## Master Thesis

# Scheduling a Proportionate Flow Shop of Batching Machines

Christoph Hertrich

October 12, 2018

**Department of Mathematics**
**Technische Universität Kaiserslautern**

| | | |
|---|---|---|
| Supervisor: | Prof. Dr. Sven O. Krumke | TU Kaiserslautern |
| Co-Supervisors: | Dr. Heiner Ackermann | Fraunhofer ITWM |
| | Dr. Sandy Heydrich | Fraunhofer ITWM |
| | Dr. Christian Weiß | Fraunhofer ITWM |



# Contents

# 1. Introduction

Cutting-edge cancer therapy involves producing individualized medicine for many patients at the same time. Within this process, most steps can be completed for a certain number of patients simultaneously. Using these resources efficiently may significantly reduce waiting times for the patients and is therefore crucial for saving human lives. However, this involves solving a complex scheduling problem, which can mathematically be modeled as a *proportionate flow shop of batching machines* (PFB). In this thesis we investigate exact and approximate algorithms for tackling many variants of this problem. Related mathematical models have been studied before in the context of semiconductor manufacturing, see e.g. [LUMV92, MS06].

## 1.1. Problem Description

In the most basic form of a PFB, $n \in \mathbb{N}$ identical jobs have to pass through a sequential production process consisting of $m \in \mathbb{N}$ machines. The machines are arranged as a *flow shop*, that is, each job has to be processed on the machines $1, \dots, m$ in this order. Each machine $i \in [m] = \{1, \dots, m\}$ is a *batching machine*, which means that a set of up to $b_i \in \mathbb{N}$ jobs can be started on it simultaneously. Such a set is called a *batch*, the values $b_i$ are called *batch capacities*. In a PFB, the *processing time* $p_i \in \mathbb{R}_{>0}$ of a batch on machine $i$ does not depend on which jobs and how many of them are included in the batch. During this time the machine is occupied and can neither suspend the current batch nor start a new one.

We consider the problem variants with and without *release dates* $r_j \in \mathbb{R}_{\geq 0}$, $j \in [n]$. The processing of job $j$ on the first machine may not start before the job has been released. The case without release dates may be modeled by setting $r_j = 0$ for all $j \in [n]$.

Given this situation, the task is to decide for each machine how to split the set of jobs into batches and when these batches should be scheduled. We consider several common scheduling objectives, e.g. minimization of the *makespan*, which is the time elapsed until all jobs have been completed.

The main part of this thesis deals with the *offline* version, where all data is known in advance. However, we are also interested in the *online* version, where each job is unknown until its release date. In particular, this means that the total number $n$ of jobs remains unknown until the end of the scheduling process.

**Example 1.1.** Consider a PFB without release dates given by $n = 5$, $m = 3$, $b_1 = 2$, $b_2 = 3$, $b_3 = 4$, $p_1 = 1$, $p_2 = 2$, $p_3 = 3$. Figure 1.1 illustrates a feasible schedule for the instance as job-oriented Gantt chart.

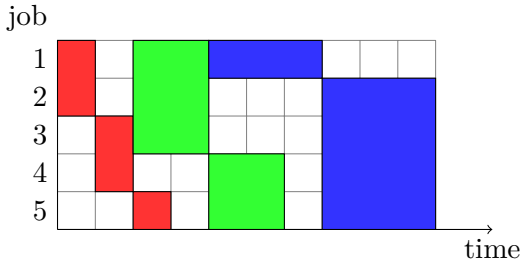


Figure 1.1.: A feasible schedule.

Each rectangle represents a batch of jobs processed together on a machine. The colors red, green, and blue of the rectangles encode the machines 1, 2, and 3, respectively. Note that in this example none of the batches can be started earlier, since either a job of the batch has just arrived when the batch is started, or the machine is occupied before. Still, the schedule does not minimize the makespan, since the schedule shown in Figure 1.2 is feasible as well and has a makespan of 9 instead of 10. We will see below that 9 is indeed the optimal makespan.

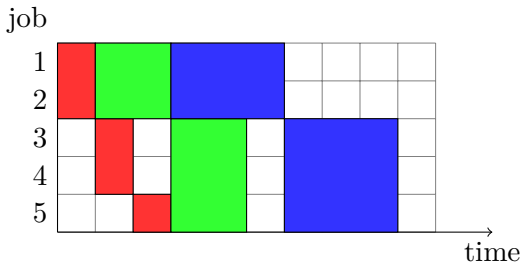


Figure 1.2.: A schedule minimizing the makespan.

The improvement in the makespan was achieved by reducing the size of the first batch on the second machine from three to two, which allows to start it one time step earlier. Observe that the first job can never finish the last machine before time step 6. Moreover, not all five jobs fit into one batch on the last machine. Hence, a second batch on the last machine is necessary in any schedule. Therefore, 9 is indeed the optimal makespan.

## 1.2. Classification of the Problem

Generally, a flow shop with job-independent processing times is called a *proportionate flow shop*. There are different definitions of the term *proportionate* in literature. Our

case is a special case of a general definition in [PSK13], where a flow shop is defined to be proportionate, if the processing time $p_{ij}$ of job $j$ on machine $i$ is given as the quotient $\frac{\overline{p}_j}{s_i}$ of a job processing time and a machine speed.

In standard scheduling literature, e.g. in [Bru07], two different kinds of batching machines are commonly distinguished. For a *serial batching machine*, the processing time of a batch is the sum of the processing times of the jobs included in the batch. Usually, an additional setup time is incurred. In contrast, for a *parallel batching machine*, the processing time of a batch is the maximum processing time of a job included in it. In a PFB, parallel batching machines are involved.

