Iterated Local Search and Other Algorithms for Buffered Two-Machine Permutation Flow Shops with Constant Processing Times on One Machine

Hoang Thanh Le
Philine Geser
Martin Middendorf
Department of Computer Science, Leipzig University, Leipzig, 04109, Germany

https://doi.org/10.1162/evco_a_00287

Abstract
The two-machine permutation flow shop scheduling problem with buffer is studied for the special case that all processing times on one of the two machines are equal to a constant \( c \). This case is interesting because it occurs in various applications, for example, when one machine is a packing machine or when materials have to be transported. Different types of buffers and buffer usage are considered. It is shown that all considered buffer flow shop problems remain \( NP \)-hard for the makespan criterion even with the restriction to equal processing times on one machine. However, the special case where the constant \( c \) is larger or smaller than all processing times on the other machine is shown to be polynomially solvable by presenting an algorithm (2BF-OPT) that calculates optimal schedules in \( O(n \log n) \) steps. Two heuristics for solving the \( NP \)-hard flow shop problems are proposed: (i) a modification of the commonly used NEH heuristic (mNEH) and (ii) an Iterated Local Search heuristic (2BF-ILS) that uses the mNEH heuristic for computing its initial solution. It is shown experimentally that the proposed 2BF-ILS heuristic obtains better results than two state-of-the-art algorithms for buffered flow shop problems from the literature and an Ant Colony Optimization algorithm. In addition, it is shown experimentally that 2BF-ILS obtains the same solution quality as the standard NEH heuristic, however, with a smaller number of function evaluations.

Keywords
Flow shops with buffers, iterated local search, NEH heuristic, permutation schedules, makespan minimization.

1 Introduction

A flow shop problem is a scheduling problem where each job (or object) goes through a sequence of \( m \) processing steps such that the \( i \)th processing step is done on machine \( M_i \) for \( i \in \{1, \ldots, m\} \). The objective is to schedule the processing of jobs such that certain constraints are satisfied and certain criteria relating to efficiency, job throughput or incurred costs are optimized. A typical constraint is that the schedule has to be a permutation schedule where the sequence of jobs is the same on all machines. The most often studied optimization criterion is to minimize the makespan, that is, the total length of the schedule. In this article, we study the case of permutation scheduling for two machines under the makespan criterion.
The flow shop problems studied here have buffer constraints where each job takes up a given amount of buffer space and the jobs are to be scheduled such that the available buffer capacity is not exceeded. Buffer constraints occur in many practical applications where not only the time, but also the size of the processed objects has to be taken into account as, for example, in production facilities with limited storage space. In particular, we consider two types of buffer constraints that have been studied for flow shops in the literature and which differ by the time span in which the buffer is occupied by a job. Flow shops of these types occur, for example, in waste processing or the production of metal parts for the aerospace industry (Martinez et al., 2006), supply chains for mineral resources (Ernst et al., 2019), or in the production of chemicals (Liu and Kozan, 2009).

In this article, we study the special case of buffered two-machine flow shops where for one of the two machines the processing times for all jobs are equal to a constant $c$. This case is relevant because there exist many applications where in one stage of the flow shop the processing steps tend to be relatively similar for all objects. An example is a two-machine flow shop where different objects are produced on the first machine and packed on the second machine. In many applications of this type the production times might vary but packing is often a standardized process which approximately takes the same time for each object. Another example is the case of a robot (which corresponds to the first machine) that transports material for the production of different objects to the second machine where the manufacturing takes place. In many applications of this type the transportation times! might be similar, for example, when all materials are stored in the same storage hall or loaded onto the same vehicle, whereas the production times might differ strongly. In other applications of this type the opposite might hold: the transportation times are different, whereas all production times are similar.

Application examples can also be found outside the area of production processes, for example, in the processing of multimedia files as described by Lin et al. (2013) and Kononova and Kochetov (2013), where the downloading of files with different sizes corresponds to different processing times on the first machine, followed by a second operation that takes a fixed amount of time, such as processing the files’ metadata or creating file previews of fixed length while the available computer memory has to be taken into account.

The studied problem is also relevant from a theoretical perspective as it explores the computational complexity of buffer flow shops, especially the boundary between $NP$-hard and efficiently solvable problems, similar to the analyses of flow shops with buffers by Kononov et al. (2019) and Ernst et al. (2019). For the studied buffered two-machine flow shop problems it is shown that they remain $NP$-hard for the makespan criterion even under the restriction to equal processing times on one machine. In addition, it is shown that two special cases can be solved optimally in polynomial time, namely, the cases where the value of $c$ describing the constant processing time on one of the machines is smaller (or larger) than all processing times on the other machine. These special cases are relevant for various applications, for example, when the two machines are of different types, such as a production machine and a packaging machine that have different processing times. In this example, it is possible that the constant packaging times are relatively small (or large) in comparison to the production times.

To solve the general case we propose a modification of the well-known NEH heuristic as well as an Iterated Local Search heuristic (2BF-ILS) that uses the modified NEH heuristic for computing its initial solution. 2BF-ILS is studied experimentally and
compared with two state-of-the-art heuristics for buffered flow shops from the literature as well as an Ant Colony Optimization algorithm. 2BF-ILS is also experimentally compared to the standard NEH heuristic.

Note that this article is an extension of Le et al. (2019). New is a detailed NP-completeness proof for the considered problems and the identification of the aforementioned efficiently solvable subcases by presenting an $O(n \log n)$ algorithm and proving its correctness. The extension also incorporates new experiments using an additional set of instances with characteristics commonly occurring in practical applications and features a more detailed comparison between the optimization behavior of the proposed algorithm 2BF-ILS and the other algorithms. Also, additional literature on more recent developments regarding heuristic, approximate methods and theoretical results for two-machine buffer flow shops and its variants, as well as extended explanations have been integrated.

The remainder of this article is structured as follows. A short overview of recent works on buffer-constrained flow shop problems is given in Section 2. A formal description of the considered flow shop problems with buffers is presented in Section 3. The NP-completeness results are shown in Section 4. The two special cases that are polynomial-time solvable are discussed in Section 5. The modified NEH heuristic is described in Section 6 and the Iterated Local Search heuristic is introduced in Section 7. Experimental results are presented in Section 8. Conclusions are given in Section 9.

2 Related Work

The buffered flow shop problems that are most often studied in the literature have a buffer between each two adjacent machines $M_i$ and $M_{i+1}$ and each job occupies the buffer after it finishes on $M_i$ and until its processing starts on $M_{i+1}$. This type of buffer is called intermediate buffer. It is known that the flow shop problem with intermediate buffers is NP-complete (Papadimitriou and Kanellakis, 1980) for two or more machines. Several authors have investigated heuristics for this problem, one of the earliest works being Leisten (1990) where a systematic overview for formalizing intermediate buffer flowshop problems is presented and several heuristics originating from infinite-buffer flow shops are tested with the NEH heuristic obtaining the best results.

Investigated heuristics also include a Tabu Search (Li and Tang, 2005), Variable Neighborhood Search (Moslehi and Khorasanian, 2014), Genetic Algorithms (Wang et al., 2006), methods based on Differential Evolution (Pan, Wang, Gao, and Li, 2011), and Particle Swarm Optimization (Liu et al., 2008). Other examples of biologically inspired algorithms that have been applied on buffer flow shops are a Discrete Artificial Bee Colony (Zhang and Gu, 2015) and Immune System algorithms (Hsieh et al., 2009). In most of these studies the performance of different heuristics for flow shops with intermediate buffers has been compared. Figure 1 summarizes the results of these comparisons.

Another type of buffer constraint that is considered in the literature is where the buffer between machines $M_i$ and $M_{i+1}$ is used by each job from its starting time on $M_i$ until its finishing time on $M_{i+1}$. This type of buffer is in the following referred to as spanning buffer. The flow shop problem with spanning buffer is also NP-complete (Lin et al., 2009). Examples for methods from the literature for this flow shop type are a Variable Neighborhood Search by Kononova and Kochetov (2013), where Integer Linear Programming was also used to solve small instances, a Branch-and-Bound algorithm
Figure 1: Overview of works containing comparisons between algorithms for buffered flow shops with intermediate buffers. The notation $A \leftarrow B$ indicates that algorithm $A$ is outperformed by algorithm $B$ in the given reference. However, the arrows are to be interpreted with some care: It does not mean that $A$ is worse than $B$ for two-machine flow shops with buffer constraints. It only means that $B$ obtained better results (on average) than $A$ for the considered test instances in the respective work.

used to calculate lower bounds and optimally solve small instances with up to 18 jobs (Lin et al., 2009) as well as a heuristic based on Lagrangian relaxation and bin packing (Kononov et al., 2019). This buffer type is also analyzed by Gu et al. (2018) for the total weighted completion time criterion where it is shown to be $NP$-hard, even if one of both permutations is fixed and Min et al. (2019) proved the $NP$-hardness for the case where the processing times on both machines are the same for each job. Another variant where the spanning buffer changes its capacity over time is also $NP$-hard even when all jobs have unit processing times, but smaller instances with up to 40 jobs can be optimally solved using Integer Linear Programming (Berlińska et al., 2019).

