Minimizing makespan in a pallet-constrained flowshop

Citation for published version (APA):

Document status and date:
Published: 01/01/1998

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author’s version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
https://www.tue.nl/index.php?id=71870

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Memorandum COSOR 98-20

Minimizing makespan in a pallet-constrained flowshop

M. Y. Wang
S. P. Sethi
C. Srisankarajah
S.L. van de Velde
J.A. Hoogeveen

Eindhoven, September 1998
The Netherlands
Minimizing Makespan in a Pallet-Constrained Flowshop

Michael Y. Wang
Faculty of Management
University of Toronto
Toronto, Canada

Suresh P. Sethi
School of Management
University of Texas at Dallas
Dallas, U.S.A.

Chelliah Sriskandarajah
School of Management
University of Texas at Dallas
Dallas, U.S.A.

Steef van de Velde
Rotterdam School of Management
Erasmus University
Rotterdam, The Netherlands

Han Hoogeveen
Department of Mathematics and Computing Science
Eindhoven University of Technology
Eindhoven, The Netherlands

3 September 1998

1This research is supported in part by NSERC A4619, OGP0104900, Manufacturing Research Corporation of Ontario, and INTAS-93-257-Ext.
ABSTRACT

We consider the problem of scheduling $n$ jobs in a pallet-constrained flowshop so as to minimize the makespan. In such a flowshop environment, each job needs a pallet the entire time, from the start of its first operation until the completion of the last operation, and the number of pallets in the shop at any given time is limited by a positive integer $K \leq n$.

We focus primarily on the two-machine flowshop and specifically on the impact of the number of pallets on the makespan. While it is an NP-hard problem to find the minimum number of pallets subject to an upper bound on the makespan, we prove a worst-case bound on the minimum $K$ that guarantees the least possible makespan. Furthermore, we investigate the empirical performance of Johnson's algorithm, which solves the problem to optimality if $K \geq n$, and Gilmore-Gomory's algorithm, which solves the problem to optimality if $K = 2$, when they are both straightforwardly adapted to deal with the situation where there are only $K$ pallets available at any time, with $2 < K < n$. Our computational experiments with randomly generated instances reveal that for Johnson's algorithm, which produces the least possible makespan, the required number of pallets to do so grows quite rapidly with the number of jobs. In contrast, Gilmore-Gomory's algorithm, which may fail to produce the least possible makespan, requires consistently only about four pallets to produce schedules with a makespan very close to optimal.
1 Introduction

Most scheduling research concerns the scheduling of jobs that require only one resource, usually machines, at a time. We consider the situation in which there is an additional scarce resource of which each job requires one unit throughout its processing. Specifically, we consider the problem of scheduling \( n \) jobs so as to minimize the makespan, i.e., the length of the schedule, in a flowshop environment where each job needs a pallet the entire time, from the start of its first operation until the completion of the last operation. The number of pallets in the shop at any given time is limited by a positive integer \( K \leq n \).

Manufacturing systems with pallet constraints are quite common, especially in automated or flexible manufacturing environments in which pallets and fixtures are essential components. Pallets are interfaces between machines and work pieces or parts. Work pieces are fixed on the pallets when they enter the system, after which the pallets are transported by some type of material handling system to machine centers where the work pieces remain mounted on to the pallets while being processed. Usually, pallets wait in a storage area or pallet pool located near the machine and are loaded into the machine by means of an automatic pallet changer (APC). There are two basic arrangements in which pallets are stored and loaded. In the first case, pallets from the pool are loaded into the machine in an arbitrary order; this is often done by a rotary indexing table. In the other case, pallets wait in a linear row and are fed into the machine in the same order as they are waiting (Viswanadham and Narahari, 1992). Figure 1 illustrates a system with two machines and \( K \) pallets.

There are several other manufacturing processes where scheduling problems occur that can modeled as pallet-constrained flowshop problems. These include sand casting (in iron foundries), where the flasks, the boxes that contain the molding sand, the cores, and the sand molds, can be seen as pallets; automated electroplating, where the fixtures for clamping the printed circuit boards are the pallets; and vacuum casting (in polymer shaping), where the moulds to produce the castings can be seen as pallets.

Our model is of course quite simplified, also since our primary focus lies on the problem of minimizing makespan, which is mainly appropriate in a capacity-driven environment. Nonetheless, we believe that our model, if not always of direct use, is at least a natural first step and a stepping stone for
properly tackling practical problems like these.

Pallet-constrained manufacturing systems are often modeled as closed queuing networks. These models look upon pallets as customers that circulate among the machine centers, neither entering nor departing from the network. The arriving job joins a queue when all pallets are occupied, or is picked up if there is a pallet available. Some references on this subject can be found in the book by Viswanadham and Narahari (1992, Chapter 4) and in the surveys by Buzacott and Yao (1986A, 1986B). The scope of this type of queuing network approach is limited, however. For one thing, the number of pallets is almost always assumed to be given. In reality, this number has to be determined in order to configure the system before it can be scheduled. After all, pallets and fixtures are expensive equipments, and a solid trade-off has therefore to be made between the number of pallets and the performance of the system. Especially interesting and worth investigating is the marginal gain of an additional pallet in terms of a specified criterion.

Little is also understood about the combinatorial nature of the problem. Pallets are renewable discrete resources and there is a bulk of literature on deterministic machine scheduling problems with scarce renewable discrete resources. The few available polynomial algorithms mainly concern single-operation models, like single and parallel machine problems. As could be expected, even simple models are already intractable; see for instance Blazewicz

Figure 1: Illustration of a manufacturing cell with pallets.
et al. (1983, 1993, Chapter 7). Most of the literature in this area mainly concerns parallel machine and jobshop problems with tools and operators; such resources are needed only to perform the operations. In contrast, pallets are resources that are needed the entire time, from the start of the first operation till the completion of the last operation.

The organization of this paper is as follows. In Section 2, we introduce our notation, define the problem under consideration, review and present some basic insights and results, and summarize the complexity mapping presented in Wang et al. (1997). A major insight is that the pallet-constrained flowshop problem is closely related to the buffer-constrained flowshop problem. In fact, we prove that a feasible schedule for the latter is a feasible schedule for the former, but not vice versa, except for the two-machine case: the two-machine flowshop with $K \geq 2$ pallets is equivalent to a two-machine flowshop with a buffer of size $K - 2$. Hence, if $K = 2$, then the problem can be solved to optimality in $O(n \log n)$ time by the well-known Gilmore-Gomory algorithm (Gilmore and Gomory, 1964). Also note that if $K \geq n$, then the availability of the pallets is never restrictive; in this case, the problem can hence be solved to optimality by scheduling the jobs by use of Johnson's algorithm (Johnson, 1954). However, if $m = 2$ and $K \geq 3$, or if $m \geq 3$ and $K \geq 2$, then there is very little hope that the problem can be solved to optimality in polynomial time—Wang et al. (1997) prove that in both cases the problem is NP-hard in the strong sense.