Finally, we give a way to define the problem of scheduling a PFB via the famous three-field notation, which was first introduced in [GLLR79]:

$$F \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid f.$$

Here, $F$ denotes a flow shop problem, $r_j$ defines that the jobs may have release dates, $p_{ij} = p_i$ states that processing times are independent from the jobs, but depend on the machines, $p$-batch denotes that jobs can be batched in parallel, $b_i$ denotes that there are upper bounds on the batch sizes which depend on the machine, and $f$ stands for an arbitrary objective function.

## 1.3. Literature Review

For most of this work we use the book of Brucker [Bru07] as a standard reference for definitions and basic results in the area of scheduling. Moreover, Brucker and Knust [BK] provide a comprehensive collection of complexity results for many scheduling problems.

Many flow shop problems have been proven to be NP-hard, see e.g. [GJS76]. However, these traditional complexity results are not applicable to scheduling a PFB due to its two major characteristics: job-independent processing times and batching machines. The former makes the problem easier, because it restricts the set of possible processing times, while the latter makes it harder, since scheduling without batching is a special case of batching with upper batch capacity one. It is not a priori clear which effects occur due to the combination of both characteristics.

A survey about scheduling a proportionate flow shop without batching is given in [PSK13]. For most objective functions, this can be achieved in polynomial time.

Scheduling problems with batching are reviewed in [PK00] and [MS06]. Parallel batching has been studied by [LUMV92] in the context of semiconducter manufacturing. Another standard work dealing with algorithms for a single batching machine is the article [BGH$^+$98] by Brucker et al. Among other results, they show that scheduling a single batching machine to minimize the maximum lateness is strongly NP-hard, even if the batch capacity is equal to two and there are no release dates. By a mirroring

argument, this implies that minimizing the makespan with release dates is NP-hard, too. In contrast to our setting, these hardness results require job-dependent processing times.

Scheduling a (not necessarily proportionate) flow shop of batching machines is NP-hard for all standard scheduling objective functions, since it generalizes usual flow shop problems. Some mixed-integer programs have been proposed for this purpose, e.g. by Buscher and Shen [BS10].

Ikura and Gimple [IG86] consider the case of a single batching machine with consistent release and due dates and identical processing times. In their work, due dates are a hard requirement for feasibility. They propose a quadratic time algorithm which computes a feasible schedule with minimum makespan or detects that no feasible schedule exists. They also observe that a simple scheduling rule called *first-only-empty batching* (FOE) is optimal, if there are no due dates. This rule uses only full batches except for the first batch, which might be only partially filled, if the total number of jobs is not a multiple of the batch capacity. Closely related is the concept of *last-only-empty batching* (LOE), in which all batches are full except for possibly the last one.

Baptiste [Bap00] presents a polynomial time dynamic programming approach to schedule a single batching machine with release dates and identical processing times for a large variety of objective functions. For maximum lateness, Condotta et al. [CKS10] give an $\mathcal{O}(n^2 \log n)$ algorithm, which significantly improves the runtime of [Bap00].

Ahmadi et al. [AADT92] study a PFB consisting of two machines without release dates. Building upon the results of [IG86], they show that applying LOE batching on the first machine and FOE batching on the second machine is optimal for makespan minimization, resulting in an $\mathcal{O}(n)$ algorithm. They also present an $\mathcal{O}(n^3)$ dynamic program to minimize the sum of completion times.

Sung and Yoon [SY97] extend the results of [AADT92] by considering makespan minimization in a two-machine PFB with release dates. They use a dynamic program to find the optimal schedule for the first machine. FOE batching remains optimal on the second machine. In total they obtain an algorithm running in $\mathcal{O}(nb_1) \subseteq \mathcal{O}(n^2)$ time. The same algorithm can be used for three machines without release dates. The reason for this is that LOE batching on the first machine is optimal in this case, allowing to remove the first machine by introducing release dates on the second machine instead.

Sung and Kim [SK03] investigate minimization of maximum lateness, number of late jobs and total tardiness in a two-machine PFB without release dates. They present polynomial time dynamic programs for each of those objective functions.

To the best of our knowledge, the work of Sung, Kim and Yoon [SKY00] is the only work in literature considering a PFB with arbitrarily many machines. They propose a reduction procedure to decrease the size of an instance by eliminating dominated machines. Using this procedure, they develop heuristics to minimize the makespan and the sum of completion times in a PFB and conduct a computational study certifying

quality and efficiency of their heuristic approach. They leave it open to classify the computational complexity of scheduling a PFB.

## 1.4. Outline of the Thesis

After this introductory chapter, we give some preparatory definitions, notations, and results in Chapter 2. Chapter 3 deals with the question for which PFB variants we can limit our studies to so-called *permutation schedules* without losing optimality. Permutation schedules are schedules in which the jobs are processed in the same order on all machines.

The Chapters 4 to 7 deal with algorithms that find an exact solution to the problem of scheduling a PFB. In Chapter 4, we propose several mixed-integer programming formulations. A dynamic program that computes an optimal solution in a job-wise manner is introduced in Chapter 5. This approach shows that many PFB variants are solvable in polynomial time, if the number $m$ of machines is fixed. In contrast, a machine-wise approach based on efficient (i.e. Pareto-optimal) partial schedules is presented in Chapter 6. This yields a polynomial algorithm for scheduling a PFB with $n = 2$ jobs, as shown in Chapter 7. Other special cases are investigated in Chapter 7 as well, namely constant batch capacities, constant processing times, and the case of $m = 2$ machines without release dates to minimize the makespan.

In Chapter 8 we give a lower bound for the completion time of each job on each machine. We interpret this bound in three ways. Afterwards, it is used to investigate the approximation ratios of two simple scheduling rules.

Chapter 9 is concerned with an online version of a PFB. In this version, jobs arrive over time and are unknown before their release date. We present algorithms reaching the best possible competitive ratio for the cases of one or two machines.

Finally, Chapter 10 concludes this thesis. Several further research questions are pointed out.