If the buffer size is infinite, the resulting flow shop problem is equivalent to a flow shop problem without buffer constraints which in the case with two machines can be optimally solved in polynomial time for the makespan criterion using Johnson’s algorithm (Johnson, 1954) while it is $NP$-hard for more than two machines (Garey et al., 1976). If the buffer has no capacity such that finished jobs on $M_i$ immediately block $M_i$ if $M_{i+1}$ is busy, one obtains the blocking flow shop which can also be solved in polynomial time for up to two machines (Gilmore and Gomory, 1964), but is otherwise $NP$-hard for the makespan criterion (G. Hall and Sriskandarajah, 1996).

The buffer-constrained flow shop has also been investigated with further additional constraints in the literature. A combination of different target criteria is investigated by Qian et al. (2009) and Liang et al. (2019) using Differential Evolution algorithms, whereas Liu and Kozan (2009) consider the combination of different buffer constraints in the same instance. Other extensions of the flow shop with buffer constraints include the processing of jobs in batches, for which an exact Branch-and-Bound algorithm for up to 40 jobs (Agnetis et al., 1998), an approximation algorithm (Pranzo, 2004), and a Differential Evolution heuristic (Fu et al., 2012) have been proposed. Another extension is the use of machines that can process multiple jobs at the same time. This type is known
as the parallel, flexible, or hybrid flow shop problem and for the buffer-constrained variant an Artificial Bee Colony algorithm is presented in Li and Pan (2015). Regarding the flexible flow shop with two machines and spanning buffer, it has been shown that this problem is \textit{NP}-hard (Fung et al., 2016). It is even \textit{NP}-hard to approximate within a factor of 4/3, even when all processing times are equal, but becomes polynomially solvable if the spanning buffers of each flow shop have the same capacity (Ernst et al., 2019). A more general case consisting of multiple parallel flow shops is considered by Zhang and Xing (2019) where a Differential Evolution heuristic is proposed.

3 Formal Description of the Problem

Each two-machine flow shop problem studied in this article has a given set \( J = \{J_1, \ldots, J_n\} \) of \( n \) jobs and two \textit{machines} \( M_1 \) and \( M_2 \). Each job \( J_i \) has two processing times \( a_i > 0, b_i > 0 \) where \( a_i \) is the processing time on \( M_1 \) and \( b_i \) is the processing time on \( M_2 \). In flow shops of this type each job has to be finished on machine \( M_1 \) before it can be started on machine \( M_2 \). The processing of a job on a machine is nonpreemptive and a machine can process at most one job at a time. A schedule for a flow shop with two machines is denoted by \( \sigma \) and specifies the starting time \( S_i^1(\sigma) \) and the completion time \( C_i^1(\sigma) \) of \( J_i \) on \( M_1 \) for each job \( J_i \) and each machine \( M_j \) \((i \in \{1, \ldots, n\}, j \in \{1, 2\})\). Thus, a schedule \( \sigma \) specifies \textit{permutations} \( \pi^1(\sigma), \pi^2(\sigma) \) describing the order of jobs on the machines \( M_1 \) and \( M_2 \), respectively. A schedule \( \sigma \) is a \textit{permutation schedule} if \( \pi^1(\sigma) = \pi^2(\sigma) \).

Each buffered two-machine flow shop problem has a buffer with a given capacity \( \Omega > 0 \) and each job \( J_i \in J \) has a size \( s_i \) which specifies how many buffer units \( J_i \) occupies when stored in the buffer.\(^1\) Two types of buffers are considered in this article which differ by the time span in which a job occupies the buffer for a given schedule \( \sigma \): (i) an \textit{intermediate buffer} is occupied by job \( J_i \) from \( C_i^1(\sigma) \) until \( S_i^2(\sigma) \) and (ii) a \textit{spanning buffer} is occupied by \( J_i \) from \( S_i^1(\sigma) \) until \( C_i^2(\sigma) \). A schedule \( \sigma \) is \textit{valid} for flow shops with buffer constraints when at every time step the total amount of buffer that is occupied by the jobs does not exceed \( \Omega \). If we impose the condition that all jobs are processed as early as possible, it is sufficient to specify the permutations \( \pi^1(\sigma) \) and \( \pi^2(\sigma) \) in order to define a schedule \( \sigma \). Figure 2 shows an example of a flow shop with an intermediate buffer (left) and a spanning buffer (right) together with a valid permutation schedule.

In this article, we study the special case of buffered two-machine flow shops where the processing times on one of the machines are the same for all jobs, that is, \( a_i = c \) or \( b_i = c \) for \( i \in \{1, \ldots, n\} \) with a constant \( c > 0 \). Regarding the buffer usage of the jobs, two common models are studied in the literature: (i) the buffer is interpreted as a “counter” for the number of stored jobs and thus \( s_i = 1 \) for \( i \in \{1, \ldots, n\} \), similar to Zhang and Gu (2015) and Moslehi and Khorasanian (2014), and (ii) the amount of storage that is used by a job is equal to its processing time on the first machine, i.e., \( s_i = a_i \) for \( i \in \{1, \ldots, n\} \). In the second model, which is found in Fung and Zinder (2016), Kononova and Kochetov (2013), Lin et al. (2009), and Min et al. (2019), the job size \( s_i \) is interpreted as being proportional to the amount of work during its initial processing step.

The criterion to be optimized is the makespan \( C_{\text{max}} \) (i.e., the time when the last job is completed on the last machine). This is also the criterion used by most of the studies mentioned in Section 2 and based on these works, we also restrict ourselves to permutation schedules (denoted by \textit{prmu}) where the sequence of jobs on both machines is the same. Hence, a schedule \( \sigma \) is determined by a single permutation \( \pi(\sigma) \) of all jobs.

\(^1\)The case \( \Omega = 0 \) leads to the blocking flow shop which is optimally solvable in polynomial time for the makespan criterion with two machines (Gilmore and Gomory, 1964) and thus not considered here.
Figure 2: Examples of a flow shop with an intermediate buffer (left, $\Omega = 3$) and a flow shop with a spanning buffer (right, $\Omega = 7$) and $s_i = a_i$ for $i \in \{1, \ldots, n\}$. In each case a valid permutation schedule $\sigma$ is shown with $\pi(\sigma) = (J_1, J_2, J_3, J_4, J_5)$. Blocks with diagonal lines indicate phases where $M_1$ is idle and the next job has to wait until enough buffer capacity becomes available.

Figure 3: Example for “mirroring” schedules in order to obtain the case $b_i = c, s_i = a_i$.

Depending on whether the buffer usage $s_i$ equals $a_i$, $b_i$, or 1 and whether the processing times on $M_1$ or $M_2$ are equal, it is possible to distinguish between six different cases. However, it can be shown that only two cases have to be considered since all other cases can be reduced to them. These two cases are (i) $b_i = c, s_i = a_i$ and (ii) $b_i = c, s_i = 1$. The reduction can be done by rescaling all values by $s_i$ (if $s_i$ is constant for all jobs) and/or by defining a new schedule $\sigma'$ with the same makespan that “mirrors” the processing times (i.e., $a'_i = b_i, b'_i = a_i$ and $s'_i = s_i$) as well as the time axis (see Figure 3 for an example).

It should be noted that permutation schedules impose a restriction for the cases where $s_i = a_i$; that is, there exist instances for which the set of optimal schedules does not contain a permutation schedule. An example, is the flow shop in Figure 3 (right) where the shown schedule with makespan 28 is a best permutation schedule but there exists a schedule $\sigma$ with makespan 27 for $\pi_1(\sigma) = (J_2, J_3, J_4, J_1)$ and $\pi_2(\sigma) = (J_2, J_4, J_3, J_1)$ that is not a permutation schedule. For the case $s_i = 1$, there always
exists a permutation schedule that is optimal (Geser, 2017). Adopting the commonly used notation introduced by Graham et al. (1979), we denote the problems considered here by

$$F2|prmu, b_i = c, \text{bufType, bufUsage}|C_{\max}, \tag{1}$$

with bufType ∈ \{spanningBuffer, intermediateBuffer\} and bufUsage ∈ \{s_i = 1, s_i = a_i\}.

4 NP-Completeness Results

It is shown in this section that all four types of flow shops considered in Eq. (1) are NP-complete. Due to space limitations, the proof is given only for the case with intermediate buffer and \(s_i = a_i\). The NP-completeness for the other considered flow shops can be shown with similar arguments.

**Theorem 4.1:** The decision problem if there exists a valid permutation schedule \(\sigma^*\) with \(C_{\max}(\sigma^*) \leq L\) for a given integer \(L\) is NP-complete for the following type of flow shops:

i) \(F2|prmu, b_i = c, \text{intermediateBuffer}, s_i = a_i|C_{\max}\),

ii) \(F2|prmu, b_i = c, \text{spanningBuffer}, s_i = a_i|C_{\max}\),

iii) \(F2|prmu, b_i = c, \text{intermediateBuffer}, s_i = 1|C_{\max}\),

iv) \(F2|prmu, b_i = c, \text{spanningBuffer}, s_i = 1|C_{\max}\).