In the remainder of the paper, we focus on the two-machine problem with $K$-pallets ($2 < K < n$). In Section 3, we consider the problem of minimizing $K$ such that the least possible makespan, i.e., the minimum makespan if $K \geq n$, can be achieved. Although this problem is also NP-hard in general, we are able to derive the minimal $K$ that guarantees the least possible makespan for a class of job instances. In Section 4, we report on our computational experiments with randomly generated instances to investigate the empirical performance of the Gilmore-Gomory algorithm and the Johnson algorithm as approximation algorithms for the pallet-constrained two-machine flowshop problem. It appears that the Johnson algorithm produces a schedule with overall minimum makespan but requires quite a larger number of pallets. The Gilmore-Gomory algorithm, which may fail to produce a schedule with minimum makespan, performs surprisingly well: it requires usually only about four pallets, and never more than six pallets, to produce a schedule with a makespan very close to optimal.
These findings also shed some light on the empirical relationship between the makespan and the number of pallets. Although it appears that in the (theoretical) worst case, the minimal $K$ to ensure the least makespan increases rather rapidly with $n$, the number of pallets required in the average case to ensure a makespan that is very close to minimum is quite small and seems to be independent from $n$: our computational results suggest that no more than six pallets are actually needed for randomly generated instances, even if $n = 500$.

Finally, in Section 5, we summarize our results and draw some conclusions.

2 Problem definition and basic results

In this section we define our pallet-constrained flowshop scheduling problem, introduce the notation, give an overview of the basic results, and summarize a complexity mapping of the problem.

2.1 Problem description and notation

The pallet-constrained flowshop problem is defined as follows. There are $m$ machines $M_1, \ldots, M_m$ available for processing $n$ independent jobs $J_1, \ldots, J_n$, and each machine can handle no more than one job at a time and is continuously available from time zero onwards. Each job consists of a chain of $m$ operations $(O_{1j}, \ldots, O_{jm})$, which implies that the execution of $O_{jk}$ must precede the execution of $O_{j,k+1}$, for $j = 1, \ldots, n$, $k = 1, \ldots, m - 1$. Furthermore, each operation $O_{jk}$ must be performed on machine $M_k$ ($j = 1, \ldots, n$, $k = 1, \ldots, m$) during an uninterrupted processing time $p_{jk} \geq 0$. In the two-machine case, we use $a_j = p_{j1}$ and $b_j = p_{j2}$ and we refer to the operations on $M_1$ and $M_2$ as a-operations and b-operations, respectively.

Moreover, a job can only be scheduled on the first machine when a pallet is available, and once the job is started, the pallet will go with it until the last operation is completed. The pallet then becomes available for another job. At any point in time, there are no more than $K$ pallets in the shop available. Depending on the way the pallets are stored and transported, a pallet may or may not pass another one during the time the part fixed on it is processed. In the case of no-passing, the pallet has to go through each
Machine $M_k$, $k = 1, 2, \ldots, m$  
the k-th machine

$K$  
number of pallets

$J_j$, $j = 1, \ldots, n$  
job $j$

$J = \{J_1, J_2, \ldots, J_n\}$  
the set of jobs to be processed.

$O_{jk}$  
the k-th operation of $J_j$, also referred to as the $k$-operation of $J_j$, $k = 1, 2, \ldots, m$. It is performed on the k-th machine and cannot be interrupted

$p_{jk} \geq 0$  
the processing time of $O_{jk}$

$S_{jk}(\sigma), C_{jk}(\sigma)$  
the start and finish time of $O_{jk}$ given schedule $\sigma$

We often write simply $S_{jk}$ and $C_{jk}$ when there is no confusion possible. Note that $C_{jk} = S_{jk} + p_{jk}$.

| Machine $M_k$, $k = 1, 2, \ldots, m$ | the k-th machine |
| $K$ | number of pallets |
| $J_j$, $j = 1, \ldots, n$ | job $j$ |
| $J = \{J_1, J_2, \ldots, J_n\}$ | the set of jobs to be processed. |
| $O_{jk}$ | the k-th operation of $J_j$, also referred to as the $k$-operation of $J_j$, $k = 1, 2, \ldots, m$. It is performed on the k-th machine and cannot be interrupted |
| $p_{jk} \geq 0$ | the processing time of $O_{jk}$ |
| $S_{jk}(\sigma), C_{jk}(\sigma)$ | the start and finish time of $O_{jk}$ given schedule $\sigma$ |

Table 1: Overview of basic notation.

Machine even if the job it carries has a zero processing time on it.

A schedule specifies a set of completion times $C_{jk}$, $\forall j, k$, such that the above conditions, including that no more than $K$ pallets are used at any point in time, are met. The length of a schedule, denoted by $C_{\text{max}}$, referred to as the makespan, is determined as $C_{\text{max}} = \max_{1 \leq j \leq n} C_{jm}$. We address the problem of finding a schedule with minimum makespan.

Table 1 summarizes our notation.

### 2.2 Complexity

We begin by showing that there is a strong connection between the pallet-constrained flowshop problem and the buffer-constrained flowshop problem. We refer to the p-buffer flowshop problem as the problem of minimizing makespan in a flowshop with buffers of total size $p = \sum_{k=1}^{m-1} p_k$, where $p_k$ is the size of the buffer between $M_k$ and $M_{k+1}$. We prove that the $K$-pallet two-machine flowshop problem is equivalent to the $(K - 2)$-buffer two-machine flowshop problem. Moreover, a feasible solution to the p-buffer $m$-machine problem is also a feasible solution to the $p + m$-pallet $m$-machine problem, but not vice versa.

Strictly speaking, the two-pallet problem cannot be equivalent to the zero-buffer problem, because passing is allowed for the former but is impossible for the latter. The following lemma characterizes a class of optimal solutions.
Lemma 1 For the two-pallet two-machine flowshop problem, it suffices to consider only no-passing schedules.

Proof. Consider a feasible schedule $\sigma$. Suppose that $J_j$ is processed before $J_i$ on $M_1$ but after $J_i$ on $M_2$. Since there are only two pallets, no other job can be processed during the time $J_i$ is being processed, neither by $M_1$, nor by $M_2$. We take out $J_i$ from its current processing interval and reschedule it such that it is processed first on both machines. This will reduce the makespan if the $a$-operation of the job that is currently scheduled first in $\sigma$ has a positive processing time and preserve the makespan otherwise. \qed

An instance of the $m$-machine $p$-buffer problem is represented by a three-tuple $(J, m, p)$, and an instance of the $m$-machine $K$-pallet problem by $(J, m, K)$. Let $S_B(J, m, p)$ be the set of feasible solutions of $(J, m, p)$ and $S_P(J, m, K)$ be the set of feasible solutions of $(J, m, K)$. Moreover, we limit $S_P(J, 2, 2)$ to no-passing schedules only.

Theorem 1 It holds that

(a) $S_B(J, 2, p) = S_P(J, 2, p + 2)$;
(b) $S_B(J, m, p) \subseteq S_P(J, m, p + m)$;
(c) $S_B(J, m, p) \neq S_P(J, m, p + m)$ for $m \geq 3$.