The appendix consists of some additional proofs omitted from the main part, references, acknowledgments, as well as a declaration of authorship (“Selbständigkeitserklärung”).

# 2. Preliminaries

In this chapter we formally define the objects and problems we deal with in this thesis. We also discuss some fundamental complexity issues arising when scheduling a PFB.

## 2.1. Schedules and Feasibility

One of the most common ways to represent a schedule is to store start and completion times of each job on each machine. This also uniquely defines which jobs are batched together. Note that in our deterministic setting it suffices to store either start or completion times, because they differ exactly by the processing time on the corresponding machine. As we mainly work with the completion times, we give the following definition:

**Definition 2.1.**

(i) A *schedule* $S$ for a PFB is given by values $c_{ij}$, $i \in [m]$, $j \in [n]$, defining for each job $j$ its completion time on machine $i$.

(ii) A *partial schedule* up to machine $i$ is a schedule for the restricted flow shop instance consisting only of the machines 1 to $i$.

**Notation 2.2.** Denote the vector of the completion times of all jobs on machine $i$ by $c_i = (c_{ij})_{j\in[n]} \in \mathbb{R}^n$. In accordance with standard scheduling notation, we also write $C_j = c_{mj}$ for the completion time of job $j$ on the last machine. The vector of all job completion times on the last machine is denoted by $C = (C_j)_{j\in[n]} \in \mathbb{R}^n$. Since we often deal with multiple schedules, we write $c_{ij}(S)$, $c_i(S)$, $C_j(S)$, and $C(S)$ for the completion times in schedule $S$.

**Definition 2.3.** A schedule is called *feasible*, if it satisfies the following conditions:

(i) No job is started before it is released, i.e. $c_{1j} \geq r_j + p_1$ for all $j \in [n]$.

(ii) No job is started on a machine before it has been finished on the previous machine, i.e. $c_{(i+1)j} \geq c_{ij} + p_{i+1}$ for all $i \in [m-1]$, $j \in [n]$.

(iii) On each machine $i$ two jobs are either in the same batch or their completion times on machine $i$ differ at least by the processing time $p_i$, i.e. $|c_{ij} - c_{ij'}| = 0$ or $|c_{ij} - c_{ij'}| \geq p_i$ for all $i \in [m]$, $j, j' \in [n]$.

(iv) A batch on machine $i$ consists of at most $b_i$ jobs, i.e. $|\{j \in [n] \mid c_{ij} = c\}| \leq b_i$ for all $i \in [m]$, $c \in \mathbb{R}$.

## 2.2. Objective Functions

In this section we define the objective functions we consider in this thesis. Most definitions coincide with those in [Bru07]. However, in contrast to [Bru07], we distinguish the *completion time* and the *flow time* of a job. We start with defining the term *objective function*.

**Definition 2.4.** An *objective function* is a function $f\colon \mathbb{R}^n \to \mathbb{R}$ mapping a vector of job completion times $C = (C_j)_{j\in[n]} \in \mathbb{R}^n$ to a real number $f(C)$.

Now, the problem of scheduling a PFB can be formulated as follows: Given $n, m, b_i, p_i, r_j$ for $i \in [m], j \in [n]$ and an objective function $f$, compute a schedule $S$ that minimizes $f(C(S))$.

All widely used objective functions belong to one of the following two types.

**Definition 2.5.**

- A *sum objective function* is given by a sum of job-wise objective functions, i.e. $f(C) = \sum_{j=1}^{n} f_j(C_j)$. It is abbreviated by $\sum f_j$.
- A *bottleneck objective function* is given as the maximum of job-wise objective functions, i.e. $f(C) = \max_{j\in[n]} f_j(C_j)$. It is abbreviated by $f_{\max}$.

Some job-wise objective functions involve a *due date* $d_j \in \mathbb{R}_{\geq 0}$ or a *weight* $w_j \in \mathbb{R}_{\geq 0}$ associated with each job $j$. These parameters are part of the instance, if they exist.

**Definition 2.6.** We define the following job-wise objective functions:

- *completion time* $f_j(C_j) = \mathbf{C_j}$,
- *flow time* $f_j(C_j) = \mathbf{F_j} = C_j - r_j$,
- *lateness* $f_j(C_j) = \mathbf{L_j} = C_j - d_j$,
- *tardiness* $f_j(C_j) = \mathbf{T_j} = \max\{0, C_j - d_j\}$,
- *unit penalty* $f_j(C_j) = \mathbf{U_j} = \begin{cases} 0, & \text{if } C_j \leq d_j \\ 1, & \text{otherwise.} \end{cases}$

In this thesis we focus on the following widely used objective functions: $C_{\max}$, $\sum C_j$, $\sum w_j C_j$, $F_{\max}$, $L_{\max}$, $\sum T_j$, $\sum w_j T_j$, $\sum U_j$ and $\sum w_j U_j$. In the context of approximation algorithms we also consider $\sum F_j$ and $\sum w_j F_j$, which differ only by a constant from $\sum C_j$ and $\sum w_j C_j$, respectively.

As in [Bru07], we define:

**Definition 2.7.** An objective function is called *regular*, if it is nondecreasing with respect to the values $C_j$.

Note that all objective functions defined in this section are regular.

## 2.3. Batchings and Batch-Active Schedules

**Definition 2.8.** A *batch* is a set of jobs $B_l \subseteq [n]$ that are processed together on a certain machine. A *batching* $\mathcal{B} = (B_1, \ldots, B_k)$ is a tuple of batches such that each $j \in [n]$ is contained in exactly one batch. The order in the tuple determines the order in which the batches are processed on the corresponding machine. A batching is called *feasible* for machine $i$, if $|B_l| < b_i$ for all $l \in [k]$.