**Proof (only for (i)):** The problem is in NP since it can be checked in polynomial time if a schedule \(\sigma^*\) is valid and satisfies \(C_{\max}(\sigma^*) \leq L\). To show the NP-hardness consider an instance of 3PARTITION: Given are positive integers \(x_1, x_2, \ldots, x_{3m}\) and an integer \(B > 0\) such that \(B/4 < x_j < B/2\) for all \(j\) and \(\sum_{j=1}^{3m} x_j = mB\). The question is if there exists a partition of \(\{x_1, x_2, \ldots, x_{3m}\}\) into \(m\) subsets \(S_1, S_2, \ldots, S_m\) such that each set \(S_k\) satisfies \(|S_k| = 3\) and \(\sum_{x \in S_k} x = B\). We denote the given instance from 3PARTITION as \(I_{3P}\). The corresponding flow shop instance \(I_{F2}\) of \(F2|prmu, b_i = c, \text{intermediateBuffer}, s_i = a_i|C_{\max}\) is constructed with a set \(\mathcal{J} = \{g_0\} \cup G \cup H\) of \(n = 4m + 1\) jobs where

- \(g_0\) is a job with \(a_{g_0} = B/4\),
- \(G\) is a set of \(m\) jobs \(g_1, g_2, \ldots, g_m\) with \(a_{g_k} = 3B\) for \(k \in \{1, 2, \ldots, m\}\), and
- \(H\) is a set of \(3m\) jobs \(h_1, h_2, \ldots, h_{3m}\) with \(a_{h_k} = x_k\) for \(k \in \{1, 2, \ldots, 3m\}\).

The additional parameters are \(\Omega = 3B/4, c = B, L = 4mB + B + B/4,\) and \(s_i = a_i\) for all jobs. This instance can be constructed in polynomial time. Now it is shown that \(I_{3P}\) has a solution if and only if \(I_{F2}\) has a permutation schedule \(\sigma^*\) with \(C_{\max}(\sigma^*) \leq L\).

"\(\leftarrow\)" Assume that \(I_{F2}\) has a permutation schedule \(\sigma^*\) with \(C_{\max}(\sigma^*) \leq L\). It is shown in the following that \(\sigma^*\) satisfies several properties that lead to a structure shown in Figure 4. These properties allow the construction of a solution for \(I_{3P}\). The first property results from the sum of all processing times on both machines, using \(\sum_{j=1}^{3m} x_j = mB\) and comparing the values to \(L\):

$$\sum_{j \in \mathcal{J}} a_j = \frac{B}{4} + m \cdot 3B + mB = 4mB + \frac{B}{4} = L - B$$

$$\sum_{j \in \mathcal{J}} b_j = B + mB + 3mB = 4mB + B = L - \frac{B}{4}.$$

Evolutionary Computation Volume 29, Number 3 421
This shows that $M_1$ cannot have a total idle time of more than $B$ time units and $M_2$ cannot be idle for more than $B/4$ time units without exceeding the maximum makespan $L$. Since $M_1$ is always idle for at least $B$ time units (while $M_2$ is processing its last job), it follows that $M_1$ cannot have any idle times before it finishes its last job.

As for $M_2$, the initial idle time interval (while $M_1$ processes its first job) cannot be shorter than $B/4$ time units and its length is equal to $B/4$ if and only if $g_0$ is the first job on both machines (a job $h_k \in H$ is not possible as the first job due to $B/4 < x_k < B/2$ being required for the corresponding 3PARTITION instance). Thus, the job $g_0$ has to be the first job on both machines and no idle time is allowed on $M_2$ between $g_0$ and the last job on $M_2$.

The second property is that it is not possible for the buffer to store any job $g \in G$ since $s_g = 3B > 3B/4 = \Omega$. Thus, the machine $M_2$ must immediately start any job $g \in G$ after it is finished on $M_1$: $C^1_{g_0}(\sigma^*) = S^2_{g_0}(\sigma^*)$. It is also necessary that $C^1_{g_0}(\sigma^*) = S^2_{g_0}(\sigma^*)$ or else the initial idle time would be exceeded. Since all jobs in $G$ are identical, we can assume without loss of generality that the $G$-jobs are processed in the order $g_1, \ldots, g_m$.

We now consider the sets $H_k \subseteq H$ of jobs processed between $g_k$ and $g_{k+1}$ (for $k \in \{0, 1, 2, \ldots, m-1\}$) and the set $H_m$ of jobs processed after $g_m$ on both machines. Note that the condition $B/4 < x_k < B/2$ for $k \in \{1, \ldots, 3m\}$ implies that it is not possible for the buffer to store more than two jobs from $H$ at any time. It is now shown that $|H_0| = 3$ and $\sum_{j \in H_0} a_j = B$ for $k \in \{0, 1, 2, \ldots, m-1\}$. Consider first $k = 0$.

Since no additional idle times are allowed on both machines, the equations $S^2_{g_1}(\sigma^*) = C^1_{g_0}(\sigma^*) + b_{g_0} + \sum_{h \in H_0} b_h = a_{g_0} + (1 + |H_0|)B$ and $C^1_{g_1}(\sigma^*) = a_{g_0} + \sum_{h \in H_0} a_h + a_{g_0}$ as well as $C^1_{g_0}(\sigma^*) = S^2_{g_0}(\sigma^*)$ lead to $\sum_{h \in H_0} a_h = (|H_0| - 2)B$ which cannot be negative or zero, implying $|H_0| \geq 3$.

Assume $|H_0| > 3$ and denote as $h^1, h^2, h^3, h^4 \in H$ the first four jobs in $H_0$ processed on $M_1$. Machine $M_2$ also has to process these jobs immediately after finishing $g_0$ or else it would be idle. Using the condition $B/4 < a_h < B/2$ for all $h \in H$, one obtains

$$C^1_{h^4}(\sigma^*) = C^1_{g_0}(\sigma^*) + \sum_{i=1}^{4} a_{h^i} = S^2_{g_0}(\sigma^*) + \sum_{i=1}^{4} a_{h^i} < S^2_{g_0}(\sigma^*) + \frac{2B}{b_{g_0} + b_{h^4}} = S^2_{h^4}(\sigma^*) .$$

This means that four jobs in $H^1_0$ are finished on $M_1$ before $M_2$ frees the buffer space occupied by $h^2$ such that at least three jobs from $H_0$ have to be stored in the buffer at time $t = C^1_{h^4}(\sigma^*)$. This exceeds the buffer capacity implying that $\sigma^*$ is not a feasible solution. It thus follows that $|H_0| = 3$.

In order to show $\sum_{j \in H_0} a_j = B$, we assume all other cases: If $\sum_{j \in H_0} a_j < B$, then the three jobs in $H_0$ would be finished on $M_1$ before $M_2$ finishes $g_0$ implying that they would have to be stored in the buffer. This is not possible since $\sum_{j \in H_0} s_j > (3/4)B$ and this would exceed the buffer capacity $\Omega$. For the case $\sum_{j \in H_0} a_j > B$, let $h^3$ be the job in
the set \( H_0 \) that \( M_2 \) processes last. It follows that
\[
C_{h^3}(\sigma^*) = S_{g_0}(\sigma^*) + \frac{B}{b_{h^0}} + \frac{3B}{\sum_{j \in H_0} b_j} = C_{g_0}(\sigma^*) + B + 3B
\]
which means that \( M_2 \) is idle (waiting for \( g_1 \)) after finishing \( h^3 \). This is a contradiction since there are no idle times between the jobs on \( M_2 \).

Hence, \(|H_0| = 3\) and \( \sum_{j \in H_0} a_j = B \) holds. This implies that \( M_1 \) finishes the set \( H_0 \) at the same time as \( M_2 \) is finishing \( g_0 \) and that the interval from \( C_{g_0}(\sigma^*) = S_{g_1}(\sigma^*) \) to \( C_{h^3}(\sigma^*) = S_{g_1}(\sigma^*) \) has a length of \( 3B \) time units. Thus, \( M_2 \) processes the jobs in \( H_0 \) during this interval or else it would be idle.

It can also be concluded that the buffer contains no jobs by the time \( M_1 \) finishes the job \( g_1 \). This allows us to apply the same arguments used in the case \( k = 0 \) for \( k \in \{1, 2, \ldots, m \} \) to show that \(|H_k| = 3\) as well as \( \sum_{j \in H_k} a_j = B \). By doing so, one obtains \( m \) sets \( H_0, H_1, H_2, \ldots, H_{m-1} \) that each contain three jobs from \( H \). Due to \(|H| = 3m\), it follows that \(|H_m| = 0\). Since \( \sum_{j \in H_k} a_j = B \) for \( k \in \{0, 1, 2, \ldots, m \} \) and since the jobs in \( H \) correspond to the numbers of the \( 3 \)-\textsc{Partition} problem \( I_{3,L} \), it is possible to construct a solution for \( I_{3,F} \).

"\( \Rightarrow \)" This follows that the subsets \( S_1, S_2, \ldots, S_m \) satisfying \( |S_k| = 3 \) and \( \sum_{x \in S_k} x = B \) for \( k \in \{1, 2, \ldots, m \} \), it is possible to construct a schedule \( \sigma^* \) as shown in Figure 4 where the \( H \)-jobs are corresponding to the subsets that are scheduled between the \( G \)-jobs. This leads to a schedule \( \sigma^* \) with \( C_{\text{max}}(\sigma^*) = L \) and a solution for \( I_{3,F} \).

5 Polynomial-Time Solvable Subcases

As shown in Section 4, the buffered flow shops with \( b_i = c \) are \( \text{NP} \)-complete for all buffer models and buffer usage values considered here. However, there exist special cases that are solvable in polynomial time, such as \( F2|\text{spanningBuffer}, s_i = a_i|C_{\text{max}} \) with the additional condition that both \( \max a_i \) and \( \max b_i \) do not exceed \( \Omega/5 \) (Kononov et al., 2019) and \( F2|\text{spanningBuffer}, s_i = a_i|C_{\text{max}} \) with the additional constraint \( \max a_i \leq \min b_i \) (Min et al., 2019).

