q = \left\lceil \frac{2}{\varepsilon} \right\rceil$. It will be shown that a $(1+\varepsilon)$-approximation of the optimal solution can be found in $O(p_2 q^2 n^q)$ time. Recall that an instance of problem $F2|storage, \omega_j, \Omega(t), p_i|C_{max}$ contains the values n, p_1 and p_2, the buffer requirements ω_i of the n jobs, and a sequence $\Omega(0), \ldots, \Omega(T-1)$, where $T = n(p_1 + p_2)$, representing the storage availability function. Thus, the instance size is $O(n(p_1 + p_2))$, and hence, the running time of the proposed algorithm is indeed polynomial.

Theorem 4 *For any instance of $F2|storage, \omega_j, \Omega(t), p_i|C_{max}$ and any given small $\varepsilon > 0$, a schedule σ such that*

$$C_{max}(\sigma) \leq (1+\varepsilon) C_{max}(\sigma^*), \tag{3}$$

where σ^* is an optimal schedule, can be constructed in $O(p_2 q^2 n^q)$ time.

Proof Assume that there are sufficiently many jobs and number them in the non-decreasing order of their storage requirements, i.e. $\omega_1 \leq \cdots \leq \omega_n$. For each job j, replace its storage requirement ω_j by a new one (denoted α_j) as follows. For each $1 \leq e \leq q-1$ and each $(e-1)k < j \leq ke$, let $\alpha_j = \omega_{ke}$, and for each $k(q-1) < j \leq n$, let $\alpha_j = \omega_n$. Observe that any schedule for the problem with the new storage requirements is feasible for the problem with the original storage requirements.

An optimal schedule for the new storage requirements can be constructed by dynamic programming as follows. For $1 \leq e \leq q$, let

$$\pi(e) = \begin{cases} ke & \text{if } 1 \leq e < q \\ n & \text{if } e = q \end{cases},$$

and consider $(q+1)$-tuples $(n_1, \ldots, n_q, i)$ such that (a) $0 \leq n_e \leq k$ for all $1 \leq e \leq q-1$, and $0 \leq n_q \leq n - k(q-1)$; and (b) $1 \leq i \leq q$ and $n_i > 0$. Each such $(q+1)$-tuple represents $n_1 + \cdots + n_q$ jobs such that, for each $1 \leq e \leq q$, this set contains n_e jobs j, whose α_j is $\omega_{\pi(e)}$.

For each $(q+1)$-tuple $(n_1, \ldots, n_q, i)$, let $F(n_1, \ldots, n_q, i)$ be the minimal time needed for completion of all jobs corresponding to $(n_1, \ldots, n_q, i)$, under the condition that the job with the largest completion time among these jobs is a job with the new storage requirement $\omega_{\pi(i)}$. Consequently, the optimal makespan is

$$C = \min_{1 \leq i \leq q} F(k, \ldots, k, n-(q-1)k, i).$$

The $(q+1)$-tuples, satisfying the condition $n_1 + \cdots + n_q = 1$, will be referred to as boundary $(q+1)$-tuples. Then, $F(n_1, \ldots, n_q, i) = p_1 + p_2$ for each boundary $(n_1, \ldots, n_q, i)$.

For any positive integer t, any $1 \le i \le q$ and any $1 \le e \le q$, let $\omega_{i,e} = \omega_{\pi(i)} + \omega_{\pi(e)}$, and

$$W_{i,e}(t) = \begin{cases} p_1 + \min\{\delta \ge 0\colon \omega_{i,e} \le \min\limits_{\tau \in [t-p_2+\delta, t)} \{\Omega(\tau)\}\} & \text{if } \omega_{i,e} \le \Omega(t-1) \\ p_1 + p_2 & \text{if } \omega_{i,e} > \Omega(t-1) \end{cases}.$$

Note that for given i, e and t, the value of $W_{i,e}(t)$ can be calculated in $O(p_2)$ time. The values of F for all $(q+1)$-tuples that are not boundary are computed using the following recursive equation:

$$F(n_1, \ldots, n_i + 1, \ldots, n_q, i) = \min_{\{e\colon n_e > 0\}} [F(n_1, \ldots, n_q, e) + W_{i,e}(F(n_1, \ldots, n_q, e))].$$

The dynamic programming algorithm above constructs an optimal schedule σ in $O(p_2 q^2 n^q)$ time, and it only remains to show that (3) holds.

Let σ^* be an optimal schedule for the problem with the original storage requirements. This schedule can be converted into a schedule η for the problem with the new storage requirements as follows. For each job j such that $1 \le j \le n-k$, let $C_j^2(\eta) = C_{j+k}^2(\sigma^*)$ and, for each job j such that $n - k < j$, let

$$C_j^2(\eta) = C_{max}(\sigma^*) + (p_1 + p_2)(j - n + k).$$

Since $p_1 \ge p_2$ and $p_1 n < C_{max}(\sigma^*)$, it holds that

$$\begin{aligned} C_{max}(\sigma) &\le C_{max}(\eta) \le C_{max}(\sigma^*) + (p_1 + p_2)k \le C_{max}(\sigma^*) + 2p_1 k \\ &\le C_{max}(\sigma^*) + p_1 n \varepsilon \le (1+\varepsilon) C_{max}(\sigma^*), \end{aligned}$$

which completes the proof. □

6 Heuristics

In this section, heuristic algorithms are proposed for problem $F2|storage, \omega_j, \Omega(t), p_i|C_{max}$. Similarly as in the previous section, it is assumed for clarity that $p_1 \ge p_2$.