Proof. Consider an instance of the $p$-buffer problem, $(J, m, p)$, and its corresponding instance of the $K$-pallet problem, $(J, m, K)$. Let us restrict the pallet problem such that there may be no more than $K_k$ pallets between $M_k$ and $M_{k+1}$ at any time. This implies that there can be at most $K_k + 2$ jobs between $M_k$ and $M_{k+1}$, including two being processed, for the restricted version of the $K$-pallet problem. Note that in the buffer problem, a job being processed on $M_k$ must wait at $M_k$ when it is finished if there are already $p_k$ jobs waiting for $M_{k+1}$. Now we can see that if $K = p + m$, then a schedule is feasible to the restricted version of the $K$-pallet problem if it is feasible to the $p$-buffer problem. Thus we have $S_B(J, m, p) \subseteq S_P(J, m, p + m)$ for $m \geq 2$.

If $m = 2$, then there is only one buffer of size $p$, and the restriction is automatically satisfied. In view of Lemma 1, we need to consider only no-passing schedules if $K = 2$, and hence we claim that a schedule is feasible
to the \((p + 2)\)-pallet problem if and only if it is feasible to the \(p\)-buffer problem, i.e., \(S_P(\mathcal{J}, 2, K) = S_B(\mathcal{J}, 2, K - 2)\). To see that \(S_P(\mathcal{J}, m, K) \subsetneq S_B(\mathcal{J}, m, K - m)\) for \(m \geq 3\), we only need to point out that while there can be at most \(K_k\) jobs in the buffer between \(M_k\) and \(M_{k+1}\) in the \(p\)-buffer problem, there can be up to \(K - 2\) pallets in the buffer between \(M_k\) and \(M_{k+1}\) in the \(K\)-pallet problem. 

Furthermore, noting that the Gilmore-Gomory algorithm solves the no-wait two-machine flowshop problem, which is equivalent to a zero-buffer two-machine flowshop problem, in \(O(n \log n)\) time (Gilmore and Gomory, 1964), we immediately have the following result.

**Corollary 1** The two-pallet two-machine flowshop problem is solvable in polynomial time by the Gilmore-Gomory algorithm.

Clearly, the pallet-constrained two-machine flowshop problem is also solvable in polynomial time if \(K \geq n\); in this case, the availability of the pallets is never restrictive, and hence Johnson's rule (Johnson, 1954) can be used to schedule the jobs optimally. We shall refer to the resulting schedule as the Johnson schedule and its makespan as the Johnson makespan.

Table 2 gives the complexity of all problem classes, obtained by Wang et al. (1997). Clearly, the problem is trivial for one machine or one pallet, in which case the problem is solvable in \(O(mn)\) time. It is also polynomially solvable if \(K = 2\) and \(m = 2\), and it is NP-hard in the strong sense if \(K \geq 3\) and \(m \geq 2\) or if \(K \geq 2\) and \(m \geq 3\). When the constraint of no-passing is imposed, the problem with \(m \geq 3\) and \(K = 2\) is NP-hard; whether it is NP-hard in the strong sense or in the ordinary sense remains open.

### 3 The two-machine problem

In this section, we limit ourselves to the two-machine flowshop. Lemma 1 shows that in case there are only two pallets, then we can restrict ourselves to no-passing schedules. If \(K \geq 3\), then there exist instances for which passing is advantageous. Nonetheless, we restrict our attention to no-passing schedules. We analyze the connection between the processing times of any
Table 2: Complexity of the \( k \)-pallet \( m \)-machine flowshop problem. NPHS stands for NP-hard in the strong sense—and NPH stands for NP-hard.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( m = 1 )</th>
<th>( m = 2 )</th>
<th>( m = 3 )</th>
<th>( m \geq 4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k = 1 )</td>
<td>( O(n) )</td>
<td>( O(n) )</td>
<td>( O(n) )</td>
<td>( O(mn) )</td>
</tr>
<tr>
<td>( k = 2 )</td>
<td>( O(n) )</td>
<td>( O(n \log n) )</td>
<td>NPHS</td>
<td>NPHS</td>
</tr>
<tr>
<td>(no-passing)</td>
<td>( O(n) )</td>
<td>( O(n \log n) )</td>
<td>NPH</td>
<td>NPH</td>
</tr>
<tr>
<td>( k = 3 )</td>
<td>( O(n) )</td>
<td>NPHS</td>
<td>NPHS</td>
<td>NPHS</td>
</tr>
<tr>
<td>( k \geq 4 )</td>
<td>( O(n) )</td>
<td>NPHS</td>
<td>NPHS</td>
<td>NPHS</td>
</tr>
<tr>
<td>( k \geq n )</td>
<td>( O(n) )</td>
<td>( O(n \log n) )</td>
<td>NPHS</td>
<td>NPHS</td>
</tr>
</tbody>
</table>

given job set and the minimum number of pallets required to achieve the Johnson makespan for that job set. Of course, if the number of pallets is fixed and restrictive, then the shop may suffer from a longer makespan. Clearly, the minimum makespan of the \( k \)-pallet two-machine flowshop is a nonincreasing function of \( k \), with Johnson’s makespan being its attainable lower bound.

Let now \( C_{k}^{\text{max}} \) be the makespan of a two-machine flowshop with \( k \) pallets, and let \( C_{\infty}^{\text{max}} \) denote Johnson’s makespan. Papadimitriou and Kanellakis (1980) prove that (for no-passing schedules), for any given job instance with \( k \geq 2 \), it holds that

\[
\frac{C_{\text{max}}^{2}}{C_{\text{max}}^{K}} \leq \frac{2K - 3}{K - 1}.
\]

They also show that the bound is tight. In other words, using \( K \) pallets rather than two can save up to \((K - 2)/(2K - 3)\) of the time needed to process any job instance. However, this does not imply that each additional pallet will lead to time saving. In fact, for any given job instance, there exists a minimal number of pallets that is sufficient to achieve Johnson’s makespan. It is therefore of considerable practical interest to determine this number. One way to pose the problem is the following:

\[
\min\{ K : C_{\text{max}}^{K} = C_{\text{max}}^{\infty} \text{ for a given job instance.} \}
\]

But this problem is NP-hard\(^2\), diminishing any hope for a polynomial algorithm to obtain the exact solution. Furthermore, it is clear that the minimal

\(^2\)To see this, note that the following two decision problems are equivalent:
The Gantt Chart of Johnson's schedule, which requires 11 pallets

The Gantt Chart of an alternative optimal schedule, which requires only 5 pallets

Figure 2: An instance that needs at least $K = 5$ pallets to achieve Johnson's makespan.

$K$ is instance dependent, ranging from 2 to $n$; this makes the solution, even if one were available, less interesting in practice: The flowshop should be configured for a class of instances, not just for a particular instance.

For this purpose, let $D$ be any class of job instances. We are interested in finding the minimal number $K_D$ that guarantees $C_{\text{max}}^K = C_{\text{max}}^\infty$ for any instance $I \in D$, i.e., we want to obtain $K_D$ such that

$$K_D = \max_{I \in D} \min\{K : C_{\text{max}}^K(I) = C_{\text{max}}^\infty(I)\}.$$ 

Clearly, $K_D$ is affected by such parameters as the maximum and minimum processing times of all the operations of the jobs in $D$. To get some idea of how $K_D$ changes with such parameters, let us examine the example depicted in Figure 2. In this instance, all the 13 $a$-operations have processing time $L > 0$, three $b$-operations have processing time $4L$, and the other ten $b$-operations have processing time $L/4$. Johnson's makespan is $15.5L$.