In this work, the subcases where the constant \( c \) is either larger or smaller than all run times on \( M_1 \) are shown to be polynomially solvable. Note that in comparison to Min et al. (2019), we additionally consider both buffer types as well as the case \( \min a_i \geq \max b_i \) when all \( b_i \) are equal to \( c \). In order to show this property for \( F2|\text{prmu}, b_i = c, \text{spanningBuffer}, s_i = a_i, c \geq a_i|C_{\text{max}} \) the following lemmata are needed.

\textbf{Lemma 5.1:} For each schedule \( \sigma \) for an instance of \( F2|\text{prmu}, b_i = c, \text{spanningBuffer}, s_i = a_i, c \geq a_i|C_{\text{max}} \), there exists a schedule \( \sigma' \) with \( \pi(\sigma) = \pi(\sigma') \) and \( C_{\text{max}}(\sigma') \leq C_{\text{max}}(\sigma) \) satisfying \( C_k(\sigma') = S_k(\sigma') \) for all \( k \in \{1, 2, \ldots, n\} \).

\textbf{Proof:} For a given schedule \( \sigma \), assume that there exists a job \( J_k \) for which \( C_k(\sigma) < S_k(\sigma) \). W.l.o.g. let \( J_k \) be the rightmost such job. Then, since \( c \geq a_i \) for all \( i \in \{1, 2, \ldots, n\} \), it follows that each job which comes after \( J_k \) in \( \sigma \) starts later than \( S_k(\sigma) \). Hence, \( J_k \) can be rescheduled on \( M_1 \) to a new schedule \( \sigma' \) such that \( C_k(\sigma') = S_k(\sigma') \) holds without increasing the makespan. By iteration the lemma follows.

For the schedule \( \sigma' \) of Lemma 5.1 it holds that for each job \( J_k \) \( (k \in \{1, 2, \ldots, n\}) \) the predecessor job finishes at time \( t \leq S_k(\sigma') \) and the successor job of \( J_k \) starts at time \( t \).
$t \geq C^1_i(\sigma') = S^1_i(\sigma')$. Hence, the schedule $\sigma'$ has the property that at most two jobs are in the buffer at any given time. A permutation schedule is minimal if it has the smallest makespan out of all permutation schedules with the same order of jobs. The following lemma holds for minimal permutation schedules.

**Lemma 5.2:** For any permutation $\pi = (J_{n-1}, J_{n-2}, \ldots, J_1)$ of the jobs $J$ in an instance of $F2|\text{prmu}, b_i = c, \text{spanningBuffer}, s_i = a_i, c \geq a_i|C_{\text{max}}$ there exists a minimal permutation schedule $\sigma$ where for each job $J_{n+k}$ and its successor $J_{n+k+1}$ ($k \in \{1, 2, \ldots, n-1\}$) the permutation schedule $\sigma$ satisfies either

1. $C^2_{n+k}(\sigma) = S^2_{n+k+1}(\sigma)$ and $s_{n+k} + s_{n+k+1} \leq \Omega$ or
2. $C^2_{n+k}(\sigma) = S^1_{n+k}(\sigma)$ and $s_{n+k} + s_{n+k+1} > \Omega$.

**Proof:** The lemma is a direct consequence of Lemma 5.1 and the facts that it can be assumed that $\sigma$ is a minimal permutation schedule and $s_{n+k+1} \leq c$. □

Let $J$ be the set of jobs of an instance of $F2|\text{prmu}, b_i = c, \text{spanningBuffer}, s_i = a_i, c \geq a_i|C_{\text{max}}$. If $C^2_i(\sigma) = S^2_i(\sigma)$ holds for two jobs $J_i$, $J_1 \in J$ in a permutation schedule $\sigma$ we say that the processing time $a_1$ of $J_i$ on $M_1$ is hidden. Two jobs $J_i$, $J_1$ are compatible if $s_i + s_1 \leq \Omega$. For a permutation schedule $\sigma$ and its corresponding job permutation $\pi(\sigma)$ let $I(\sigma) \subseteq \{1, 2, \ldots, n\}$ be the set of indices $i$ of jobs $J_i$ which are not compatible with their predecessor, i.e., $s_{\pi_i-1} + s_{\pi_i} > \Omega$. The following corollary is a direct consequence of Lemma 5.2.

**Corollary 5.1:** Let $\sigma$ be a minimal permutation schedule for an instance of $F2|\text{prmu}, b_i = c, \text{spanningBuffer}, s_i = a_i, c \geq a_i|C_{\text{max}}$ and $\pi(\sigma) = (J_{\pi_1}(\sigma), J_{\pi_2}(\sigma), \ldots, J_{\pi_n}(\sigma))$ its corresponding permutation of jobs in $J$. Then, it holds that $C_{\text{max}}(\sigma) = a_{\pi_1} + \sum_{i \in I(\sigma)} a_i + n \cdot c$.

In the following we present Algorithm 1 which computes an optimal schedule for an instance of $F2|\text{prmu}, b_i = c, \text{spanningBuffer}, s_i = a_i, c \geq a_i|C_{\text{max}}$. It first requires that the jobs are sorted in order of decreasing $a_i$ which can be done in $O(n \log n)$ steps. In the case where the jobs are already sorted, the resulting algorithm runs in linear time.

In order to solve $F2|\text{prmu}, b_i = c, \text{spanningBuffer}, s_i = a_i, c \geq a_i|C_{\text{max}}$, Algorithm 1 considers two special cases first: (i) all jobs in $J$ are pairwise not compatible (lines 1 and 2) and (ii) all jobs in $J$ are pairwise compatible (lines 3 and 4). In case of (i) it is clear that none of the processing times on $M_1$ can be hidden. Hence, any minimal schedule $\sigma$ has the makespan $C_{\text{max}}(\sigma) = \sum_{i=1}^n a_i + n \cdot c$ and is thus optimal. In case (ii), it follows that $I(\sigma) = \emptyset$. Therefore, Corollary 5.1 implies that any minimal schedule $\sigma$ which has $J_n$ as its first job has the makespan $C_{\text{max}}(\sigma) = a_n + n \cdot c$ and is optimal since $a_n \leq a_i$ for $i \in \{1, 2, \ldots, n-1\}$.

If none of the cases (i) and (ii) hold, there exists a minimal $k \in \{2, 3, \ldots, n-1\}$ such that $J_{k+1}$ and $J_k$ are compatible. Since $a_{k+1} \geq a_i$ for $i > k + 1$, all jobs in $\{J_{k+1}, J_{k+2}, \ldots, J_n\}$ are compatible with $J_k$ (line 6). When the processing time $a_i$ of a job $J_k \in \{J_1, \ldots, J_k\}$ can only be hidden, then a job from $R := \{J_{k+1}, \ldots, J_n\}$ (line 7) is its predecessor. Clearly, in an optimal schedule the total processing time of jobs $\{J_1, \ldots, J_{k-1}\}$ on $M_1$ that can be hidden has to be maximal. To determine this maximum total processing time the following greedy approach is taken in lines 8–14. The jobs $J_1, \ldots, J_{k-1}$ are considered in this order and we introduce a function $f$ that assigns jobs from $R$ to these jobs as predecessors as follows.

For the next job $J_k$ the smallest job in $R$ that has so far not been assigned as predecessor to one of the jobs $J_1, \ldots, J_{k-1}$ and that is compatible with $J_k$ is always assigned.
as the predecessor of \( J_i \) (if it exists). If all jobs in \( \mathcal{R} \) have been assigned (line 15), the assignment \( f \) defines the only possible assignment of jobs in \( \mathcal{R} \) to jobs in \( \{ J_1, \ldots, J_k \} \) such that for the maximum number of jobs in \( \{ J_1, \ldots, J_k \} \) the processing time on \( M_1 \) can be hidden and also the total hidden processing time on \( M_1 \) of these jobs is maximized. That the greedy algorithm gives the optimal solution follows from the fact that the set of all subsets of \( \{ J_1, \ldots, J_k \} \) which can be hidden by assigning jobs from \( \mathcal{R} \) as predecessors (where each job in \( \mathcal{R} \) is assigned to at most one job in \( \{ J_1, \ldots, J_k \} \)) is an independence system over \( \{ J_1, \ldots, J_k \} \) and forms a matroid together with the used weighting function \( w(J_i) = a_i \) (see standard textbooks on greedy algorithms for details). It follows that any minimal schedule \( \sigma \) for the order \( (J_k, f(J_{k-1}), J_{k-1}, \ldots, f(J_1), J_1) \) has the makespan \( C_{\max}(\sigma) = a_k + \sum_{i \in I(\sigma)} a_i + n \cdot c \) and is optimal (line 16). An analogous result holds for jobs in \( \{ J_1, \ldots, J_k \} \) if there exists an \( i \) such that \( f(J_i) = \emptyset \) (line 18) and \( h \) is the maximum such \( i \) (line 19). In this case there exists an optimal schedule \( \sigma \) where \( (J(h), f(J_{h-1}), J_{h-1}, \ldots, f(J_1), J_1) \) is a suffix of the permutation \( \pi(\sigma) \).