The main difficulty in designing heuristics for the considered problem consists in the frequent changes of the available storage size. Indeed, suppose a partial schedule σ for time interval $[t, t+\delta)$ is built, and in order to schedule the remaining jobs or to improve some other schedule part, σ has to be moved to start at time $t' \ne t$. The buffer availability pattern in interval $[t', t'+\delta)$ may be completely different from that in the original interval $[t, t+\delta)$, and hence, keeping the job order from σ may require additional idle times due to insufficient storage space. In such a case, it is probable that the schedule modification will be counterproductive. Therefore, simple heuristics building the schedules from left to right are proposed first, and the remaining algorithms consist in improving the initial schedules by small modifications.

Algorithm **LF** implements the Largest Fit rule. Every time t machine M_1 becomes idle, the largest available job which can be executed without violating the buffer limit is started on it. If no such job can be found, machine M_1 remains idle for one unit of time, and the search for a feasible job is repeated at time $t+1$.

Using algorithm LF may be disadvantageous if it is often the case that the buffer is large enough to hold two small jobs, but no job fits together with the largest available job. This motivates designing heuristic **LFAhead**, which also uses the largest fit rule, but additionally looks one job ahead in an attempt to avoid idle time on the first machine. Precisely, the job to be scheduled on M_1 at time t is the largest feasible job such that another available job can be started immediately after it, at time $t + p_1$. If no such job can be found, then the largest job that fits in the storage space is chosen. If no job can be started at time t, the algorithm moves to time $t+1$.

Algorithm **Rnd** constructs a random job sequence. The jobs are started without unnecessary delay, as soon as the previous job completes on the first machine and a sufficient amount of storage space is available. This algorithm is used mainly to verify if the remaining heuristics perform well in comparison to what can be achieved without effort.

The next group of heuristics are local search algorithms **LFLocal**, **LFAheadLocal** and **RndLocal**. Each of them starts with an initial schedule delivered by the corresponding heuristic described above (LF, LFAhead or Rnd). Then, for each pair of jobs it is checked whether swapping their positions leads to improving the schedule. Let $S = [S_{[1]}, S_{[2]}, \ldots, S_{[n]}]$ and $S' = [S'_{[1]}, S'_{[2]}, \ldots, S'_{[n]}]$ be the increasing sequences of job starting times on machine M_1 in the current schedule σ and in a new schedule σ', respectively. Schedule σ' is considered better than σ if its makespan is smaller than that of σ, or if the two makespans are equal and sequence S' is lexicographically smaller than S. The swap that results in the best schedule (if any) is executed, and the search is continued until no further improvement is possible.

Furthermore, variable neighborhood search (VNS) algorithms are proposed. Variable neighborhood search is a metaheuristic consisting in systematically changing the neighborhoods used during local search. In the proposed VNS, the following three neighborhoods are used. For a given schedule σ, neighborhood $N_1(\sigma)$ contains all schedules obtained from σ by swapping a single pair of jobs. Neighborhood $N_2(\sigma)$ consists of all schedules obtained from σ by moving a job from position i to some other position j, for any $i, j \in \{1, \ldots, n\}$, $i \neq j$. Neighborhood $N_3(\sigma)$ contains all schedules obtained from σ by reversing a sequence of jobs $\sigma(i), \ldots, \sigma(j)$, for any pair of positions $i, j \in \{1, \ldots, n\}$, $i < j$. For a given initial solution, the variable neighborhood search starts with setting the current neighborhood number to $k = 1$. In each step of the algorithm, if local search with respect to neighborhood N_k leads to a schedule improvement, the current schedule is updated and k is changed to 1. In the opposite case, k is increased by 1. The search continues until reaching $k = 4$, which means that the current solution could not be improved. Corresponding to the choice of the initial schedule, the proposed variable neighborhood search algorithms are denoted by **LFVNS**, **LFAheadVNS** and **RndVNS**.

Finally, the proposed variable neighborhood search is embedded in the iterated search framework, using two tunable parameters $f \in (0, 0.5)$ and $r > 0$. Given an initial schedule, the variable neighborhood search procedure described above is applied. In the obtained schedule, $\lceil fn \rceil$ randomly chosen pairs of jobs are swapped, and then the variable neighborhood search is run again. This procedure is repeated r times. The best solution found is recorded and returned as the final result. Reflecting the way the initial schedule is constructed, the iterated variable neighborhood search (IVNS) algorithms are called **LFIVNS**, **LFAheadIVNS** and **RndIVNS**.