Given a feasible schedule, one can uniquely determine a naturally corresponding feasible batching $\mathcal{B}^{(i)}$ for each $i \in [m]$. Suppose the set $\{c_{ij} \mid j \in [n]\}$ consists of the numbers $t_1 < t_2 < \cdots < t_k$, $k \leq n$. Define $B_l^{(i)} = \{j \in [n] \mid c_{ij} = t_l\}$. Then, for fixed $i$, each job is contained in exactly one batch $B_l^{(i)}$. Moreover, two jobs are contained in the same set $B_l^{(i)}$, if and only if they have the same completion time on machine $i$. Also, the batches are ordered by their completion times. Finally, point (iv) of Definition 2.3 ensures that $\mathcal{B}^{(i)}$ is in fact feasible for machine $i$.

Conversely, feasible batchings $\mathcal{B}^{(i)}$, $i \in [m]$, do not uniquely determine a corresponding schedule, because it is not specified when to start the batches. However, if we additionally require that there is no unnecessary waiting time, then the corresponding schedule becomes unique. This requirement is captured by the next definition.

**Definition 2.9.** A schedule is called *batch-active*, if no batch can be started earlier without violating feasibility of the schedule and without changing the order of batches.

This means that in a batch-active schedule, each batch on each machine is is started as soon as all jobs of the batch have arrived and the machine has finished the previous batch. Hence, given feasible batchings $\mathcal{B}^{(i)}$, $i \in [m]$, a corresponding batch-active schedule can be uniquely calculated in the following way. Proceed machine-wise, starting with machine 1, and for each machine batch-wise. For a batch $B_l^{(i)}$ on machine $i$ let $t_1 = \max\{c_{(i-1)j} \mid j \in B_l^{(i)}\}$ be the largest completion time of a job in the batch on machine $i-1$. We use $c_{0j} = r_j$ as starting values. Additionally, if $l > 1$, let $t_2$ be the completion

time of the previous batch $B_{l-1}^{(i)}$ on machine $i$. If $l = 1$, set $t_2 = -\infty$. Then the completion times of all jobs in $B_l^{(i)}$ are given by $\max\{t_1, t_2\} + p_i$.

The same construction can be used to make an arbitrary schedule batch-active. This keeps all batchings and eliminates unnecessary waiting times. The values $c_{ij}$, $i \in [m]$, $j \in [n]$, are not increased by doing this. Thus, the value of any regular objective function does not increase either. This observation is summarized by the following lemma.

**Lemma 2.10.** *A schedule for a PFB can always be transformed into a batch-active schedule such that any regular objective function is not increased. This transformation does not change the order in which the jobs are processed on the machines.*

As a consequence, we obtain the following lemma, which is not true for arbitrary objective functions.

**Lemma 2.11.** *For a PFB with a regular objective function, there always exists an optimal schedule.*

*Proof.* Note that there are only finitely many batch-active schedules. For a regular objective function, Lemma 2.10 implies that one of them is optimal. □

Another lemma about batch-active schedules will be helpful in further considerations:

**Lemma 2.12.** *In batch-active schedules all completion times are bounded by* $M := \max_{j\in[n]} r_j + n \sum_{i=1}^{m} p_i$.

*Proof.* Consider a batch-active schedule. After all jobs have been released and before all jobs have been completed there cannot be a period in which all machines are idle. Moreover, each machine $i$ processes at most $n$ batches, which needs at most $np_i$ time. Therefore, all jobs are completed at latest at time $M$. □

## 2.4. Complexity Issues and High-Multiplicity Problems

This section briefly explains how we analyze the complexity of scheduling different PFB variants. In some cases this leads to so-called high-multiplicity problems. For standard definitions and results about complexity theory we refer to [GJ79].

In order to study the complexity of a scheduling problem, three different versions are usually considered.

**Recognition (or decision) version** Given an instance of a scheduling problem and a real number $k \in \mathbb{R}$, is there a schedule with objective function less or equal to $k$?

**Evaluation version** Given an instance of a scheduling problem, compute the minimal objective value of a feasible schedule.

**Optimization version** Given an instance of a scheduling problem, compute a feasible schedule minimizing the objective function.

The most fundamental question for each variant is whether it can be solved in polynomial time. Commonly, *polynomial* means that the runtime is bounded by a polynomial in the encoding length of the instance. In the case of a PFB, job-independent processing times impose an interesting challenge. If the instance does not contain job-specific parameters (e.g. when minimizing makespan without release dates), only the number of jobs has to be encoded. In a binary encoding this yields an encoding length proportional to the logarithm of $n$ instead of $n$ itself. A scheduling problem where such a compact encoding of the input is possible is called *high-multiplicity scheduling problem*. This term has been introduced by Hochbaum and Shamir [HS90, HS91]. Brauner et al. [BCGvdK05] propose a complexity framework for the analysis of high-multiplicity scheduling problems. Moreover, high-multiplicity scheduling is extensively discussed in the dissertation of Grigoriev [Gri03].

The short encoding length has some interesting effects. First of all, the encoding length of an explicit schedule storing start or completion times of each job on each machine contains the factor $n$. This is not polynomial in the encoding length of the input any more. Therefore, the optimization version cannot be solved in polynomial time, if we require the output of such an explicit schedule to be returned. This does not hold for the evaluation and decision versions. However, the short input encoding makes solving these versions in polynomial time more challenging.

Usually, the decision version of a scheduling problem is contained in the complexity class NP, since an optimal schedule serves as a polynomial certificate for a yes-instance. This is no longer true for high-multiplicity problems, since for these problems explicit schedules do not have polynomial size, and hence, cannot be used as certificate. Therefore, it is not a priori clear whether the decision version of a high-multiplicity problem is contained in NP.

Although the high-multiplicity property gives rise to interesting theoretical questions, we conclude this section by remarking that this property does not often matter in practice. Even if there are no job-specific parameters defining the mathematical scheduling problem, jobs are usually still encoded separately, e.g. because they have a name or at least an ID. Also they have to be processed physically. Therefore, from a practical point of view, an algorithm polynomial in $n$ instead of $\log n$ may still be fast enough.