In line 23 of Algorithm 1 it holds for the remaining jobs \( J' \) that the run time on \( M_1 \) can be hidden with jobs from \( \mathcal{R}' := \mathcal{R} \setminus \{ f(J_i) \mid i \leq h \} \). However, in order to find an assignment for which the resulting makespan \( a_h + (|\mathcal{R}'| + k - h) \cdot c \) where \( J_i \) is the first job is minimal the assignment \( f \) might have to be redefined for the subset \( \mathcal{R}' \). For this we need the following lemma.
Lemma 5.3: If there exists an assignment of some jobs in $\mathcal{R}'$ to the jobs $\{J_k, J_{k-1}, \ldots, J_{h+1}\}$ where each job in $\{J_k, J_{k-1}, \ldots, J_{h+1}\}$ gets assigned exactly one job $f(J_i) \in \mathcal{R}'$ such that $f(J_i)$ is compatible with $J_j$ then there also exists such an assignment $f$ where $f(J_i)$ is the smallest possible job. This assignment $f$ satisfies the following property: a minimal permutation schedule for the order $f(J_i), R, J_k, f(J_{k-1}), J_{k-1}, \ldots, f(J_{h+1})$, $J_{h+1}$ where $R$ is any order of the jobs $\mathcal{R}' \setminus \{f(J_i) \mid i \in \{h+1, h+2, \ldots, k\}\}$ is an optimal permutation schedule for the jobs in $\{J_k, \ldots, J_{h+1}\} \cup \mathcal{R}'$.

Proof: Assume there exists no permutation schedule $\sigma$ with an assignment $f$ such that the schedule $f(J_i), R, J_k, f(J_{k-1}), J_{k-1}, \ldots, f(J_{h+1})$, $J_{h+1}$ is optimal; that is, all optimal schedules use an assignment $f$ and an order of the jobs in $J'$ that only coincides with the $t$ rightmost elements $J_{h+t}, J_{h+t-1}, \ldots, J_{h+1}$ after which a different order $J_{t_1}, J_{t_2}, \ldots, J_{t_{n-h}}$ is used for the remaining jobs $J_k, J_{k-1}, \ldots, J_{h+1}$ in $J'$:

$$f(J_{t_1}), R, J_{t_i}, f(J_{t_{i-1}}), J_{t_{i-1}}, \ldots, f(J_{t_{n-h+1}}), J_{t_{n-h+1}}, f(J_{h+2}), J_{h+2}, f(J_{h+1}), J_{h+1}. \quad (2)$$

Let $t$ be maximal such that the resulting permutation (2) is optimal and has the highest number of jobs coinciding with the sequence $J_k, J_{k-1}, \ldots, J_{h+2}, J_{h+1}$. The makespan of the corresponding schedule is equal to $a_t + (|\mathcal{R}'| + k - h) \cdot c$ where $t$ is the index corresponding to the first job $f(J_i)$ in the schedule.

We now show that we can construct a new permutation schedule with the same makespan where the $t + 1$ rightmost elements coincide with this sequence, thereby contradicting that a maximal $t$ exists: Since $h + t < k$, the order of the jobs in $J'$ is only identical for $J_{h+t}, J_{h+t-1}, \ldots, J_{h+1}$ whereas the job $J_{h+1}$ is at a different position $i_j$ in the permutation (2): $J_{h+t+1} = J_{i_j}$. In addition, the first job $J_{i_{k-h-t}}$ at which the two permutations differ corresponds to a job $J_s \in J'$ with $s \geq h + t + 2$, that is, $i_{k-h-t} = h + t + 2$. Since the schedule is optimal it must hold that $f(J_{i_j})$ and $f(J_{i_{j+1}})$ are compatible with $J_t$. Since $J_t$ is the largest job in $J_{i_j}, J_{i_{j+1}}, \ldots, J_{i_{k-h-t}}$ it holds that $f(J_{i_j})$ and $f(J_{i_{j+1}})$ are compatible with each job in $J_{i_j}, J_{i_{j+1}}, \ldots, J_{i_{k-h-t}}$. Therefore, the minimal schedule for the order $f(J_{i_h}), R, J_{i_j}, \ldots, f(J_{i_{j+1}}), J_{i_{j+1}}, \ldots, f(J_{i_{k-h-t}}), J_{i_{k-h-t}}$, $f(J_{i_{j+1}}) J_{h+t+1}, f(J_{h+t}), J_{h+t}, \ldots, f(J_{h+2}), J_{h+2}, f(J_{h+1}), J_{h+1}$ has the same makespan $a_t + (|\mathcal{R}'| + k - h) \cdot c$ as $\sigma$ and coincides with $t + 1$ jobs from the sequence $J_k, J_{k-1}, \ldots, J_{h+2}, J_{h+1}$ contradicting the maximality of $t$.

Thus, there exists a permutation schedule with an assignment $f$ such that the sequence $f(J_k), R, J_k, f(J_{k-1}), J_{k-1}, \ldots, f(J_{h+1})$, $J_{h+1}$ is optimal.

Lemma 5.3 shows that an optimal permutation schedule can be found with a compatible predecessor assignment for the jobs in $\{J_k, \ldots, J_{h+1}\}$ such that $f(J_k)$ is minimal. This is done in lines 23–26 of Algorithm 1 where always the largest possible job from $\mathcal{R}'$ is assigned to the next job in $\{J_k+1, \ldots, J_{h+1}\}$ and the smallest remaining job is afterwards assigned to $J_k$. In addition, it can be seen that Algorithm 1 always constructs schedules $\sigma$ with the maximum number of neighbored pairs of compatible jobs. Thus, it is also optimal for the case where $c \leq a_t$ for all $i \in \{1, 2, \ldots, n\}$ in which case the length of the schedule is $\sum_{i=1}^{n} a_i + |\mathcal{R}'| \cdot c$.

Theorem 5.1: Given an instance of the type $F2|\text{prmu}, b_i = c, \text{bufType}, \text{bufUsage}|C_{\text{max}}$ with $\text{bufType} \in \{\text{spanningBuffer, intermediateBuffer}\}$ and $\text{bufUsage} \in \{s_i = 1, s_i = a_t\}$, an optimal schedule can be constructed in time $O(n \log n)$ if $c \geq \max_i a_i$ or $c \leq \min_i a_i$.

Proof: We consider all possible values for $\text{bufType}$ and $\text{bufUsage}$. It was already shown that Algorithm 1 computes an optimal schedule in $O(n \log n)$ steps for the case of a spanning buffer and $s_j = a_j$.
For the case of an intermediate buffer and $c \geq \max_i a_i$ it is possible to construct a schedule $\sigma$ as shown in Figure 5 which starts with a job $J_k \in \mathcal{J}$ satisfying $a_k \leq a_j$ for all $j \in \{1, 2, \ldots, n\}$ and where each job is immediately processed on $M_2$ after being finished on $M_1$ leaving the intermediate buffer unoccupied. This schedule $\sigma$ has a makespan of $C_{\text{max}}(\sigma) = a_k + nc$, which is a trivial lower bound for the makespan such that $\sigma$ is optimal.

Next, assume that the flow shop instance has an intermediate buffer and satisfies $c \leq \min_i a_i$. In this case every schedule $\sigma$ where all jobs are processed as early as possible has a structure similar to the schedule shown in Figure 6 where the machine $M_2$ is always idle by the time $M_1$ finishes a job $J \in \mathcal{J}$ allowing it to immediately start $J$ and leaving the intermediate buffer unoccupied. These schedules have a makespan of $C_{\text{max}}(\sigma) = \sum_j a_j + c$ which is also a trivial lower bound for the makespan so that all schedules of this type are optimal for this case.

It remains to consider the case of a spanning buffer and $s_i = 1$. Consider first that $\Omega = 1$. Then no two jobs are processed at the same time on both machines so that every schedule $\sigma$ (where every job starts as early as possible) is optimal with a makespan of $C_{\text{max}}(\sigma) = \sum_i a_i + nc$. If $\Omega \geq 2$, then it is possible to construct the same schedules as in Figures 5 and 6 (depending on whether $c \geq \max_i a_i$ or $c \leq \min_i a_i$) since a maximum of two jobs occupy the buffer at any time.

6 A Modification of the NEH Heuristic

The NEH algorithm (Nawaz et al., 1983) is a very successful and the most famous heuristic for makespan minimization in permutation flow shop problems. The algorithm starts with an empty permutation building it up by iteratively inserting jobs (in decreasing order of the sums of their processing times) into the positions which lead to the smallest increases in makespan. It can be shown that this heuristic also calculates an optimal solution for all special cases considered in Theorem 5.1 since it also performs greedy insertions maximizing the number of hidden jobs as well as the sum of their processing times.
Several variants of the NEH heuristic have been studied in the literature. Most of them deal with ties (Fernandez-Viagas and Framinan, 2015) or use different orders of job insertions, for example, based on the moments of processing times (Liu et al., 2017) or using genetic programming (Vázquez-Rodríguez and Ochoa, 2011). An advantage of the NEH heuristic is its relatively smaller run time since it checks \( \Theta(n^2) \) insertion points.

In particular, when the solution of the NEH heuristic is further used by improvement heuristics, for example, as the starting solution for an Iterated Local Search heuristic, it is desirable to have a faster heuristic for the computation of a starting solution. Here, we introduce a modification of the NEH heuristic (mNEH) that is faster and particularly suitable for the considered two-machine flow shops with buffers where all processing times on the second machine \( M_2 \) are equal. In this case, the jobs differ only by their processing time on \( M_1 \). Therefore, it is likely to occur that many jobs are similar or even identical. This property is used in the mNEH heuristic to reduce the number of positions that are checked for the insertions of the jobs and thus the total number of evaluated schedules. The main idea is to split the \( n \) jobs into \( G(n) \) groups of similar jobs and to maintain for each group a list of \( L(n) \) "good" candidate positions for the insertion. This is based on the assumption that insertions of similar jobs at the same positions lead to similar changes in the resulting makespan.