In order to assess the quality of the results obtained by the heuristics, an integer linear program delivering optimal solutions is proposed. Recall that $T = n(p_1 + p_2)$ is an upper bound on the minimum schedule length C_{max}. For each $j = 1, \dots, n$ and $t = 0, \dots, T - 1$, define binary variables $x_{j,t}$ such that $x_{j,t} = 1$ if job j starts on machine M_1 at time t, and $x_{j,t} = 0$ in the opposite case. The minimum schedule length is computed as follows.

$$\text{minimise} \quad C_{max} \tag{4}$$

$$\sum_{j=1}^{n} \sum_{\tau=\max\{0,t+1-p_1-p_2\}}^{t} \omega_j x_{j,\tau} \le \Omega(t) \quad \text{for } t = 0, \dots, T-1 \tag{5}$$

$$\sum_{j=1}^{n} \sum_{\tau=t}^{t+p_1-1} x_{j,\tau} \le 1 \quad \text{for } t = 0, \dots, T - p_1 \tag{6}$$

$$\sum_{t=0}^{T-1} x_{j,t} = 1 \quad \text{for } j = 1, \dots, n \tag{7}$$

$$\sum_{t=0}^{T-1} t x_{j,t} + p_1 + p_2 \le C_{max} \quad \text{for } j = 1, \dots, n \tag{8}$$

$$x_{j,t} \in \{0, 1\} \quad \text{for } j = 1, \dots, n, \quad t = 0, \dots, T-1 \tag{9}$$

In the above program, constraints (5) guarantee that the jobs executed in the interval $[t, t+1)$ fit in the available buffer. Inequalities (6) ensure that at most one job starts on machine M_1 in each interval $[t, t + p_1)$. Since $p_1 \ge p_2$, this means that no two jobs are executed on the same machine at the same time. Each job starts exactly once by (7). Inequalities (8) guarantee that all jobs are completed by time C_{max}.

7 Computational experiments

The quality of the delivered solutions and the running times of the proposed heuristics were tested in a series of computational experiments. The algorithms were implemented in C++ and run on an Intel Core i7-7820HK CPU @ 2.90 GHz with 32 GB RAM. Integer linear programs were solved using Gurobi.

The number of jobs in the generated instances was $n \in \{30, 100\}$. In the tests where the two operations of each job had the same duration, their execution times were $p_1 =$

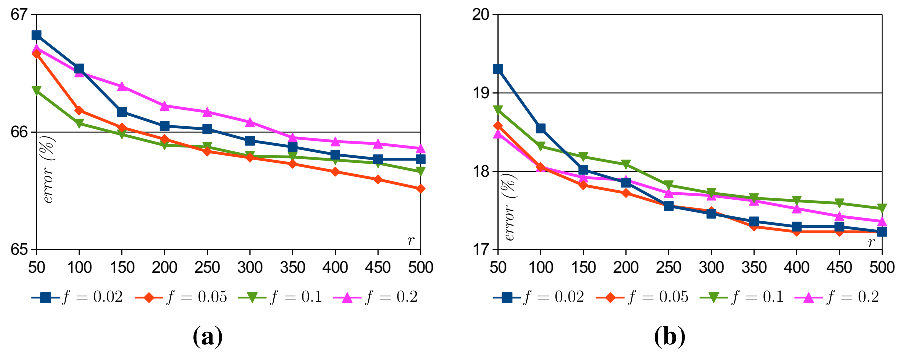

Fig. 2 RndIVNS tuning, $n = 100$, $p_1 = p_2 = 5$. **a** Rnd instances, **b** Dec instances

$p_2 \in \{1, 3, 5\}$. In the instances with different processing times of the two operations, values $(p_1, p_2) \in \{(2, 1), (4, 2), (5, 2)\}$ were used. The buffer requirements ω_j were selected uniformly at random from the range $[10n, 20n]$. Available buffer sizes $\Omega(t)$ were chosen randomly from the range $[20n, 32n]$, for each $t = 0, 1, \dots, T - 1$ independently. This choice of ranges was based on the experimental analysis presented in [2]. In addition to the tests with random buffer availability, instances with available storage space increasing or decreasing with time were constructed by sorting the values $\Omega(t)$. These three groups of tests will be referred to as Rnd, Inc and Dec instances, correspondingly. For each analysed setting, 30 tests were generated and solved.

Many test instances could not be solved by Gurobi to optimality in reasonable time. Therefore, a 1 h time limit was imposed. Since the optimal solutions were not always known, the quality of schedules was measured by the relative percentage error from the lower bound $LB = \max\{np_1 + p_2, LB_G\}$, where LB_G was the lower bound obtained by Gurobi during its computations.

The proposed iterated variable neighborhood search algorithms have two parameters, f and r, which have to be tuned. In order to avoid bias due to starting at a specific schedule, algorithm RndIVNS was used for choosing the values of these parameters. Figure 2 presents the results delivered by this algorithm with $f \in \{0.02, 0.05, 0.01, 0.2\}$ and $r \le 500$ for Rnd and Dec instances with $n = 100$ and $p_1 = p_2 = 5$. Similar results were obtained for other analysed combinations of instance type and values p_1 and p_2. Naturally, for any fixed f, the quality of the obtained schedules improves with increasing r. Thus, $r = 500$ was selected, as it seems a good compromise between quality and time. The value f that results in the shortest schedules depends on the number of iterations r. For $r = 500$, the best results were obtained for $f = 0.05$ in most settings. In the cases when some other value of f was better, the difference between the obtained errors was insignificant (see Fig. 2b). Therefore, $f = 0.05$ was chosen.