## 2.5. Reductions between Objective Functions

Our next goal is to establish some relations between the objective functions defined in Section 2.2. Polynomial reductions are the key tool for this.

**Definition 2.13.** We say that an objective function $f$ *reduces* to another objective function $g$, if there is a polynomial reduction between the decision versions of the optimization problems to minimize $f$ and $g$, respectively.

Observe that the notion of a reduction between objective functions defined above is transitive.

In [Bru07, Section 3.3] one can find an overview of reductions that are common for all scheduling problems. Additionally, note that $F_{\max}$ reduces to $L_{\max}$ by setting $d_j = r_j$ for all $j \in [n]$.

However, observe that in the case of a PFB, some of the standard reductions are not polynomial, if high-multiplicity problems are involved. For example, minimizing $C_{\max}$ without release dates is a high-multiplicity problem, while minimizing $L_{\max}$ involves due dates and is therefore not a high-multiplicity problem. Hence, the usual reduction from $C_{\max}$ to $L_{\max}$ is not polynomial in this special case. For the same reason, the reductions from $\sum C_j$ to $\sum T_j$ and $\sum w_j C_j$ are not polynomial for a PFB without release dates.

Now we establish a reduction, which holds specifically for a PFB.

**Theorem 2.14.** *For a PFB, $C_{\max}$ reduces to $\sum C_j$.*

*Proof.* Consider an instance $I$ of a PFB with makespan as the objective function. Construct a modified instance $I'$, where another machine $m+1$ is added at the end of the process. Set $b_{m+1} = n$ and $p_{m+1} = nM$, where $M = \max_{j\in[n]} r_j + n\sum_{i=1}^{m} p_i$ is defined for $I$ as in Lemma 2.12.

We show that in a solution of $I'$ which minimizes $\sum C_j$, all $n$ jobs are batched together on machine $m+1$. Suppose this is not the case. Then there is a job $j$ with $C_j > 2nM$ and $C_{j'} > nM$ for all $j' \neq j$. Hence, $\sum_{j=1}^{n} C_j > (n+1)nM$. On the other hand, if we batch all jobs together on the last machine, we can obtain a smaller sum of completion times as follows: By Lemma 2.12, we can finish all jobs on the first $m$ machines at latest at time $M$. At this time, we can start the full batch on the last machine and obtain $\sum_{j=1}^{n} C_j = n(M + p_{m+1}) = n(M + nM) = (n+1)nM$. This shows that any schedule for $I'$ minimizing $\sum C_j$ uses a full batch on machine $m+1$.

Thus, in any schedule for $I'$ minimizing $\sum C_j$, all jobs have to wait for the last job before being processed on machine $m+1$. Hence, the completion time of any job in an optimal schedule for $I'$ is the optimal makespan in $I$ plus the processing time of machine $m+1$. Therefore, there exists a schedule with $C_{\max} \leq k$ in $I$, if and only if there exists a schedule with $\sum C_j \leq n(k + nM)$ in $I'$. Finally, note that the encoding length of

$p_{m+1} = nM$ is still polynomial in the encoding length of the original input, even in the high-multiplicity case. □

Figure 2.1 shows an extended version of [Bru07, Figure 3.5] for scheduling a PFB including the results of this section. An arrow from $f$ to $g$ symbolizes a reduction from $f$ to $g$. Note that without release dates minimizing $C_{\max}$ and $\sum C_j$ are high-multiplicity problems. Therefore, no arrow leaves the component formed by these two objectives.

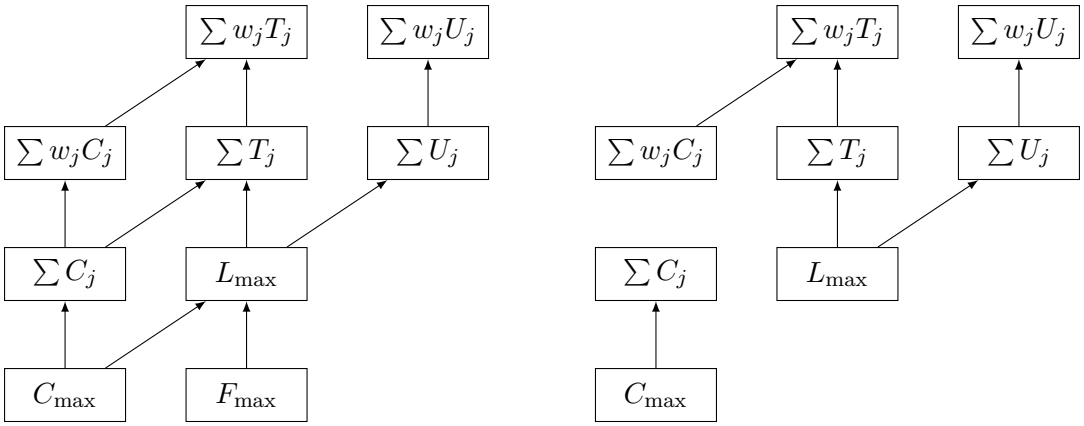


Figure 2.1.: Reductions between objective functions for scheduling a PFB with release dates (left) and without release dates (right).

# 3. Permutation Schedules

In the literature on flow shop problems, a permutation schedule is a schedule in which the order of jobs on a machine is fixed throughout the whole flow shop. This restriction makes scheduling a PFB easier, since one only needs to decide how to batch the jobs, and not in which order to process the batches. This chapter deals with the question for which PFB variants we can restrict to permutation schedules without losing optimality.

## 3.1. Definitions

In this section we give several definitions needed to investigate permutation schedules in the context of a PFB.