QA. Given a job set $J$, an integer $K$, and an integer $C$, is there a schedule with $K$ pallets and makespan $C_{\text{max}}$ such that $C_{\text{max}} \leq C$?

QB. Given a job set $J$, an integer $K$, and an integer $C$, is there a schedule with no more than $K$ pallets and makespan $C_{\text{max}}$ such that $C_{\text{max}} \leq C$?
Although Johnson’s schedule needs 11 pallets (see Figure 2(a)), Johnson’s makespan can be achieved using only 5 pallets (see Figure 2(b)). Note that by delaying the start times of the last two jobs on $M_1$ in Figure 2(a) by $2L$ units, we can achieve Johnson’s makespan using only 9 pallets. This example also shows that Johnson’s makespan is not attainable for $K < 5$ pallets. This exercise suggests the following definitions.

Consider a job set $\mathcal{J} = \{J_1, \ldots, J_n\}$ with processing times $a_j$ and $b_j$ on machine $M_1$ and $M_2$, respectively, for $j = 1, \ldots, n$. Reorder the processing times so that $a_{(1)} \leq a_{(2)} \leq \cdots \leq a_{(n)}$ and $b_{(1)} \leq b_{(2)} \leq \cdots \leq b_{(n)}$.

**Definition 1** We say that $\mathcal{J}$ is an instance of order $\beta$, where $\beta$ is a non-negative integer, if

$$b_{(n)} \leq \sum_{j=1}^{\beta} a_{(j)}$$

and

$$a_{(n)} \leq \min\left(\sum_{j=1}^{\beta} a_{(j)}, \sum_{j=1}^{\beta} b_{(j)}\right).$$

For instance, the example depicted in Figure 2 is an instance of order 4. Note that an instance of order $\beta$ is also an instance of order $\beta + 1$. Also note that condition (1) is equivalent to saying that, for any $J_i$ and a subset of $\mathcal{J}$, $A$, $|A| = \beta$, we have

$$b_i \leq \sum_{j \in A} a_j.$$

A similar observation can be made for condition (2).

We shall only consider instances of order $\beta$ throughout this section, unless stated otherwise.

We may now define $\mathcal{D}_\beta$ to be the set of all instances of order $\beta$. Conversely, given any set $\mathcal{D}$, we can compute the minimal $\beta$ value for each instance in $\mathcal{D}$ and use the greatest one as the $\beta$ of the set. From Figure 2, we see that there exist instances of order $\beta$ that require at least $K = \beta + 1$ pallets to achieve Johnson’s makespan. In the remainder of this section, we shall prove that $\beta + 2$ pallets are sufficient to guarantee the achievement of the Johnson’s makespan for all instances in $\mathcal{D}_\beta$.
To this end, let us consider a Johnson schedule in which the second op­
eration starts as soon as machine $M_2$ is available. Such a schedule will then look like the one in Figure 3. We see that the schedule is segmented into $q$ subschedules, and in each subschedule, the first job can be processed without waiting for $M_2$. Let the first jobs of each subschedule be $J_{i1}, J_{i2}, \ldots, J_{iq}$. Then in intervals $[S_{i1,1}, C_{i1,1}], \ldots, [S_{iq,1}, C_{iq,1}]$, only one pallet is required to serve $M_1$, and at $C_{i1,1}, \ldots, C_{iq,1}$, all but one pallet become available. Now it is clear that if we can preserve the makespan for each subschedule with $K$ pallets, and if at least one pallet becomes available at the completion of the last a­
operation of the subschedule, then Johnson’s makespan is also attained using $K$ pallets. In other words, if the jobs in each subschedule can be resequenced to achieve the same makespan using $K$ pallets, then Johnson’s makespan is preserved.

Johnson’s algorithm proceeds by first partitioning the jobs into two sets, set $U = \{J_j \mid a_j \leq b_j\}$ and set $V = \{J_j \mid a_j > b_j\}$, after which the jobs in $U$ are sequenced in nondecreasing order of the $a_j$’s and the jobs in $V$ in the nonincreasing order of the $b_j$’s—the final sequence is the ordered $U$ followed by the ordered $V$ (Johnson, 1954).

Based on the working of Johnson’s algorithm, we first analyze three spe­
cial types of Johnson’s schedules. When it does not cause any confusion, we shall simplify the notation by indexing the jobs according to their positions in a schedule; thus $J_1$ is the first job scheduled, $J_2$ the second, and so on. The index of a job is determined only after the job is scheduled. Let us denote

\[ A(u, v) = \sum_{j=u}^{v} a_j, \]
\[ B(u, v) = \sum_{j=u}^{v} b_j. \]
Consider a special class of job sets that admits so-called *left-compact* schedules. A schedule is said to be left-compact if it satisfies

\[ S_{12} = C_{11} \quad \text{and} \quad C_{r2} = C_{r-1,2} + p_{r2}, \quad \text{for} \quad 2 \leq r \leq n, \]  

(3)

that is, all the operations can be scheduled contiguously with the first \( b \)-operation started *immediately* as the first \( a \)-operation finishes. Note that (3) implies that

\[ B(1, r) \geq A(2, r + 1), \quad \text{for all} \; 1 \leq r \leq n - 1. \]  

(4)

Now we define three special left-compact Johnson’s schedules called *U*-type, *V*-type, and *UV*-type, respectively. Apart from (3), they also satisfy the following conditions:

(i) *U*-type: \( a_j \leq b_j \forall j, \quad \text{and} \quad a_1 \leq a_2 \leq \cdots \leq a_n; \)

(ii) *V*-type: \( a_j > b_j \forall j, \quad \text{and} \quad b_1 \geq b_2 \geq \cdots \geq b_n; \)

(iii) *UV*-type: there is an \( r, 1 < r < n, \) such that \( a_j \leq b_j \) and \( a_1 \leq a_2 \leq \cdots \leq a_r \) for \( 1 \leq j \leq r; \) and \( a_j > b_j \) and \( b_{r+1} \geq b_{r+2} \geq \cdots \geq b_n \) for \( r < j \leq n. \)

Figure 2(a) shows a *UV*-type schedule.

From Figure 3 we can infer that any given Johnson’s schedule can be decomposed into a number of *U*-type schedules followed by a *UV*-type schedule and then by a number of *V*-type schedules. Note that there can be at most one *UV*-type subschedule in any Johnson’s schedule. We now impose the pallet constraint and reschedule the jobs in an attempt to maintain the makespan. We first show that \( \beta + 1 \) pallets are sufficient for a *V*-type schedule.

### 3.1 Rescheduling a *V*-type schedule

**Theorem 2** If \( I \) is an instance of order \( \beta \) that permits a *V*-type left-compact schedule \( \sigma \), then \( \beta + 1 \) pallets are sufficient.