Table 1 presents the results delivered by the respective algorithms for Rnd instances with $n = 30$ and $p_1 = p_2$. The algorithms are divided into groups according to their complexity, and in consequence, running time: from the fastest simple heuristics to the slowest integer linear programming (ILP). All integer programs were solved within the time limit for $p_i \in \{1, 3\}$, but for $p_i = 5$, optimal solutions could not be obtained

Table 1 Average percentage distance from *LB* and running time for $n = 30$, $p_1 = p_2$, Rnd instances

Algorithm	$p_1 = p_2 = 1$		$p_1 = p_2 = 3$		$p_1 = p_2 = 5$	
	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
LF	10.53	1.82E−5	8.94	2.35E−5	8.43	3.22E−5
LFAhead	9.55	2.68E−5	8.43	3.83E−5	7.36	5.16E−5
Rnd	30.31	1.75E−5	21.79	2.02E−5	17.07	1.98E−5
LFLocal	9.85	3.44E−3	7.43	1.11E−2	6.33	1.03E−2
LFAheadLocal	8.78	2.36E−3	7.67	4.11E−3	5.66	7.43E−3
RndLocal	10.34	1.81E−2	7.58	2.61E−2	6.04	2.79E−2
LFVNS	7.96	1.14E−2	6.20	2.04E−2	4.80	2.61E−2
LFAheadVNS	8.78	3.50E−3	7.11	8.18E−3	5.26	1.44E−2
RndVNS	8.08	2.52E−2	6.49	3.63E−2	5.13	3.80E−2
LFIVNS	2.50	3.93E+0	2.28	5.45E+0	1.75	6.29E+0
LFAheadIVNS	2.60	3.89E+0	2.30	5.41E+0	1.91	6.15E+0
RndIVNS	2.51	3.95E+0	2.30	5.49E+0	1.83	6.32E+0
ILP	0.00	2.26E+0	0.00	6.67E+1	0.03	3.98E+2

for some tests. Still, ILP delivers the best solutions for the analysed instances. In the group of simple heuristics, the best results are achieved by LFAhead, followed by LF. Both these algorithms deliver much better results than Rnd. Algorithms LFLocal and LFAheadLocal do not gain much in comparison to their initial schedules. Local search brings significant improvement only when it starts from a random solution. The quality of schedules delivered by RndLocal is similar to those of LFLocal and LFAheadLocal. Using variable neighborhood search gives better results, but the difference between local search and variable neighborhood search algorithms is not very large. All iterated variable neighborhood search algorithms obtain very good schedules, with average errors below 3%. Thus, the random shake step in IVNS is indeed helpful in moving from local minima to substantially better solutions. The choice of the initial schedule does not seem very important for IVNS algorithms, as all variants deliver solutions of similar quality in similar time. It is interesting that although the instance size increases with growing p_i, because the time horizon T gets larger, the quality of schedules delivered by heuristic algorithms improves with increasing p_i.

The results obtained for Rnd instances with 30 jobs and $p_1 \neq p_2$ are shown in Table 2. The heuristic algorithms achieve here smaller errors than in the case of $p_1 = p_2$. Thus, it seems that instances with $p_1 \neq p_2$ are easier to solve. Indeed, if $p_2 < p_1$, then the second machine is idle for at least $p_1 - p_2$ time units after executing each job. As a single job always fits in the storage space, the buffer limit is automatically observed in such periods, and hence, it may be easier to construct a good solution. Moreover, the relative distance between the upper bound $n(p_1 + p_2)$ on the schedule length and the trivial lower bound $np_1 + p_2$ is smaller when the difference between p_1 and p_2 is larger. The relationships between individual algorithms are similar to the case of $p_1 = p_2$. The average errors of all IVNS algorithms are again smaller than 3%.

Table 2 Average percentage distance from *LB* and running time for $n = 30$, $p_1 \neq p_2$, Rnd instances

Algorithm	$p_1 = 2, p_2 = 1$		$p_1 = 4, p_2 = 2$		$p_1 = 5, p_2 = 2$	
	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
LF	8.25	1.76E−5	7.15	2.04E−5	6.31	2.10E−5
LFAhead	7.01	2.52E−5	6.74	3.40E−5	5.34	3.25E−5
Rnd	19.71	1.82E−5	15.23	1.81E−5	12.95	2.00E−5
LFLocal	7.57	2.68E−3	6.24	8.50E−3	5.45	7.62E−3
LFAheadLocal	6.76	2.00E−3	6.20	3.54E−3	4.95	3.73E−3
RndLocal	6.75	2.08E−2	6.01	2.44E−2	4.46	2.30E−2
LFVNS	5.61	1.09E−2	5.19	2.07E−2	4.35	1.80E−2
LFAheadVNS	6.62	3.26E−3	5.58	7.52E−3	4.39	8.20E−3
RndVNS	6.40	2.21E−2	5.02	3.01E−2	4.01	3.16E−2
LFIVNS	2.19	3.91E+0	1.88	4.98E+0	1.62	4.99E+0
LFAheadIVNS	2.38	3.90E+0	1.75	4.90E+0	1.69	4.95E+0
RndIVNS	2.09	3.96E+0	1.79	5.02E+0	1.79	4.94E+0
ILP	0.00	8.83E+0	0.04	1.60E+2	0.02	1.67E+2

Table 3 Average percentage distance from *LB* and running time for $n = 30$, $p_1 = p_2$, Inc instances