**Definition 3.1.** A *permutation* of $[n]$ is a bijective map from $[n]$ to itself. We denote the set of all permutations of $[n]$ by $\mathrm{Sym}(n)$. For a permutation $\sigma \in \mathrm{Sym}(n)$ and a vector $v = (v_j)_{j\in[n]} \in \mathbb{R}^n$ we define the vector $\sigma(v)$ componentwise by $(\sigma(v))_j = v_{\sigma^{-1}(j)}$ for $j \in [n]$. A vector $w \in \mathbb{R}^n$ is a *permutation* of $v$, if there exists a $\sigma \in \mathrm{Sym}(n)$ such that $w = \sigma(v)$.

Note that in our definition of $\sigma(v)$ we use $\sigma^{-1}$ in the indices. This makes sense for the following reason. Suppose $\sigma$ maps $j_1$ to $j_2$. One might expect that the element at position $j_1$ in $v$ will be at position $j_2$ in $\sigma(v)$. This means that we need $(\sigma(v))_{j_2} = v_{j_1} = v_{\sigma^{-1}(j_2)}$, justifying our definition above.

**Definition 3.2.** A vector $v \in \mathbb{R}^n$ is *ordered nondecreasingly*, if for all $i \leq j$ it holds that $v_i \leq v_j$. A permutation $\sigma \in \mathrm{Sym}(n)$ *orders* $v \in \mathbb{R}^n$ *nondecreasingly*, if $\sigma(v)$ is ordered nondecreasingly.

**Definition 3.3.** A schedule $S$ for a PFB is called a *permutation schedule*, if there exists a $\sigma \in \mathrm{Sym}(n)$ such that all vectors $c_i(S)$ are ordered nondecreasingly by $\sigma$. In other words, the jobs are processed in the fixed order $\sigma^{-1}(1), \sigma^{-1}(2), \ldots, \sigma^{-1}(n)$ on all machines. In this case we say that $S$ *is ordered by* $\sigma$.

**Definition 3.4.** A permutation $\sigma \in \mathrm{Sym}(n)$ is *optimal*, if there exists an optimal schedule which is a permutation schedule ordered by $\sigma$.

## 3.2. Assignment Problems and Monge Matrices

It turns out that finding an optimal job permutation is closely related to the so-called assignment problem as well as to one of its variants, the bottleneck assignment problem. Following the lines of [BDM09], we define these two problems, present special cases in which they have a trivial solution, and apply this to derive helpful lemmas for our investigation of permutation schedules.

**Definition 3.5.** Let a cost matrix $A = (a_{ij})_{i,j\in[n]}$ be given.

(i) The *assignment problem* is to find a permutation $\sigma \in \mathrm{Sym}(n)$ such that $\sum_{i\in[n]} a_{i\sigma(i)}$ is minimized.

(ii) The *bottleneck assignment problem* is to find a permutation $\sigma \in \mathrm{Sym}(n)$ such that $\max_{i\in[n]} a_{i\sigma(i)}$ is minimized.

The (bottleneck) assignment problem has a trivial solution, if the cost matrix $A$ has the (bottleneck) Monge property, compare [BDM09, Section 5.2 and Subsection 6.2.6]. This is captured by the next definition and theorem.

**Definition 3.6.** Let a matrix $A = (a_{ij})_{i,j\in[n]}$ be given.

(i) $A$ has the *Monge property* (or is a *Monge matrix*), if for $1 \le i < k \le n$ and $1 \le j < l \le n$, it holds that

$$a_{ij} + a_{kl} \le a_{il} + a_{kj}.$$

(ii) $A$ has the *bottleneck Monge property* (or is a *bottleneck Monge matrix*), if for $1 \le i < k \le n$ and $1 \le j < l \le n$, it holds that

$$\max\{a_{ij}, a_{kl}\} \le \max\{a_{il}, a_{kj}\}.$$

**Theorem 3.7.** *If the cost matrix $A$ has the (bottleneck) Monge property, then the identity $\mathrm{id}_{[n]}$ is an optimal solution for the (bottleneck) assignment problem, respectively.*

*Proof.* See [BDM09, Proposition 5.7 and Proposition 6.14]. □

Burkard et al. [BDM09, Section 5.2] mention several sufficient conditions for a matrix to be a Monge matrix. We generalize them by the following lemma. Its assumptions are motivated by [Bru07, Subsection 4.6.2]. They are met by most of the standard scheduling objective functions, as we will see later.

**Lemma 3.8.** *Let $v \in \mathbb{R}^n$ be ordered nondecreasingly and let $f_i \colon \mathbb{R} \to \mathbb{R}$, $i \in [n]$, be functions such that, for $i < j$, the function $f_i - f_j$ is nondecreasing. Then the matrix $A$ with entries $a_{ij} = f_i(v_j)$ is a Monge matrix.*

*Proof.* For $1 \leq i < k \leq n$ and $1 \leq j < l \leq n$, it holds that

$$a_{ij} + a_{kl} - (a_{il} + a_{kj}) = f_i(v_j) - f_k(v_j) - (f_i(v_l) - f_k(v_l)) \leq 0,$$

since $f_i - f_k$ is nondecreasing and $v_j \leq v_l$. Hence, $A$ has the Monge property. □

Using these results, we obtain a lemma that will help to prove optimality of permutation schedules in certain situations involving (weighted) sum objectives. It generalizes the famous *rearrangement inequality*, compare [BDM09, Proposition 5.8]. This inequality by Hardy, Littlewood and Pólya [HLP52, Section 10.2] states that for two nondecreasing vectors $u, v \in \mathbb{R}^n$ the sum $\sum_{i\in[n]} u_i v_{\sigma(i)}$ is maximized for $\sigma = \mathrm{id}_{[n]}$.