A pseudocode of the modified heuristic is given in Algorithm 2. The number of evaluations to be performed can then be adjusted by appropriate choice of \( L(n) \) and \( G(n) \). In this article, we memorize \( L(n) = 2\sqrt{n} \) positions, that is, potentially \( \sqrt{n} \) before and \( \sqrt{n} \) after the newly inserted job for each of the \( G(n) = \sqrt{n} \) groups of equal size (an exception is the last group as described later) and it follows that the resulting algorithm performs \( O(n\sqrt{n}) \) evaluations. It should be noted that in this case the incomplete permutation \( \pi \) contains less than \( L(n) \) jobs during the scheduling of the first two groups \( S_1 \) and \( S_2 \). For the jobs in these two groups, the mNEH heuristic tests all possible insertion positions in the same way as in the standard NEH heuristic. For the last group \( S_{G(n)} \), all possible positions are checked since this group contains the shortest jobs allowing for a finer optimization of the partial permutation \( \pi \). Regarding the time complexity (including the time to evaluate a permutation as well as the time to build and update the list)

\begin{algorithm}
\textbf{Algorithm 2 Modified NEH heuristic (mNEH)}

\textbf{Parameters:} number of groups \( G(n) \), number of positions \( L(n) \) to memorize

1: \( \pi^{\text{sort}} \leftarrow \) sorted sequence of jobs by descending \( a_i \)
2: \( \pi \leftarrow \) empty permutation
3: \( S_1, S_2, \ldots, S_{G(n)} \leftarrow \) partition of \( \pi^{\text{sort}} \) into \( G(n) \) groups of equal size
4: \textbf{for} \( S \in \{ S_1, S_2, \ldots, S_{G(n)-1} \} \) \textbf{do}
   5: \hspace{1em} \( (j_1, j_2, \ldots, j_k) \leftarrow \) jobs in the current group \( S \)
   6: \hspace{1em} test insertion of \( j_1 \) in all possible positions of \( \pi \) and memorize the best \( L(n) \) insertions
   7: \hspace{1em} insert \( j_1 \) at the best position in \( \pi \)
   8: \hspace{1em} update list of memorized \( L(n) \) positions
   9: \hspace{1em} \textbf{for} \( \ell \in \{2, 3, \ldots, k\} \) \textbf{do}
      10: \hspace{2em} test insertion of \( j_\ell \) into \( \hat{\pi} \) at the memorized \( L(n) \) positions
      11: \hspace{2em} test insertion of \( j_\ell \) into neighbor positions of \( j_{\ell-1} \)
      12: \hspace{2em} insert \( j_\ell \) at the best tested position in \( \pi \)
      13: \hspace{2em} update list of memorized \( L(n) \) positions, if necessary
   14: \hspace{1em} \textbf{end for}
15: \textbf{end for}
16: insert the jobs in \( S_{G(n)} \) into \( \pi \) as in the standard NEH heuristic
17: \textbf{return} \( \pi \)
\end{algorithm}
it can be shown that the mNEH heuristic with the used parameter values is faster than the standard NEH heuristic by a factor of $O(\sqrt{n})$.

7 Iterated Local Search

In this section, we propose an Iterated Local Search heuristic for the two-machine flow shop problems with buffers. The heuristic is denoted by 2BF-ILS. The local search operations that are considered for 2BF-ILS are the following:

- inserting a job on position $i$ into another position $j$ (insert)
- inserting a pair of adjacent jobs at the positions $(i, i+1)$ into the positions $(j, j+1)$ (pairInsert)
- swapping two jobs at the positions $i$ and $j$ (swap)

The naming of these operations is based on Zhang and Gu (2015) and Moslehi and Khorasanian (2014). The actual selection of these operations and their order used in the experiment is later determined by the algorithm configurator irace in Section 8.3.

As such, the following description is based on the generalized case where a sequence $op_1, op_2, \ldots, op_\ell$ is given with $op_i \in \{\text{insert}, \text{pairInsert}, \text{swap}\}$ for $i \in \{1, 2, \ldots, \ell\}$. Each of the considered operations $op \in \{\text{insert}, \text{pairInsert}, \text{swap}\}$ takes two parameters $i$ and $j$ so that we denote the result of $op$ applied on $\pi$ with these parameters as $op(\pi, i, j)$. The resulting neighborhood of permutations around $\pi$ is denoted $N_{op}(\pi) = \{\pi' | \exists i, j : op(\pi, i, j) = \pi'\}$, that is, the set of all permutations $\pi'$ that can be obtained by a single application of $op$ on $\pi$. The size of the neighborhoods for the considered operations is quadratic with respect to the problem size $n$ since the number of possible values for the parameters $i$ and $j$ linearly increases with $n$. One way to reduce the size is to fix a parameter $i$:

$$N_{i, op}(\pi) = \{\pi' | \exists j : op(\pi, i, j) = \pi'\}.$$ 

This lowers the number of checked permutations leading to potentially good solutions being missed. However, the linear size of the resulting neighborhood allows for more local search steps to be performed in the same amount of time.

The main steps of our proposed method are shown in Algorithm 3: It starts with the solution obtained from the mNEH heuristic proposed in Section 6 (with $G(n) = \sqrt{n}$ and $L(n) = 2\sqrt{n}$). Then 2BF-ILS iterates through a sequence of operations $(op_1, op_2, \ldots, op_\ell)$ and repeatedly performs local search steps with changing neighborhoods. The choice of which parameter $i$ to fix in $N_{i, op}(\pi)$ is based on a random permutation $\pi^{rand}$ calculated beforehand. The search steps with the current operation $op$ are repeated until the permutation $\pi^{cur}$ obtained so far cannot be improved further. Afterwards, the local search procedure is restarted with the next given operation.

If the best-known solution obtained so far did not improve after all operations, the next iteration of the algorithm uses a perturbed variant of the best-known solution. The perturbation we use is based on a geometric distribution: We repeatedly draw (uniformly) random numbers $r$ from $[0,1]$ and then randomly apply one of the given operations with random parameters on the permutation until $r$ is greater than $ps$. This distribution favors a small number of perturbations. The strength of perturbation $ps$ increases additively (here set as $\varepsilon = 0.05$ with $ps$ maxed out at 0.99) if successive iterations do not yield any improvement of the best-known solution, otherwise it is reset to the initial value $ps_{init}$. 

Evolutionary Computation Volume 29, Number 3
In this section, we describe the experiments and their results.

8.1 Algorithms for Comparison

Based on the literature overview given in Section 2 and Figure 1, we selected the Hybrid Variable Neighborhood Search (HVNS) and the Discrete Artificial Bee Colony (DABC) for a comparison with our proposed method since these algorithms are described in fairly recent works (Zhang and Gu, 2015; Moslehi and Khorasanian, 2014) and have not been outperformed by any other algorithm to the best of our knowledge. For the implementation, we asked the authors of both algorithms for the source code but did not receive a reply. Thus, we reimplemented both algorithms.

In addition to HVNS and DABC, we also used an Ant Colony Optimization algorithm (ACO) for comparison. The reason is that ACO generally achieves good results for many scheduling problems. It is thus a promising algorithm which so far has not been applied to flow shops with buffers. The ACO in this work is a standard ACO without heuristic components where each pheromone value $\tau_{ij}$ encodes the desirability that job $J_j$ is the successor of job $J_i$. At each iteration the pheromone values are updated by the solution that is obtained by 2BF-ILS applied to the permutation generated by the best ant in the current iteration (i.e., the iteration best solution becomes $\pi_{\text{cur}}$ in line 4 in Algorithm 3).

Each application of 2BF-ILS was limited to 1000 search steps.

8.2 Generation of Problem Instances

The commonly used benchmark instances for flow shop problems (from Taillard, 1993, Reeves, 1995 and the recent VRF benchmark by Vallada et al., 2015) work with at least
five machines and contain no buffer constraints so that they cannot be directly applied to the flow shops with buffers considered here. Authors who studied flow shop problems with buffers and two machines (e.g., Kononova and Kochetov, 2013 and Lin et al., 2009) generated instances with random processing times uniformly drawn over the set \(\{1, 2, \ldots, 100\}\). Based on the studies performed by the these authors, we generated instances for our experimental study as follows.

We chose \(n \in \{50, 100, 150\}\) as the number of jobs for small, medium and large instances and created three “incomplete instances” for each size that contain only the \(M_1\) processing times \(a_i\) drawn randomly from a uniform distribution over the set \(\{1, 2, \ldots, 100\}\). Each of these incomplete instances was then used to build three subordinate instances by choosing the constant processing times \(c\) on \(M_2\) as the 25\%, 50\% or the 75\% percentile \(q_{0.25}, q_{0.50}, q_{0.75}\) of the values \(a_i\). Each value of \(c\) was then used to create (complete) instances for the intermediate buffer (spanning buffer) by setting \(s_1 = 1\) and \(\Omega = 1\) (\(\Omega = 3\)) or \(s_1 = a_i\) and \(\Omega = q_{0.25}\) (\(\Omega = \max a_i + q_{0.25}\)). Note that the buffer capacity \(\Omega\) cannot be smaller than \(\max s_i\) for the spanning buffer model or else there exists no valid schedule. The resulting set of instances contained 108 flow shop problems.