Algorithm	$p_1 = p_2 = 1$		$p_1 = p_2 = 3$		$p_1 = p_2 = 5$	
	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
LF	5.48	1.78E−5	4.34	2.06E−5	5.76	2.31E−5
LFAhead	16.35	2.81E−5	15.86	3.62E−5	16.63	4.29E−5
Rnd	26.00	1.77E−5	26.27	1.92E−5	26.00	1.93E−5
LFLocal	4.94	6.40E−3	3.37	5.39E−3	3.69	8.40E−3
LFAheadLocal	11.43	5.64E−3	9.60	8.27E−3	10.30	9.43E−3
RndLocal	8.45	1.65E−2	6.83	1.74E−2	6.76	1.83E−2
LFVNS	3.98	1.03E−2	2.94	8.99E−3	3.13	1.13E−2
LFAheadVNS	10.32	8.80E−3	8.75	1.20E−2	8.34	1.70E−2
RndVNS	5.84	2.43E−2	4.76	2.51E−2	4.76	2.83E−2
LFIVNS	1.03	3.29E+0	0.59	3.06E+0	0.50	3.20E+0
LFAheadIVNS	0.73	3.28E+0	0.62	3.03E+0	0.50	3.30E+0
RndIVNS	0.96	3.25E+0	0.59	3.17E+0	0.52	3.25E+0
ILP	0.00	7.49E+0	0.00	1.99E+2	0.03	5.92E+2

Table 3 contains the results obtained for Inc instances with 30 jobs and $p_1 = p_2$. For such tests, the best simple heuristic is LF, which delivers solutions below 6% from the optimum on average. The schedules constructed by LFAhead are much worse, although not as bad as random solutions. The local search and variable neighborhood search algorithms perform best when starting from an LF schedule. Using a random initial solution leads to substantially better results than starting with an LFAhead

Table 4 Average percentage distance from *LB* and running time for $n = 30$, $p_1 \neq p_2$, Inc instances

Algorithm	$p_1 = 2, p_2 = 1$		$p_1 = 4, p_2 = 2$		$p_1 = 5, p_2 = 2$	
	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
LF	3.52	1.71E−5	4.29	1.97E−5	3.71	1.91E−5
LFAhead	9.29	2.72E−5	9.79	3.12E−5	8.35	3.11E−5
Rnd	15.87	1.80E−5	16.29	1.69E−5	13.57	1.88E−5
LFLocal	2.30	3.86E−3	3.22	5.33E−3	3.03	3.79E−3
LFAheadLocal	6.66	5.33E−3	6.94	6.36E−3	5.65	7.24E−3
RndLocal	4.23	1.58E−2	5.06	1.46E−2	4.05	1.51E−2
LFVNS	2.03	7.42E−3	2.32	9.20E−3	2.38	7.61E−3
LFAheadVNS	6.07	8.48E−3	6.39	9.60E−3	4.99	1.05E−2
RndVNS	3.35	2.41E−2	3.08	2.35E−2	2.59	2.42E−2
LFIVNS	0.32	3.00E+0	0.59	2.95E+0	0.55	2.98E+0
LFAheadIVNS	0.41	3.03E+0	0.42	3.04E+0	0.53	2.93E+0
RndIVNS	0.32	3.01E+0	0.40	2.99E+0	0.53	2.94E+0
ILP	0.00	1.91E+1	0.00	3.48E+2	0.18	8.50E+2

Table 5 Average percentage distance from *LB* and running time for $n = 30$, $p_1 = p_2$, Dec instances

Algorithm	$p_1 = p_2 = 1$		$p_1 = p_2 = 3$		$p_1 = p_2 = 5$	
	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
LF	29.90	1.85E−5	32.39	2.21E−5	32.34	2.63E−5
LFAhead	10.60	2.28E−5	11.30	2.59E−5	9.87	3.02E−5
Rnd	44.43	1.76E−5	47.57	1.91E−5	47.66	2.09E−5
LFLocal	26.56	3.73E−3	28.26	4.02E−3	21.44	8.46E−3
LFAheadLocal	10.60	1.75E−3	11.30	1.77E−3	9.87	1.93E−3
RndLocal	12.28	1.49E−2	14.63	1.49E−2	14.90	1.52E−2
LFVNS	9.53	1.09E−2	11.06	1.30E−2	10.15	1.53E−2
LFAheadVNS	10.17	2.68E−3	11.30	2.80E−3	9.55	3.59E−3
RndVNS	9.49	1.84E−2	11.05	2.00E−2	11.03	2.23E−2
LFIVNS	3.16	2.56E+0	4.68	2.80E+0	4.46	3.07E+0
LFAheadIVNS	2.87	2.53E+0	4.68	2.77E+0	4.07	3.04E+0
RndIVNS	3.20	2.56E+0	4.68	2.79E+0	4.74	3.07E+0
ILP	0.10	2.52E+2	0.75	1.58E+3	1.21	1.87E+3

schedule. All variants of iterated variable neighborhood search obtain similar results and achieve very high quality, as their average errors are below 1% in almost all cases.

Table 4 presents the results for Inc instances with $n = 30$ and $p_1 \neq p_2$. Similarly to the case of Rnd tests, the reported errors are smaller for $p_1 \neq p_2$ than for $p_1 = p_2$. Apart from this, no significant differences between these two groups of tests with an increasing $\Omega(t)$ function can be seen.