**Proof.** As we have \( \beta + 1 \) pallets available, we must show that when the \((r + \beta + 1)\)-th job is started at least \( r \) jobs have finished, i.e., we must
show that $A(1, r + \beta) \geq a_1 + B(1, r)$, for $r = 1, \ldots, n - \beta$. This follows immediately from the inequalities $B(2, r) < A(2, r)$ (we are dealing with a $V$-type schedule) and $b_1 \leq A(r + 1, r + \beta)$ (as we have an instance of order $\beta$).

3.2 Rescheduling a $U$-type schedule

We now turn to rescheduling a $U$-type schedule with the restriction of $\beta + 1$ pallets. We distinguish two cases, where the distinction is based on whether $A(2, n) \geq B(1, n - \beta)$ or not. We first deal with the case that $A(2, n) \geq B(1, n - \beta)$. The algorithm described below reschedules such a $U$-type schedule for $\beta + 1$ pallets.

**Algorithm U**

(i) If $n \leq \beta + 1$, then stop.

(ii) Find the smallest $r$ such that $B(1, r) > A(2, r + \beta)$ (i.e., when pallet shortage is imminent and idle time on $M_1$ is created when executing the jobs in the current order). If no such $r$ exists, then stop.

(iii) Find the smallest $u > r + \beta$ such that $a_u > b_{u-\beta}$.

(iv) Reschedule $J_u$ immediately after $J_r$. Reindex the jobs according to the new schedule and go to (ii).

**Theorem 3** Given a $U$-type left-compact schedule of an instance of order $\beta$ with $A(2, n) \geq B(1, n - \beta)$ as the input, Algorithm U produces a left-compact schedule without idle time on $M_1$ that uses no more than $\beta + 1$ pallets.

**Proof.** We first show that we can always find an index $u > r + \beta$ such that $a_u > b_{u-\beta}$ (if $r$ exists). Assume to the contrary that $a_u \leq b_{u-\beta}$ for each $u = r + \beta + 1, \ldots, n$; this implies that $A(r + \beta + 1, n) \leq B(r + 1, n - \beta)$. Moreover, we have that $A(2, r + \beta) < B(1, r)$. Combining these inequalities yields $A(2, n) < B(1, n - \beta)$, which contradicts the assumption.

Second, we show that scheduling $J_u$ immediately after $J_r$ does not produce any idle time on $M_2$. This proceeds by simply checking the completion times.
of the jobs. The first job of interest is $J_u$: we must show that $A(2,r) + a_u \leq B(1,r)$. But this is straightforward, as $B(1,r) > A(2,r + \beta) = A(2,r) + A(r+1,r+\beta) \geq A(2,r) + a_u$. For job $J_s$ ($s = r+1, \ldots, u-1$), we must show that $A(2,s) + a_u \leq B(1,s-1) + b_u$. This follows immediately by adding up the inequalities $B(1,r) > A(2,r) + a_u$, $A(r+1,s-1) \leq B(r+1,s-1)$ ($a_j \leq b_j$ for $j = r+1, \ldots, s-1$), and $a_s \leq a_u \leq b_u$. From job $u+1$ onwards, the situation has not changed.

Finally, we show that in each iteration of the algorithm the index $u$ is increased; hence, after at most $O(n)$ iterations, we are not able to find an index $u$ any more, which implies that clearly no index $r$ can exist (see the first part of the proof). Consider the situation that arises when job $J_u$ is moved to immediately after job $J_r$; for the ease of exposition, we do not reindex the jobs. Suppose that the value of $r$ does not change. When we look for a new index $u$, we first compare $a_r + \beta$ to $b_u$, and if $a_r + \beta \leq b_u$, then we compare $a_{r+s}$ to $b_{r+s-\beta}$, for $s = \beta + 1, \ldots, n - r$ until we find an $s$ such that $a_{r+s} > b_{r+s-\beta}$. But by choice of $u$, we already know that $a_r + \beta \leq b_{r+s-\beta}$, for $s = 1, \ldots, u - \beta - 1$, as these pairs were checked in the previous iteration. Hence, if we can prove that $a_r + \beta \leq b_u$, then we are in business (note that if $r$ is increased, then we are done immediately, as we then do not even take the index $(r + \beta)$ into consideration). We show this by induction. Our induction hypothesis is that until now the index $u$ has increased in each iteration. As the jobs were indexed in order of non-decreasing $a_j$ value in the input schedule, we know that the original index of the current job $(r + \beta)$ was at most equal to $u - 1$, which implies that $a_r + \beta \leq a_u \leq b_u$.

We now come to the second case, in which we have that $A(2,n) < B(1,n-\beta)$. In this case, $M_2$ still has to execute more than $\beta$ jobs at the time $M_1$ is ready with processing the jobs in this segment. This implies, however, that this segment is the last one: otherwise, the first job in the next segment, say $J_{r+1}$, would have $a_{r+1} > B(n-\beta + 1,n)$ by definition of the segment, which contradicts the definition of $\beta$. Hence, we are allowed to insert idle time on $M_1$, as long as this does not delay any operation on $M_2$. The strategy is simple then: execute the jobs in the given order and insert idle time on $M_1$ when there are no pallets available. Suppose that the first time that there is no pallet available occurs when job $r + \beta + 1$ is about to be started on $M_1$; this start is then delayed until the time that job $r$ is completed by $M_2$. None of the jobs $r + 1, \ldots, n$ will be delayed on $M_2$, unless $A(r + \beta + 1,s) >$
$B(r+1, s-1)$ for some $s \in \{r+\beta+1, n\}$. But this clearly cannot happen, as $A(r+\beta+1, s-1) \leq B(r+\beta+1, s-1)$ (since $a_j \leq b_j$) and $a_s \leq B(r+1, r+\beta)$ (by definition of $\beta$). This proof can be repeated each time idle time is inserted on $M_1$.

### 3.3 Rescheduling a UV-type schedule

We now come to the case of rescheduling a UV-type schedule with the restriction of $\beta+2$ pallets. Our approach is very similar to the one employed in the rescheduling of the U-type schedule, with one major exception: we now sometimes reschedule $J_u$ immediately *before* $J_r$. Just like in the case of rescheduling the U-type schedule, we distinguish two cases: $A(2, n) \geq B(1, n - \beta - 1)$ and $A(2, n) < B(1, n - \beta - 1)$ (the right-hand side is now $B(1, n - \beta - 1)$ instead of $B(1, n - \beta)$, as we have $\beta+2$ pallets available). In the first case, inserting idle time on $M_1$ is prohibited, but we know that then $J_u$ must exist; in the second case, the insertion of idle time on $M_1$ is allowed. **Algorithm UV** below deals with the first case.

**Algorithm UV**

(i) If $n \leq \beta+2$, then stop.

(ii) Find the smallest $r$ such that $B(1, r) > A(2, r + \beta + 1)$. If no such index $r$ exists, then stop.

(iii) Find the smallest $u > r + \beta + 1$ such that $a_u > b_{u-\beta-1}$.

(iv) If $B(1, r) > A(2, r+\beta)+a_u$ and $a_u > b_u$, then reschedule $J_u$ immediately before $J_r$; otherwise, schedule $J_u$ immediately after $J_r$.

(v) Reindex the jobs according to the new schedule. Set $i = i + 1$ and go to (ii).