In addition, we generated a set of instances based on studies conducted by Amar and Gupta (1986) and Watson et al. (2002) where it was argued that problem data generated from uniform distributions do not contain characteristics of problem instances commonly occurring in practical applications, namely gradients or correlations with respect to job processing times. In instances with the former property, the processing times \(a_i\) on \(M_1\) are shorter (or longer) than the processing time \(b_i\) on the second machine for all jobs \(J_i\). Regarding flow shops satisfying \(b_i = c\) for all jobs \(J_i\), job gradients lead to the cases analyzed in Section 5 which are efficiently solvable. Thus, we considered the latter characteristic by drawing the processing times \(a_i\) for \(M_1\) from normal distributions \(N(\mu, \sigma^2)\) with mean value \(\mu = c\). Higher values for \(c\) then lead to tendentially longer processing times on \(M_1\) and vice versa. In particular, we calculated \(n\) integer values for processing times \(a_i\) \((n \in \{50, 100, 150\}\) by drawing \(n\) random numbers \(r_1, r_2, \ldots, r_n\) from normal distributions \(N(\mu, \sigma^2)\) with \(\mu = c, \sigma = 10\) and \(c \in \{25, 50, 75\}\) and setting \(a_i = \max([r_i], 1)\) for all \(i \in \{1, 2, \ldots, n\}\). The values for the buffer usage \(s_i\) are calculated in the same way as for the first set of instances leading to 108 additional flow shop problems.

### 8.3 Parameter Values

For the algorithms 2BF-ILS and ACO the parameter values were determined with the algorithm configurator irace (López-Ibáñez et al., 2016). The configuration was performed on a separate set of instances with 100 jobs (generated using the method described in Subsection 8.2 for uniformly distributed processing times) with the standard irace parameter values and a budget of 250 runs for each algorithm. The limits used by irace for the parameter values were chosen as \(ps_{\text{init}} \in (0.01, 0.99), \rho \in (0.01, 0.05), N_{\text{ants}} \in \{1, 2, \ldots, 20\}\). As possible sequences for operations we considered all sequences of length 1, 2, or 3 that can be formed by the operations insert, pairInsert, or swap (without repetition leading to 15 possible sequences in total). The algorithms HVNS and DABC were applied with the parameters given by the respective authors as well as tuned parameters calculated by irace on the same instances. In the following, the tuned versions of these algorithms are referred to as HVNS-T and DABC-T.

The resulting parameter values are given in Table 1. Due to page limitations, we refer the reader to https://github.com/L-HT/TwoMachineFlowShopBuffers where the resulting parameter values for HVNS-T and DABC-T as well as the source code for each
Table 1: Parameters for ACO and 2BF-ILS calculated by irace.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameter</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACO</td>
<td>$ps_{\text{init}}$</td>
<td>0.914</td>
<td>initial perturbation strength</td>
</tr>
<tr>
<td></td>
<td>$\text{op}$</td>
<td>$(\text{swap, insert, pairInsert})$</td>
<td>sequence of operations</td>
</tr>
<tr>
<td></td>
<td>$N_{\text{ants}}$</td>
<td>19</td>
<td>number of ants</td>
</tr>
<tr>
<td></td>
<td>$\rho$</td>
<td>0.018</td>
<td>pheromone evaporation</td>
</tr>
<tr>
<td>2BF-ILS</td>
<td>$ps_{\text{init}}$</td>
<td>0.240</td>
<td>initial perturbation strength</td>
</tr>
<tr>
<td></td>
<td>$\text{op}$</td>
<td>$(\text{pairInsert, insert, swap})$</td>
<td>sequence of operations</td>
</tr>
</tbody>
</table>

algorithm and the following evaluations (written in R and C++) are available. After the configuration, each algorithm was executed on all 216 instances and the resulting values were averaged over 10 replications. All test runs were performed on a computer with eight 3.4-GHz-cores (each run being executed on one core) and 32 GB RAM with time limits of 5, 10, and 15 minutes for the small, medium, and large problems, respectively.

8.4 Comparison of 2BF-ILS with Other Metaheuristics

The performance of each algorithm was evaluated based on (i) the number of performed function evaluations $FE$ to calculate the makespan and (ii) the elapsed absolute runtime. The latter was used to calculate the normalized runtime $NT$ as a time measure where the reference runtime for a given instance was chosen as the mean runtime of 10 runs of the standard NEH heuristic.

For each instance and each time measure, we generated (similar to the study performed by Weise et al., 2014) the progress curve (PC, the quality of the best-known solution over time), the estimated running time (ERT, indicating the time necessary to reach a given solution quality), and the empirical cumulative distribution function (ECDF, describing the percentage of runs reaching a given solution quality over time) with respect to the relative percentage difference $\text{RPD} = (F(\pi) - \hat{F})/\hat{F}$ between the solution quality $F(\pi)$ of the permutation $\pi$ and the best solution quality $\hat{F}$ found in all runs on this instance. The target value of the ECDF was chosen as a 1%-deviation of $\hat{F}$ such that these diagrams show how consistent an algorithm reaches the performance $\text{RPD} \leq 0.01$.

Since an individual evaluation of over 1300 diagrams is not feasible, we calculated the area under curve (AUC) as an aggregate quality measure for the PC, ERT, and ECDF diagrams (with respect to both time measures) since algorithms with low AUC values (for PC and ERT diagrams) and high AUC values (for ECDF diagrams) tend to find better solutions faster (Weise et al., 2014). As such, these values quantify the performance of an algorithm for a given instance over time. The relative AUC values, averaged over all instances used in this study, are shown in Table 2 for both sets of instances.

It can be seen that 2BF-ILS outperformed the other algorithms in all evaluation measures on both sets of flow shop problems. ACO showed a competitive performance when compared to HVNS and DABC with respect to the values for the PC diagrams which indicates that it quickly improves at the beginning when the first solutions are created. The irace-tuned parameters for HVNS obtained better results than the untuned parameters for uniformly random processing times on $M_1$ with respect to nearly all evaluation measures whereas the results for DABC-T in comparison to DABC are mixed. It can also be seen for HVNS, HVNS-T, and ACO that the values in Table 2 for the instances with normally distributed processing times are smaller than for uniformly random instances for almost all evaluation criteria which indicates that these methods...
Table 2: Relative area under curve (AUC) values for each algorithm, evaluation diagram, and time measure (indicated by the subscript $NT$ or $FE$) with respect to both sets of flow shop instances.

<table>
<thead>
<tr>
<th></th>
<th>HVNS</th>
<th>HVNS-T</th>
<th>DABC</th>
<th>DABC-T</th>
<th>ACO</th>
<th>2BF-ILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform $PC_{FE}$</td>
<td>2.22</td>
<td>2.13</td>
<td>1.81</td>
<td>1.66</td>
<td>2.01</td>
<td>1.03</td>
</tr>
<tr>
<td>$PC_{NT}$</td>
<td>2.41</td>
<td>2.40</td>
<td>1.54</td>
<td>1.71</td>
<td>2.06</td>
<td>1.04</td>
</tr>
<tr>
<td>$ERT_{FE}$</td>
<td>2.62</td>
<td>2.59</td>
<td>3.07</td>
<td>3.02</td>
<td>2.58</td>
<td>1.11</td>
</tr>
<tr>
<td>$ERT_{NT}$</td>
<td>2.49</td>
<td>2.48</td>
<td>1.91</td>
<td>2.36</td>
<td>2.60</td>
<td>1.25</td>
</tr>
<tr>
<td>$ECDF_{FE}$</td>
<td>1.28</td>
<td>1.27</td>
<td>1.23</td>
<td>1.23</td>
<td>1.44</td>
<td>1.00</td>
</tr>
<tr>
<td>$ECDF_{NT}$</td>
<td>1.35</td>
<td>1.35</td>
<td>1.21</td>
<td>1.22</td>
<td>1.50</td>
<td>1.00</td>
</tr>
<tr>
<td>normal $PC_{FE}$</td>
<td>1.90</td>
<td>1.93</td>
<td>1.95</td>
<td>1.98</td>
<td>1.89</td>
<td>1.04</td>
</tr>
<tr>
<td>$PC_{NT}$</td>
<td>2.00</td>
<td>2.03</td>
<td>1.61</td>
<td>1.89</td>
<td>1.94</td>
<td>1.03</td>
</tr>
<tr>
<td>$ERT_{FE}$</td>
<td>2.47</td>
<td>2.58</td>
<td>3.16</td>
<td>3.29</td>
<td>2.16</td>
<td>1.11</td>
</tr>
<tr>
<td>$ERT_{NT}$</td>
<td>2.40</td>
<td>2.54</td>
<td>1.89</td>
<td>2.36</td>
<td>2.13</td>
<td>1.26</td>
</tr>
<tr>
<td>$ECDF_{FE}$</td>
<td>1.05</td>
<td>1.07</td>
<td>1.08</td>
<td>1.12</td>
<td>1.24</td>
<td>1.00</td>
</tr>
<tr>
<td>$ECDF_{NT}$</td>
<td>1.08</td>
<td>1.09</td>
<td>1.05</td>
<td>1.11</td>
<td>1.25</td>
<td>1.00</td>
</tr>
</tbody>
</table>

For each instance, the area under the curve was calculated for each diagram and the best obtained value was chosen as the “reference” to calculate the relative values for the other algorithms. The obtained values were averaged over all 108 instances. A low number indicates that the curves for the respective algorithm show a high similarity (on average) to the best performing algorithm on each instance. The lowest values are shown in bold for each criterion.