Table 6 Average percentage distance from *LB* and running time for $n = 30$, $p_1 \neq p_2$, Dec instances

Algorithm	$p_1 = 2, p_2 = 1$		$p_1 = 4, p_2 = 2$		$p_1 = 5, p_2 = 2$	
	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
LF	13.70	1.88E−5	14.64	2.00E−5	14.50	2.12E−5
LFAhead	5.49	2.25E−5	5.78	2.45E−5	5.96	2.57E−5
Rnd	22.97	1.90E−5	23.00	2.16E−5	18.90	1.90E−5
LFLocal	11.80	4.55E−3	13.75	2.75E−3	14.05	2.39E−3
LFAheadLocal	5.49	1.73E−3	5.78	1.71E−3	5.96	1.71E−3
RndLocal	6.46	1.49E−2	7.72	1.47E−2	6.17	1.49E−2
LFVNS	5.23	1.15E−2	5.52	1.05E−2	5.50	9.76E−3
LFAheadVNS	5.49	2.50E−3	5.78	2.63E−3	5.96	2.62E−3
RndVNS	5.06	1.92E−2	5.45	1.87E−2	5.63	1.81E−2
LFIVNS	1.77	2.45E+0	1.69	2.58E+0	2.24	2.56E+0
LFAheadIVNS	1.51	2.44E+0	1.74	2.57E+0	2.20	2.52E+0
RndIVNS	1.82	2.44E+0	2.10	2.58E+0	2.20	2.54E+0
ILP	0.15	4.73E+2	0.08	9.63E+2	0.97	1.92E+3

The results obtained for Dec instances with $n = 30$, $p_1 = p_2$ are shown in Table 5. This time, the best results among the simple heuristics are delivered by LFAhead, while the schedules produced by LF are much worse. Moreover, LFLocal delivers significantly worse results than RndLocal. However, the distance between VNS variants is small. The quality of results produced by LFAheadLocal is the same as that of LFAhead. Thus, it seems that LFAhead schedules are local optima with respect to the neighborhood used in the proposed local search procedure. Using additional neighborhoods in LFAheadVNS gives only very small improvements. In consequence, for $p_i < 5$, LFAheadVNS is outperformed by LFVNS and RndVNS. All IVNS algorithms deliver again similar results, with average errors below 5%. The results produced by all heuristics are generally worse than for Rnd and Inc instances with corresponding p_i values. Moreover, the running time of ILP is the longest for Dec tests. Thus, the instances with a decreasing $\Omega(t)$ function seem the most difficult to solve.

Table 6 presents the results obtained for Dec instances with 30 jobs and $p_1 \neq p_2$. Once again, it is confirmed that it is easier to produce good schedules when the execution times of the two operations of a job are different. The obtained errors are about two times smaller than for Dec instances with $p_1 = p_2$. In particular, the average errors of IVNS algorithms are less than 2.5%. The simple heuristic LFAhead delivers good schedules with average errors smaller than 6%, which are not improved by local search or variable neighborhood search.

The experimental results for tests with $n = 100$ will be presented only for values $(p_1, p_2) \in \{(1, 1), (4, 2), (5, 5)\}$, representing short and long jobs, as well as equal and different execution times of the two operations of a job. The results obtained for Rnd instances are shown in Table 7. For $p_1 = p_2 = 1$, the results of most heuristic algorithms are slightly worse than for the corresponding instances with 30 jobs. The quality loss is the largest for Rnd and iterated variable neighborhood search algorithms.

Table 7 Average percentage distance from *LB* and running time for $n = 100$, Rnd instances

Algorithm	$p_1 = p_2 = 1$		$p_1 = 4$, $p_2 = 2$		$p_1 = p_2 = 5$	
	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
LF	11.76	7.83E−5	27.50	1.00E−4	73.31	1.70E−4
LFAhead	10.38	1.57E−4	27.57	2.08E−4	71.59	3.97E−4
Rnd	35.64	3.17E−5	39.82	3.20E−5	88.69	3.77E−5
LFLocal	11.25	4.97E−2	26.97	3.92E−1	71.29	9.70E−1
LFAheadLocal	9.89	3.30E−2	26.79	1.02E−1	70.51	3.37E−1
RndLocal	11.74	1.17E+0	28.12	1.17E+0	74.07	8.28E−1
LFVNS	9.80	3.67E−1	26.36	1.22E+0	69.34	2.99E+0
LFAheadVNS	9.74	7.06E−2	26.53	2.73E−1	69.53	1.16E+0
RndVNS	10.19	1.59E+0	26.37	2.47E+0	69.70	3.77E+0
LFIVNS	5.89	3.20E+2	23.84	4.87E+2	65.73	7.51E+2
LFAheadIVNS	5.99	3.25E+2	23.84	4.87E+2	65.56	7.44E+2
RndIVNS	6.08	3.23E+2	23.93	4.81E+2	65.65	7.47E+2
ILP	0.26	1.14E+3	23.99	3.61E+3	64.61	3.61E+3

All variants of IVNS produce schedules around 6% longer than the lower bound. The errors obtained for $p_i > 1$ are much larger than for the corresponding instances with $n = 30$. Recall that these errors are computed with respect to the lower bound *LB*. For instances with 100 jobs and $p_i > 1$, Gurobi was not able to find good lower bounds within the imposed time limit, and hence, the trivial lower bound $np_1 + p_2$ had to be used. Thus, the distance between the lower bound and the actual optimum is very large, and this is the key factor determining the reported error values. For $p_i > 1$, IVNS algorithms reach solution quality similar to that of ILP, in a much shorter time. No instances with $n = 100$ and $p_i > 1$ were solved to the optimum by ILP. Note that although the time for solving the integer linear program was limited to 1 h, the algorithm running time also includes building the program and retrieving the schedule found. Hence, the average ILP times reported in Table 7 are slightly greater than 1 h when $p_i > 1$.