**Lemma 3.9.** *Let $v \in \mathbb{R}^n$ be ordered nondecreasingly and let $f_i \colon \mathbb{R} \to \mathbb{R}$, $i \in [n]$, be functions such that, for $i < j$, the function $f_i - f_j$ is nondecreasing. Then $\sigma = \mathrm{id}_{[n]}$ minimizes the expression*

$$\sum_{i\in[n]} f_i(v_{\sigma(i)})$$

*among all $\sigma \in \mathrm{Sym}(n)$.*

*Proof.* Use Theorem 3.7 and Lemma 3.8. □

Notice that the rearrangement inequality mentioned above follows by setting $f_i(x) = -u_i x$.

We also need a similar statement in the bottleneck case. We use a sufficient condition for a matrix to have the bottleneck Monge property taken from [BDM09, Subsection 6.2.6].

**Lemma 3.10.** *Let $u, v \in \mathbb{R}^n$ be ordered nondecreasingly. Then the matrix $A$ with entries $a_{ij} = v_i - u_j$ is a bottleneck Monge matrix.*

*Proof.* Let $1 \leq i < k \leq n$ and $1 \leq j < l \leq n$. Then it holds that

$$\max\{a_{il}, a_{kj}\} \geq a_{kj} = v_k - u_j \geq \max\{v_i - u_j, v_k - u_l\} = \max\{a_{ij}, a_{kl}\}.$$

Hence, $A$ has the bottleneck Monge property. □

Applying this we obtain a lemma that will help to prove optimality of permutation schedules in certain situations involving bottleneck objectives. It states that, if two vectors are ordered in the same way, the minimum entry of the difference of two vectors is maximized, and the maximum entry is minimized.

**Lemma 3.11.** *Let $u, v \in \mathbb{R}^n$ be ordered nondecreasingly and let $\sigma, \pi \in \mathrm{Sym}(n)$. Then it holds that*

*(i)* $\min_{j\in[n]}(v_j - u_j) \geq \min_{j\in[n]}(v_{\pi(j)} - u_{\sigma(j)})$ *and*

*(ii)* $\max_{j\in[n]}(v_j - u_j) \leq \max_{j\in[n]}(v_{\pi(j)} - u_{\sigma(j)})$.

*Proof.* We first prove (ii). By altering $\sigma$ accordingly, we can assume without loss of generality that $\pi = \mathrm{id}_{[n]}$. Then (ii) follows with Lemma 3.10 and Theorem 3.7. Part (i) follows from (ii) by exchanging roles of $u$ and $v$:

$$\begin{aligned}\min_{j\in[n]}(v_j - u_j) &= -\max_{j\in[n]}(u_j - v_j)\\ &\geq -\max_{j\in[n]}(u_{\sigma(j)} - v_{\pi(j)})\\ &= \min_{j\in[n]}(v_{\pi(j)} - u_{\sigma(j)}).\end{aligned}$$

□

An alternative proof of the previous Lemma, which does not use Monge matrices, but applies the pigeonhole principle instead, can be found in the appendix, see Section A.1.

## 3.3. Optimality of Permutation Schedules

In [SKY00, Lemma 1], Sung et al. claim that for minimizing a regular objective function it suffices to consider permutation schedules. However, as we will see, this is only true because their instances do not involve release dates. Moreover, even if there exists an optimal permutation schedule, it might be difficult to determine a corresponding optimal permutation. Therefore, this subsection aims to clarify under which conditions optimal permutation schedules exist and whether it is easy to determine an optimal permutation.

**Lemma 3.12.** *Let $S$ be a feasible schedule for a PFB. Let $\sigma \in \mathrm{Sym}(n)$ be a permutation such that the vector of release dates $r = (r_j)_{j\in[n]}$ is ordered nondecreasingly by $\sigma$. Then there exists a feasible schedule $S'$ with the following two properties:*

*(a) $S'$ is a permutation schedule ordered by $\sigma$.*

*(b) For each $i \in [m]$, $c_i(S')$ is a permutation of $c_i(S)$.*

*Proof.* Without loss of generality, assume that $\sigma$ is the identity function $\mathrm{id}_{[n]}$ by renaming the jobs. The proof proceeds as follows: We first define completion times $c_{ij}(S')$, $i \in [m]$, $j \in [n]$, such that (a) and (b) are satisfied. Afterwards we show that the schedule $S'$ defined by these completion times is indeed a feasible schedule.

For each machine $i$ let $\pi_i$ be a permutation that orders $c_i(S)$ nondecreasingly. Then we define

$$c_{ij}(S') = c_{i\pi_i^{-1}(j)}(S)$$

for all $i \in [m]$, $j \in [n]$. This is equivalent to $c_i(S') = \pi_i(c_i(S))$ for all $i \in [m]$. By construction, a schedule $S'$ with those completion times satisfies (a) and (b). It remains to show feasibility.

By feasibility of $S$ it follows that points (iii) and (iv) of Definition 2.3 are satisfied for $S'$ as well.

We prove (i) and (ii) simultaneously by setting $c_{0j} = r_j$ for all $j \in [n]$ and $\pi_0 = \sigma = \mathrm{id}_{[n]}$. Fix $i \in [m]$. Then we have to show that for each $j \in [n]$ it holds that

$$c_{ij}(S') \geq c_{(i-1)j}(S') + p_i. \tag{3.1}$$

Using that $c_i(S')$ and $c_{i-1}(S')$ are ordered nondecreasingly and that they are permutations of $c_i(S)$ and $c_{i-1}(S)$, respectively, Lemma 3.11 implies that

$$\min_{j\in[n]} \left(c_{ij}(S') - c_{(i-1)j}(S')\right) \geq \min_{j\in[n]} \left(c_{ij}(S) - c_{(i-1)j}(S)\right).$$

By feasibility of $S$ it follows that the right-hand side is greater or equal to $p_i$. This shows (3.1). Hence, $S'$ is feasible. □

The previous lemma can be applied to derive optimality of permutation schedules in various settings.