**Theorem 4** Given a UV-type left-compact schedule of an instance of order $\beta$ with $A(2, n) \geq B(1, n - \beta - 1)$ as the input, **Algorithm UV** produces a left-compact schedule without idle time (except for the initial idle time on $M_2$) that uses no more than $\beta+2$ pallets.
Proof. Observe that we can always find an index $u$ such that $a_u > b_{u-\beta-1}$ (if $r$ exists), as we would have $A(2,n) < B(1,n-\beta-1)$ otherwise (see the corresponding part of the proof of Theorem 3).

First, suppose that $J_u$ is a $U$-type job, that is, $a_u \leq b_u$. In that case, ALGORITHM UV behaves in exactly the same way as ALGORITHM U (the availability of an extra pallet influences the choice of the index $u$ only). As we start with a $UV$-type schedule, we know that, if $J_u$ is a $U$-type job, then all jobs $J_1, \ldots, J_u$ are $U$-type jobs. Hence, we can apply the proof of Theorem 3 to show that moving $J_u$ to immediately after $J_r$ does not yield additional idle time on $M_2$ and that in each iteration the value of $u$ increases.

Now, suppose that $J_u$ is a $V$-type job, that is, $a_u > b_u$. Then, depending on whether $B(1,r) \leq A(2,r+\beta) + a_u$ or not, we reschedule $J_u$ immediately after or immediately before $J_r$.

First, suppose that $B(1,r) \leq A(2,r+\beta) + a_u$. If we reschedule $J_u$ immediately after $J_r$, then there is a pallet available when job $J_{r+\beta+1}$ is ready to start on $M_1$; this implies that after reindexing the jobs the index $r$ has increased by at least one. What remains to be proven is that rescheduling $J_u$ immediately after $J_r$ does not lead to additional idle time on $M_2$. First, consider $J_u$: we must prove that $J_u$ is completed by $M_1$ at the time that $J_{u-1}$ is completed at $M_2$, which implies that $J_u$ is done on $M_1$ at the time $J_{u-1}$ is completed at $M_2$. Finally, consider any job $J_j$ with $j \in \{r+1, \ldots, u-\beta-1\}$. We can prove the desired inequality $A(2,j) + a_u \leq B(1,j-1) + b_u$ by adding up the inequalities $A(2,r+\beta+1) < B(1,r)$ (definition $r$); $A(r+\beta+2,j+\beta) \leq B(r+1,j-1)$ (definition $r$); and $a_u \leq A(j+1,j+\beta) + b_u$ (definition of $u$), which implies that $J_j$ is done on $M_1$ at the time $J_{j+1}$ is completed at $M_2$. Finally, consider any job $J_j$ with $j \in \{u-\beta, \ldots, u-1\}$. We can prove the desired inequality $A(2,j) + a_u \leq B(1,j-1) + b_u$ by adding up the inequalities $A(2,r+\beta+1) < B(1,r)$ (definition $r$) and $A(r+\beta+2,u-1) \leq B(r+1,u-\beta-2)$ (definition $u$) and the inequality $a_u \leq B(u-\beta-1,j-1) + b_u + A(j+1,u-1)$; we will show later why this inequality holds.

Now we turn to the situation $B(1,r) > A(2,r+\beta) + a_u$, in which case we reschedule $J_u$ immediately before $J_r$. Below we will show that this move does not lead to pallet shortage at the start of $J_{r+\beta+1}$, which means that at the next iteration after reindexing the index $r$ has increased, and that this
move does not lead to additional idle time on $M_2$. As to the first part, we must show that $A(2, r + \beta) + a_u \geq B(1, r - 1) + b_u$; this follows immediately by adding up the inequalities $A(2, r + \beta) \geq B(1, r - 1)$ (definition of $r$) and $a_u > b_u$. To show that we do not have to insert additional idle time on $M_2$, we start with checking the situation at the start of $J_u$ and $J_r$ on $M_2$. For $J_u$, we must show that $A(2, r + f_3) + a_u \leq B(1, r - 1)$, which follows by adding up the inequalities $A(2, r + f_3) + a_u < B(1, r)$ and $b_r \leq A(r + 1, r + \beta) + b_u$.

For the jobs after $J_r$ the situation is exactly the same as in the case that we rescheduled $J_u$ immediately after $J_r$. As we proved above that we did not need any additional idle time on $M_2$, without using the inequality $B(1, r) \leq A(2, r + \beta) + a_u$, we are done, if we tie up the loose end of showing that $a_u \geq B(u - f_3 - 1, j - l) + b_u$ for $j = u - \beta, \ldots, u - 1$.

To show that $a_u \geq B(u - \beta - 1, j - 1) + b_u + A(j + 1, u - 1)$ for $j = u - \beta, \ldots, u - 1$, we need the special structure of the original schedule before rescheduling. First, suppose that $b_i \geq a_i$ for $i = u - \beta - 1, \ldots, j - 1$; in this case, we obtain that $B(u - \beta - 1, j - 1) + b_u + A(j + 1, u - 1) \geq A(u - \beta - 1, j - 1) + b_u + A(j + 1, u - 1) \geq a_u$ by definition of $\beta$. If $b_i < a_i$ for some $i \in \{u - \beta - 1, \ldots, j - 1\}$, then we must have $b_i < a_i$ for each $i \in \{j + 1, \ldots, u - 1\}$, as the original schedule was of $UV$-type. But then, we find that $B(u - \beta - 1, j - 1) + b_u + A(j + 1, u - 1) \geq B(u - \beta - 1, j - 1) + b_u + B(j + 1, u - 1) \geq a_u$ by definition of $\beta$. Hence, we are done if we can show that our rescheduling activities do not change the $UV$-type nature of the jobs that are currently indexed $u - \beta - 1, \ldots, u - 1$. Therefore, we must show that in this part of the schedule there is no $V$-type job that precedes a $U$-type job. But this is immediately clear if we look at the position that $J_u$ is moved to. If $J_u$ is rescheduled immediately before $J_r$, then in the next iteration this job will have index no more than $r < u - \beta - 1$, which implies that it will never occupy one of the positions $u - \beta - 1, \ldots, u - 1$, as the index $u$ increases in each iteration. Similarly, if $J_u$ is rescheduled immediately after $J_r$, then this job will occupy position $r + 1$ in the next iteration, with $r + 1 < u' - \beta - 1$, where $u'$ is the index $u$ in the next iteration.

We have thus proven that each time that we do not have to insert any additional idle time on $M_2$ when we reschedule $J_u$, and that in each iteration the index $r$ is increased. Hence, after $O(n)$ iterations, we end up with a left-compact schedule without idle time on $M_1$ that uses no more than $\beta + 2$
pallets.