The results obtained for Inc instances with 100 jobs are shown in Table 8. In this group, no tests were solved to optimality by ILP, even for $p_1 = p_2 = 1$. Thus, it seems that instances with increasing storage availability are harder to solve by ILP than those with random buffer changes. This is also confirmed by the fact that in the group of tests with $n = 30$, the average ILP execution time was longer for Inc instances than for the corresponding Rnd instances (see Tables 1, 2, 3, 4). For $p_1 = p_2 = 1$, the average error of ILP solutions is approximately 4%, and the IVNS schedules are about 5% longer than the lower bound. For larger p_i, the reported errors strongly increase, reflecting growing distance between the lower bounds found and the optimal solutions. All variants of iterated variable neighborhood search significantly outperform ILP when $p_i > 1$, achieving both better solution quality and shorter execution time.

Table 9 presents the results obtained for Dec instances with $n = 100$. No tests in this group were solved to the optimum by ILP. The average ILP error reaches 15%

Table 8 Average percentage distance from *LB* and running time for $n = 100$, Inc instances

Algorithm	$p_1 = p_2 = 1$		$p_1 = 4, p_2 = 2$		$p_1 = p_2 = 5$	
	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
LF	9.06	6.45E−5	22.65	8.39E−5	45.89	3.52E−4
LFAhead	21.55	1.58E−4	30.35	2.07E−4	65.75	3.69E−4
Rnd	32.61	2.98E−5	38.50	3.34E−5	78.42	3.57E−5
LFLocal	9.01	1.85E−1	22.14	2.15E−1	45.18	2.99E−1
LFAheadLocal	20.57	6.11E−2	28.15	2.38E−1	59.24	6.72E−1
RndLocal	12.74	1.16E+0	22.85	7.60E−1	49.74	7.76E−1
LFVNS	8.25	3.80E−1	21.34	4.90E−1	44.36	7.12E−1
LFAheadVNS	20.12	1.55E−1	27.69	4.48E−1	57.93	1.36E+0
RndVNS	10.51	2.04E+0	21.29	1.70E+0	46.01	2.20E+0
LFIVNS	5.06	1.68E+2	18.59	1.44E+2	41.06	2.05E+2
LFAheadIVNS	5.08	1.75E+2	18.49	1.52E+2	41.12	2.11E+2
RndIVNS	4.98	1.71E+2	18.49	1.46E+2	41.19	2.02E+2
ILP	4.02	3.60E+3	24.88	3.61E+3	53.24	3.61E+3

Table 9 Average percentage distance from *LB* and running time for $n = 100$, Dec instances

Algorithm	$p_1 = p_2 = 1$		$p_1 = 4, p_2 = 2$		$p_1 = p_2 = 5$	
	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
LF	52.00	7.08E−5	28.66	8.76E−5	56.90	1.39E−4
LFAhead	23.33	1.15E−4	16.12	1.48E−4	24.95	1.88E−4
Rnd	64.98	3.07E−5	34.99	3.18E−5	69.04	3.74E−5
LFLocal	51.66	2.84E−2	28.64	2.84E−2	53.99	1.05E−1
LFAheadLocal	23.33	2.18E−2	16.12	2.31E−2	24.95	3.06E−2
RndLocal	24.74	5.68E−1	15.19	5.69E−1	26.37	7.20E−1
LFVNS	24.39	3.50E−1	16.17	3.04E−1	26.73	4.93E−1
LFAheadVNS	23.33	4.75E−2	16.12	5.34E−2	24.95	9.29E−2
RndVNS	20.04	1.05E+0	14.00	9.68E−1	22.51	1.24E+0
LFIVNS	15.90	8.96E+1	11.21	8.87E+1	17.43	1.25E+2
LFAheadIVNS	15.65	8.95E+1	11.27	8.87E+1	17.52	1.25E+2
RndIVNS	15.74	8.97E+1	11.23	8.86E+1	17.23	1.26E+2
ILP	15.09	3.60E+3	13.21	3.61E+3	25.44	3.61E+3