Figure 7: Progress curve for an instance using the intermediate buffer, $s_i = 1$, $c = q_{0.50}$, uniformly random processing times and 50 jobs (left) and an instance with spanning buffer, $s_i = a_i$, $c = q_{0.50}$, normally distributed processing times and 100 jobs (right).

Evolutionary Computation Volume 29, Number 3
Table 3: Results of the pairwise comparisons between the algorithms using the two-sided sign test for both sets of flow shop problems (each containing $n = 108$ instances).

<table>
<thead>
<tr>
<th></th>
<th>HVNS</th>
<th>HVNS-T</th>
<th>DABC</th>
<th>DABC-T</th>
<th>ACO</th>
<th>2BF-ILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HVNS</td>
<td>-/-</td>
<td>▲▲ ▲</td>
<td></td>
<td>▲</td>
<td>-/-</td>
<td>▲▲ ▲</td>
</tr>
<tr>
<td>HVNS-T</td>
<td>-/-</td>
<td>-/-</td>
<td>▲</td>
<td></td>
<td>-/-</td>
<td>▲</td>
</tr>
<tr>
<td>DABC</td>
<td>▲/▲ ▲</td>
<td>-/-</td>
<td>-/-</td>
<td>▲</td>
<td>-/-</td>
<td>▲</td>
</tr>
<tr>
<td>DABC-T</td>
<td>▲/▲ ▲</td>
<td>▲/▲ ▲</td>
<td>-/-</td>
<td>▲</td>
<td>-/-</td>
<td>▲</td>
</tr>
<tr>
<td>ACO</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>▲</td>
<td>-/-</td>
<td>▲</td>
</tr>
<tr>
<td>2BF-ILS</td>
<td>▲/▲ ▲</td>
<td>▲/▲ ▲</td>
<td>▲/▲ ▲</td>
<td>▲</td>
<td>▲/▲ ▲</td>
<td></td>
</tr>
<tr>
<td>normal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HVNS</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>▲</td>
</tr>
<tr>
<td>HVNS-T</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>▲</td>
</tr>
<tr>
<td>DABC</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>-/▲ ▲</td>
<td>-/-</td>
<td>▲</td>
</tr>
<tr>
<td>DABC-T</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>-/▲ ▲</td>
<td>-/-</td>
<td>▲</td>
</tr>
<tr>
<td>ACO</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
<td>▲</td>
</tr>
<tr>
<td>2BF-ILS</td>
<td>▲/▲ ▲</td>
<td>▲/▲ ▲</td>
<td>▲/▲ ▲</td>
<td>▲/▲ ▲</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first value in each cell shows the test result with respect to the performance at 100,000 evaluations and the second value refers to the performance reached at the end of the time limit. A triangle indicates that the measured difference is statistically significant ($p < 0.05/15$ due to Bonferroni correction) and that the algorithm at which the triangle is pointed at is significantly better according to the test statistic.

an optimal solution was reached. We performed further experiments using additional instances of this type with similar results which indicates that this special case could be “easier” to solve than other problems (even though in theory it is still NP-complete).

To compare the performance of the algorithms at specific points in time the sign test for paired samples was applied. This non-parametric test neither requires the given data to be normally distributed nor the difference distributions between the methods to be symmetric. Using this test, we compared the performance of the algorithms during the run (at 100,000 evaluations) and the performance that was reached at the end of the time limit. The results are shown in Table 3 for both sets of instances. Note that the tables are symmetric since the 15 possible pairwise comparisons for each of the two points in time were performed with two-sided tests. Similar to Table 2, it can be seen that 2BF-ILS obtained a significantly better performance than the other algorithms at both points indicating a high and consistent performance over time. The only difference is that in the case for normally distributed processing times the tests showed fewer statistically significant differences (i.e., values $p < 0.05/15$) between the competitor algorithms than for instances with uniformly random processing times.

Plotting the RPD values of the final solutions on each instance in relation to 2BF-ILS as points ($RPD_{2BF-ILS}$) due to the paired nature of the measured data (see Figure 8) also shows the high performance of 2BF-ILS. It can be seen that for larger problem sizes the gap in solution quality to the other algorithms increases. In particular, the average RPD ratio between 2BF-ILS and the other algorithms is significantly smaller for instances with 50 jobs than for instances with 150 jobs ($p = 0.02$ for uniformly distributed processing times and $p = 0.003$ for normally distributed processing times according to the Wilcoxon rank-sum test). In addition, it can be seen that there are instances where ACO obtained worse results than the other algorithms—most often on the larger instances with normally distributed processing times. One reason for this is that there exist many similar or equal jobs in the studied types of flow shops leading to problems for the ants. In addition, Figure 8 (right) shows that the majority of points are gathered...
around lower RPD values for 2BF-ILS, HVNS(-T), and DABC(-T) indicating that these algorithms tend to obtain solutions with higher and more similar quality for instances with correlations between the processing times $a_i$ and $c_i$ than for problems with uniformly random processing times. This observation as well as the differences between the two sets of instances noted previously for Table 2 support the results from Amar and Gupta (1986) and Watson et al. (2002) stating that instances containing structural features are easier to solve than arbitrarily random problems.

8.5 Comparison of 2BF-ILS with NEH

In this section, we compare the standard NEH with 2BF-ILS which uses mNEH with the parameter values $G(n) = \sqrt{n}$ and $L(n) = 2\sqrt{n}$ for constructing its starting solution. In particular, the time was measured for 2BF-ILS to obtain the same solution quality $C_{NEH}$ as the standard NEH. This experiment was performed using the same instances described in Subsection 8.2. For an example instance, Figure 9 left shows the solution quality of the standard NEH heuristic as well as the performance over time for 2BF-ILS (averaged over 10 replications). It can be seen that mNEH needed approximately half as many evaluations as NEH to finish (though initially obtaining a slightly worse solution quality). The reduced number of evaluations then allowed 2BF-ILS to obtain the quality $C_{NEH}$ with fewer evaluations than NEH.

Visualizing the number of evaluations for both algorithms (see Figure 9 right) shows that there is a small number of outliers (at most 4 out of 36 instances per problem size and processing time distribution) where mNEH takes a noticeably high number of evaluations before reaching $C_{NEH}$. However, the median values for 2BF-ILS were still smaller than the number of evaluations performed by the standard NEH heuristic for all problem sizes and both sets of instances. In addition, the sign test for flow shops with uniformly distributed processing times ($n = 108$) showed that 2BF-ILS needed
Figure 9: Left: Comparison between the standard NEH and 2BF-ILS for an instance with 150 jobs, spanning buffer, $s_i = 1$, $c = q_{0.50}$ and uniformly distributed processing times on $M_1$. Dashed lines indicate when the standard NEH finishes (vertical lines) and the attained solution quality $C_{NEH}$ (horizontal lines). A cross marks the quality of the initial mNEH solution and a circle marks the number of evaluations at which a better quality than NEH was reached for the first time. Right: Box plot for 2BF-ILS showing the number of evaluations (averaged over 10 replications on each instance) until $C_{NEH}$ was reached for the considered problem sizes and both sets of instances. A red line indicates the number of evaluations at which the standard NEH heuristic finished.

significantly fewer evaluations to reach $C_{NEH}$ than NEH ($p < 0.001$). The differences were also significant for the instances with normally distributed processing times on $M_1$ ($p = 0.013$, $n = 108$).

9 Conclusion

Algorithms and complexity results for two-machine permutation flow shops with buffers were presented in this article for the specific case where all processing times on one of the machines are equal to a constant $c$. For such problems, two types of buffers (intermediate buffers and spanning buffers) and two types of buffer usage (all jobs occupy the same amount of buffer and the amount of buffer usage for each job equals its processing time on the machine with non-constant times) were considered. It was shown that all four resulting cases of buffered two-machine permutation flow shops are $NP$-hard under the makespan criterion. A modified version of the NEH heuristic (mNEH) was proposed that utilizes characteristics of the considered flow shop problems. Also, an Iterated Local Search heuristic (2BF-ILS) was proposed that uses mNEH for computing its starting solution. It was experimentally shown that 2BF-ILS obtains better schedules over time outperforming a standard Ant Colony Optimization algorithm as well as two state-of-the-art algorithms for flow shop problems with buffers from the literature. The results show that the compared algorithms perform better if the flow shop instances contain correlations between processing times, a structural feature that commonly occurs in practical applications. In addition, the experimental results show that 2BF-ILS obtains the same solution quality as the standard NEH heuristic but uses a smaller number of function evaluations.
For all four types of buffered two-machine permutation flow shops two polynomial-time solvable subcases were identified, namely, the cases where the constant $c$ is either larger or smaller than all processing times on the other machine. The most interesting of these subcases is the spanning buffer when $c$ is larger than all processing times on the other machine and for each job the amount of buffer usage equals its run time on the non-constant time machine. For this case we presented an algorithm 2BF-OPT which calculates optimal schedules in $O(n \log n)$ steps.

Acknowledgments

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), project number 392050753.

References