We now come to the second case: $A(2, n) < B(1, n - \beta - 1)$. Then, the current segment must be the last one in the schedule, which implies that we are allowed to insert idle time on $M_1$ to avoid pallet problems, as long as we do not delay any operation on $M_2$. Given such a segment, we first apply Algorithm UV. Suppose that the algorithm ends with sequence $\sigma$ when it is unable to find a job $J_u$ with $a_u > b_{u-\beta-1}$. Then we apply the same strategy as we did in the similar case of the $U$-type segment: execute the jobs in the given order $\sigma$ and insert idle time on $M_1$ when there are no pallets available. Suppose that the first time that there is no pallet available occurs when job $r + \beta + 2$ is about to be started on $M_1$; this start is then delayed until the time that job $r$ is completed by $M_2$. None of the jobs $r + 1, \ldots, n$ will be delayed on $M_2$, unless $A(r + \beta + 2, s) > B(r + 1, s - 1)$ for some $s \in \{r + \beta + 1, n\}$. But this clearly cannot happen, as $a_j \leq b_{j-\beta-1}$ for $j = r + \beta + 2, \ldots, n$ (since we could find a job $J_u$), which implies that $A(r + \beta + 2, s) \leq B(r + 1, s - \beta - 1) \leq B(r + 1, s - 1)$. This proof can be repeated each time idle time is inserted on $M_1$.

We are now ready to prove the main result of this section.

**Theorem 5.** If $m = 2$ and $\mathcal{J}$ is an instance of order $\beta$, then

(a) Johnson's makespan can be achieved using at most $\beta + 2$ pallets.

(b) at least $\beta + 1$ pallets are needed to guarantee Johnson's makespan for all instances of order $\beta$.

**Proof.** We only prove the first claim; the last part follows immediately from the example depicted in Figure 2.

Let $\sigma$ be the schedule that results when applying Johnson's algorithm to $\mathcal{J}$. We decompose $\sigma$ into left-compact $U$, $V$, or $UV$-type segments, and then run them through Algorithms U, V, or UV. Each of the algorithms returns a left-compact schedule without idle time on $M_1$, except for the last segment, which may contain some additional idle time on $M_1$. We now patch the revised segments up again. To prove the claim, it suffices to show that the pallet constraint is not violated on both sides of the border of two adjacent segments.
Let us consider any two adjacent subschedules; let $J_v$ be the last job in the first subschedule, and let $J_w$ be the first job in the second subschedule. Due to the way we decomposed $\sigma$ into segments, we know that $J_w$ is completed on $M_1$ after the completion of $J_v$ on $M_2$, that is, $C_{w,1} > C_{v,2}$. Hence, if there is a pallet available at the start of $J_w$ on $M_1$, then we can patch up the segments without pallet problems, as the remaining pallets will all be available at time $C_{w,1}$. Now look at the situation at the time that $J_w$ is ready to start on $M_1$. The pallets are occupied by jobs from the first segment that still need to be completed by $M_2$, but they all complete in a period of time with length less than $a_w$. Hence, there can be at most $\beta - 1$ of these unfinished jobs, which means that there is a pallet available for $J_w$.

Hence, for any given job instance, we can compute its $\beta$ value and define

$$K_\beta = \beta + 2$$

to be the minimum number of pallets that guarantees Johnson’s makespan.

4 Computational results

In this section, we discuss further the problem of choosing an appropriate number of pallets for the two-machine flowshop.

The previous section provides a worst-case $K_\beta$ that guarantees the sufficiency of pallets for the entire set of instances. However, the worst case is not important for real-world situations if it seldom happens. Of more concern is the performance of a given policy in typical situations. Furthermore, it is not necessary to achieve the Johnson makespan for all possible cases; a schedule may be acceptable as long as its makespan is within a tolerance of the Johnson makespan. Unfortunately, we do not have a good method for evaluating the performance of a given number of pallets in an arbitrary situation. Moreover, the algorithms of the previous section are not so suitable for most instances, since they are designed for the worst case.

In order to better understand how the makespan is affected by $K$, we conduct some computational experiments. The experiments are inspired by three observations:

(i) the makespan of a schedule is a nonincreasing function of $K$ for a given job sequence. The marginal reduction in the makespan with
each additional pallet is important in determining a proper number of pallets.

(ii) the Gilmore-Gomory algorithm, which solves the problem to optimality if $K = 2$, may not give the the optimal solution even if $K = n$. Alternatively, Johnson's algorithm, which solves the problem to optimality if $K \geq n$, may require many more pallets than really necessary to enforce a schedule with Johnson's makespan. Hence, for any $K \geq 3$, it is interesting to investigate the empirical performance of both algorithms.

(iii) $K_\beta$ pallets are sufficient for any given instance of order $\beta$, but it may be too large in most situations, since it is derived for the worst case. It is worthwhile to investigate how fast the makespan will decrease when the number of pallets $K$ increases from 2 to $K_\beta$, and hence to determine a more appropriate number.

In the experiments, we generate uniformly distributed processing times for problem instances with 50 jobs up to 500 jobs. The processing times for both operations are uniformly distributed over $[1, 100]$.

In the first experiment, we first find for each instance Johnson's schedule and compute its makespan, $C_{\text{max}}^\infty$, and the number of pallets used to reach this makespan, $K_J$. Meanwhile, we compute the $\beta$ value, and hence $K_\beta = \beta + 2$, for the instance. (We set $K_\beta = n$ if $\beta + 2 \geq n$.) Then we derive the Gilmore-Gomory schedule and use the resulting sequence and $K = K_\beta$ pallets to compute the corresponding makespan $C_{GG}^{K_\beta}$.

If $C_{GG}^{K_\beta} > C_{\text{max}}^\infty$, then we say that the Gilmore-Gomory algorithm fails to achieve Johnson's makespan (regardless of the number of pallets). If $C_{GG}^{K_\beta} = C_{\text{max}}^\infty$, we progressively reduce the number of pallets, one at a time, until we find a minimal $K_G$ such that $C_{GG}^{K_G} = C_{\text{max}}^\infty$ still holds. Our first finding is that the Gilmore-Gomory algorithm fails in one third of the instances we have generated. This result is listed in Table 4; it is actually better than we expected. We noted that for those successful instances, three to four pallets are sufficient in almost all the instances, and at most six pallets are required to achieve Johnson's makespan. It is also interesting to note the homogeneity of the distribution of $K_G$ required with respect to the number of jobs. In contrast, Table 5 shows the distribution of $K_J$, a surrogate of $K_\beta$, which drifts rapidly to the right as the number of jobs increases (the distributions for $K_J$ are very close to those for $K_\beta$ in our experiments).
The makespan using the Gilmore-Gomory sequence with \( K_m \) pallets;

\[ K'_G = \text{number of pallets required in an improved Gilmore-Gomory schedule.} \]

It is derived as follows: We iteratively reduce \( K \) with an initial value \( K_G \), and in each iteration submit the Gilmore-Gomory schedule as input to Algorithm UV. The procedure is repeated until the makespan increases.

\[ C_{GH} = \text{the makespan derived when obtaining } K'_G; \]

\[ C_H = \text{the makespan derived by running the Johnson’s schedule through Algorithm UV with } K'_G \text{ pallets if } K'_G < K_J; \]

\[ C_G = \text{Gilmore-Gomory makespan with no limit on pallets;} \]

\[ r_G = \frac{C_G}{C_{max}}; \]

\[ r_{GH} = \frac{C_{GH}}{C_{max}}; \]

\[ r_H = \frac{C_H}{C_{max}}. \]

Table 3: Notation with respect to the computational results.