already for instances with $p_1 = p_2 = 1$, which suggests that the case of decreasing buffer availability is the hardest to solve for this algorithm. This conforms with the earlier observation that the Dec tests seem the most difficult in the group of instances with $n = 30$. The iterated variable neighborhood search algorithms achieve much better results than ILP for $p_i > 1$. It may seem surprising that the errors obtained by VNS, IVNS and ILP algorithms for Dec instances with $p_1, p_2 > 1$ are smaller than for the corresponding Rnd or Inc instances. However, this can be explained by the fact that in Dec instances, the available buffer is large at the beginning of the scheduling period, and hence, the optimum schedules are shorter than in the case of Rnd or Inc instances. Hence, the distance between the optimum and the lower bound $np_1 + p_2$, which is the main factor influencing the reported errors, is smaller for Dec instances. Obtaining smaller errors for $p_1 = 4$, $p_2 = 2$ than for $p_1 = p_2 = 1$ confirms the earlier observation that tests with $p_1 \neq p_2$ are easier to solve than those with $p_1 = p_2$. This effect is not visible for Rnd and Inc instances with $n = 100$ and $p_1 = 4$, $p_2 = 2$, because the distance between the lower bound and the optimal solution increases fast with growing job execution times when the $\Omega(t)$ function is not decreasing.

The results of the performed computational experiments can be summarised as follows. The optimal solutions can be found using ILP, but at a high computational cost, which seems the largest in the case of decreasing $\Omega(t)$, and the smallest in the case of random buffer changes. If a schedule has to be found very fast, one of algorithms LF and LFAhead can be used. LF is suitable for increasing buffer size, and LFAhead should be used when the storage space function is decreasing or random. For $n = 30$, choosing a correct simple heuristic usually leads to obtaining solutions at most 10% from the lower bound. Better results can be obtained using local search, and variable neighborhood search provides further improvements. However, for some types of instances, choosing a good initial schedule may be necessary to obtain high quality results using these algorithms. Among the proposed heuristics, the best results are delivered by iterated variable neighborhood search, in reasonable time. Moreover, the choice of the initial schedule has a very small impact on the performance of IVNS, and hence, no additional knowledge about the instance is required to achieve high quality solutions. For the largest analysed instances, iterated variable neighborhood search produces better results than ILP with 1 h time limit.

8 Conclusions

This paper analyses makespan minimisation in two-machine flow shops with job-dependent storage requirements and storage availability changing in time. It shows that the problem is strongly NP-hard even in the case when the duration of each operation is one unit of time. The existence of optimal permutation no-wait schedules is proved for the case when the smallest processing time on the first-stage machine is greater than or equal to the largest processing time on the second-stage machine, and the case when the smallest processing time on the second-stage machine is greater than or equal to the largest processing time on the first-stage machine. For the case where all jobs have the same processing time on the first machine, and the same processing time on the second machine, the paper presents a polynomial-time approximation scheme and several heuristic algorithms. Computational experiments show that iterated variable neighborhood search algorithms are a good tool for solving the considered problem. The average relative errors obtained by all variants of IVNS for instances with $n = 30$ are below 5%. For the most difficult instances with 100 jobs, the performance of IVNS is hard to estimate because good lower bounds cannot be found, but the delivered solutions are close to or better than those produced by ILP. Future research should include the worst-case analysis of the approximation algorithms. An interesting open question is whether problem $F2|storage, \omega_j, \Omega(t), p_{i,j} = 1|C_{max}$ remains strongly NP-hard when $\Omega(t)$ is a monotonic function.

References

1. Berlińska, J.: Heuristics for scheduling data gathering with limited base station memory. Ann. Oper. Res. **285**, 149–159 (2020)
2. Berlińska, J., Kononov, A., Zinder, Y.: Two-machine flow shop with a dynamic storage space and UET operations. In: World Congress on Global Optimization, pp. 1139–1148. Springer, Berlin (2019)
3. Błażewicz, J., Kubiak, W., Szwarcfiter, J.: Scheduling unit-time tasks on flow-shops under resource constraints. Ann. Oper. Res. **16**(1), 255–266 (1988)
4. Błażewicz, J., Lenstra, J.K., Rinnooy Kan, A.: Scheduling subject to resource constraints: classification and complexity. Discrete Appl. Math. **5**(1), 11–24 (1983)
5. Brucker, P., Knust, S.: Complex Scheduling. GOR-Publications. Springer, Berlin (2012)
6. Emmons, H., Vairaktarakis, G.: Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications. Springer, Berlin (2013)
7. Fung, J., Singh, G., Zinder, Y.: Capacity planning in supply chains of mineral resources. Inf. Sci. **316**, 397–418 (2015)
8. Fung, J., Zinder, Y.: Permutation schedules for a two-machine flow shop with storage. Oper. Res. Lett. **44**(2), 153–157 (2016)
9. Fung, J., Zinder, Y., Singh, G.: Flexible flow shop with storage: Complexity and optimisation methods. IFAC-PapersOnLine **49**(12), 237—242 (2016). 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016 Troyes, France, 28–30 June 2016
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
11. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Annals of Discrete Mathematics, vol. 5, pp. 287–326. Elsevier, Amsterdam (1979)
12. 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)
13. 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: Liefooghe, A., Paquete, L. (eds.) Evolutionary Computation in Combinatorial Optimization, pp. 50–65. Springer, Cham (2019)
14. 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)
15. 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 (2020)
16. Röck, H.: Some new results in flow shop scheduling. Z. Oper. Res. **28**(1), 1–16 (1984)
17. Süral, H., Kondakci, S., Erkip, N.: Scheduling unit-time tasks in renewable resource constrained flowshops. Z. Oper. Res. **36**(6), 497–516 (1992)