The second experiment is to examine the relative performance of the Gilmore-Gomory schedule if sufficient pallets are provided and the degree of improvement if Algorithm UV is used to improve the schedules.

Table 4 gives an overview of our notation used in our reports on the computational experiments and results.

We first determine \( K_m \) and \( C_G \) and we find \( K'_G \) as well as \( C_{GH} \). If \( \min\{K_m, K'_G\} < K_J \), we compute \( C_H \).

The results are encouraging. Both \( r_G \) and \( r_{GH} \) are negligibly small; see Table 7 that gives the estimates of \( r_G \) and \( r_{GH} \). The distributions for \( K'_G \) are listed in Table 6, from which we see that the number of pallets required is effectively under 5. Note that \( K'_G \) is the number of pallets needed to achieve \( C_G \); note that \( C_G \geq C_{max} \). The slight difference between \( r_G \) and \( r_{GH} \) indicates that while Algorithm UV can effectively reduce the number of pallets used, it does little to improve makespan. Also note that the relatively high \( r_H \) shows that Johnson’s schedule is not a good approximation if \( K \) is small.

We then focus on the instances for which the Gilmore-Gomory algorithm fails. For these instances we compute \( r(K) \), the ratio of the makespan using the Gilmore-Gomory sequence with \( K \) pallets over Johnson’s makespan.
<table>
<thead>
<tr>
<th># jobs</th>
<th>Frequency (%)</th>
<th>$K_G$</th>
<th>Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>9 43 9 1 0</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>5 42 16 1 0</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>4 43 16 5 1</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>2 52 12 3 0</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>1 42 14 1 2</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>2 42 19 1 0</td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: The empirical distributions (%) of $K_G$ as a function of $n$ ($m = 2$).

<table>
<thead>
<tr>
<th># jobs</th>
<th>Frequency (%)</th>
<th>$K_J$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>12 14 13 16 19 16 5 4 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1 4 8 5 6 7 5 18 18 17 5 5 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>1 2 2 7 6 13 22 9 18 10 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>3 2 4 4 12 16 10 9 14 12 7 2 4 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>1 1 5 7 11 18 14 19 10 8 4 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>1 1 2 4 12 7 19 13 15 12 5 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: The empirical distributions (%) of $K_J$ as a function of $n$. 
### Table 6: The empirical distributions (%) of $K_G'$ as a function of $n$.

<table>
<thead>
<tr>
<th># jobs</th>
<th>$K_G'$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td>200</td>
<td>4</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
</tr>
<tr>
<td>400</td>
<td>1</td>
</tr>
<tr>
<td>500</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 7: The performance of the Gilmore-Gomory schedule with $K_G$ and $K_G'$ pallets.

<table>
<thead>
<tr>
<th># jobs</th>
<th>$r_G$</th>
<th>$r_{GH}$</th>
<th>$r_H$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1.0043</td>
<td>1.0039</td>
<td>1.2007</td>
</tr>
<tr>
<td>100</td>
<td>1.0013</td>
<td>1.0012</td>
<td>1.2385</td>
</tr>
<tr>
<td>200</td>
<td>1.0005</td>
<td>1.0004</td>
<td>1.2733</td>
</tr>
<tr>
<td>300</td>
<td>1.0005</td>
<td>1.0005</td>
<td>1.2902</td>
</tr>
<tr>
<td>400</td>
<td>1.0003</td>
<td>1.0003</td>
<td>1.2912</td>
</tr>
<tr>
<td>500</td>
<td>1.0002</td>
<td>1.0002</td>
<td>1.3015</td>
</tr>
</tbody>
</table>


23
Table 8: The ratios of the Gilmore-Gomory makespan given $K$ over Johnson's makespan for instances on which Gilmore-Gomory failed to give Johnson's makespan.

Columns 2-5 of Table 8 list their average. The last column shows the largest ratio we encountered. As a comparison, Table 9 lists the ratios for the successful instances. These ratios help explain why four or five pallets are sufficient in most cases. In our experiment, the largest value for the ratio $C_{GG}^2/C_{max}^\infty$ is less than 1.17. In other words, no matter how many pallets are used, the saving is at most 17%. Since the difference between the Gilmore-Gomory solution and Johnson's makespan is small, extra idle times in the Gilmore-Gomory schedule caused by pallet shortage can be expected not to be significant. Thus using four or five pallets is quite effective in reducing the shortage or even eliminate it in most situations.

To test the impact of the distribution intervals on the performance, we repeated the above computations by setting different distribution intervals for the two operations. First we let the $a$-operations be distributed over [1, 100] and the $b$-operations over [1, 200], and then we exchange the intervals so that the $a$-operations are distributed over [1, 200] and the $b$-operations over [1, 100]. In both situations, the Gilmore-Gomory algorithm performed better.

3Note that there are two sources for causing a greater makespan in this situation: One is due to shortage of pallets, and the other is inherent to the Gilmore-Gomory solutions which cannot be eliminated by using more pallets.
Table 9: The ratios of the Gilmore-Gomory makespan given \( K \) over the Johnson’s makespan for those instances for which the Gilmore-Gomory succeeded to produce Johnson’s makespan.

<table>
<thead>
<tr>
<th># jobs</th>
<th>2</th>
<th>3</th>
<th>≥ 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1.0076</td>
<td>1.0005</td>
<td>1.0001</td>
</tr>
<tr>
<td>100</td>
<td>1.0074</td>
<td>1.0004</td>
<td>1.0000</td>
</tr>
<tr>
<td>200</td>
<td>1.0063</td>
<td>1.0005</td>
<td>1.0001</td>
</tr>
<tr>
<td>300</td>
<td>1.0032</td>
<td>1.0001</td>
<td>1.0000</td>
</tr>
<tr>
<td>400</td>
<td>1.0032</td>
<td>1.0001</td>
<td>1.0000</td>
</tr>
<tr>
<td>500</td>
<td>1.0032</td>
<td>1.0002</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

5 Conclusion

We have analyzed the problem of scheduling \( n \) jobs in a pallet-constrained flowshop so as to minimize the makespan. This problem is shown to be related to the problem of scheduling a flowshop with finite buffers in the sense that a feasible solution to the latter is also feasible to the former, but not vice versa.

Particularly, we focused on the impact of the number of pallets on the makespan in the two-machine flowshop. It is an NP-hard problem to find the minimum number of pallets for a given instance subject to an upper bound on makespan. However, for a class of job instances, we are able to derive the minimum number of pallets that is needed in a two-machine flowshop in order to guarantee the least possible makespan. Recognizing that this number is derived with respect to the worst case, we conducted computational experiments on randomly generated instances to evaluate the performance of two approximation algorithms: Johnson’s algorithm, which solves the problem to optimality if \( K \geq n \), and Gilmore-Gomory’s algorithm, which solves the problem to optimality if \( K = 2 \). Both algorithms can straightforwardly be adapted to the situation where there are \( K \) pallets available, with \( 3 \leq K < n \). Our finding is that the Gilmore-Gomory algorithm performs surprisingly well in a two-machine flowshop: with no more than six pallets,
irrespective of the number of jobs to be scheduled, it consistently produces schedules with makespan very close to optimal.

References