**Theorem 3.13.** *For a PFB with a regular objective function and without release dates, there is always an optimal permutation schedule.*

*Proof.* Let $S$ be an optimal schedule, which exists due to Lemma 2.11. Let $\sigma$ be a permutation that orders $C(S)$ nondecreasingly. Since all release dates are zero, it follows that $\sigma$ orders $r$ nondecreasingly as well. Hence, we can apply Lemma 3.12 to obtain a permutation schedule $S'$ that satisfies $C(S') = C(S)$. In particular, $S$ and $S'$ have the same objective value. □

Note that the previous theorem is not constructive: In order to obtain the optimal permutation we started with an optimal schedule. The next theorem proves the existence of optimal permutation schedules for arbitrary release dates, if the objective function fulfills some symmetry property.

**Theorem 3.14.** *Suppose that the objective function $f\colon \mathbb{R}^n \to \mathbb{R}$ is regular and satisfies $f(C) = f(\sigma(C))$ for all $\sigma \in \mathrm{Sym}(n)$ and $C \in \mathbb{R}^n$. Then any $\sigma \in \mathrm{Sym}(n)$ that orders the release dates $r_j$, $j \in [n]$, nondecreasingly is optimal.*

*Proof.* Again, let $S$ be an optimal schedule, which exists due to Lemma 2.11. Let $S'$ be the corresponding schedule ordered by $\sigma$ as in Lemma 3.12. By our assumption on the objective function it follows that $S$ and $S'$ have the same objective value. □

Observe that Theorem 3.14 explicitly describes how an optimal permutation can be found. It suffices to sort the jobs by release dates. The next two theorems prove optimality of permutation schedules in other specific settings. They will also be helpful in deriving optimal permutations for many objective functions later.

**Theorem 3.15.** *If a permutation $\sigma \in \mathrm{Sym}(n)$ orders due dates $d_j$, negative weights $-w_j$, and release dates $r_j$ nondecreasingly, then it is optimal for minimizing $\sum w_j T_j$.*

*Proof.* By renumbering the jobs, we can assume without loss of generality that $\sigma = \mathrm{id}_{[n]}$. Let $f_j(x) = w_j \max\{0, x - d_j\}$ be the weighted tardiness of job $j$ if it completes at time $x$. We show that for $i < j$ the function $f_i - f_j$ is nondecreasing, which allows us to apply Lemma 3.9. Fix some values $x < y$. For all $j \in [n]$, let $\widetilde{d}_j = \max\{x, \min\{d_j, y\}\}$ be a modified due date such that it lies within the interval $[x, y]$. Let $1 \leq i < j \leq n$. Note that $d_i \leq d_j$ implies $\widetilde{d}_i \leq \widetilde{d}_j$. We obtain

$$\begin{aligned} f_i(y) - f_j(y) - (f_i(x) - f_j(x)) &= f_i(y) - f_i(x) - (f_j(y) - f_j(x)) \\ &= w_i(y - \widetilde{d}_i) - w_j(y - \widetilde{d}_j) \\ &\geq 0, \end{aligned}$$

since $w_i \geq w_j \geq 0$ and $y - \widetilde{d}_i \geq y - \widetilde{d}_j$. Hence, the functions $f_j$, $j \in [n]$, satisfy the assumption of Lemma 3.9.

Let $S$ be a schedule minimizing $\sum w_j T_j$. Let $S'$ be the corresponding schedule ordered by $\sigma = \mathrm{id}_{[n]}$ as in Lemma 3.12. As $C(S')$ is ordered nondecreasingly and is a permutation of $C(S)$, Lemma 3.9 implies that

$$\sum_{j=1}^{n} w_j T_j(S') = \sum_{j=1}^{n} f_j(C_j(S')) \leq \sum_{j=1}^{n} f_j(C_j(S)) = \sum_{j=1}^{n} w_j T_j(S).$$

Since $S$ is optimal, $S'$ must be optimal, too. □

**Theorem 3.16.** *If a permutation $\sigma \in \mathrm{Sym}(n)$ orders due dates $d_j$ and release dates $r_j$ nondecreasingly, then it is optimal for minimizing $L_{\max}$.*

*Proof.* By renumbering the jobs, we can assume without loss of generality that $\sigma = \mathrm{id}_{[n]}$. Let $S$ be a schedule minimizing $L_{\max}$. Let $S'$ be the corresponding schedule ordered by $\sigma = \mathrm{id}_{[n]}$ as in Lemma 3.12. Using Lemma 3.11, the fact that $C(S')$ and $d_j$ are ordered nondecreasingly, and that $C(S')$ is a permutation of $C(S)$, we obtain

$$L_{\max}(S) = \max_{j \in [n]} (C_j(S) - d_j) \geq \max_{j \in [n]} (C_j(S') - d_j) = L_{\max}(S').$$

Since $S$ is optimal, $S'$ must be optimal, too. □

**Remark 3.17.** Theorem 3.16 can also be proven by Theorem 3.15 and the way how $L_{\max}$ reduces to $\sum w_j T_j$.

Finally, the next two corollaries show cases in which an optimal permutation for different objective functions can easily be found by sorting the jobs with respect to input parameters like release dates, weights, or due dates.

**Corollary 3.18.** *Any $\sigma \in \mathrm{Sym}(n)$ that orders $r_j$ nondecreasingly is optimal for $F_{\max}$, $C_{\max}$ and $\sum C_j$.*

*Proof.* The claim for $F_{\max}$ and $C_{\max}$ follows from Theorem 3.16 by setting $d_j = r_j$ and $d_j = 0$, respectively. The claim for $\sum C_j$ follows from Theorem 3.15 by setting $d_j = 0$ and $w_j = 1$. □

**Corollary 3.19.** *Suppose all jobs are released simultaneously.*

*(i) Any $\sigma \in \mathrm{Sym}(n)$ that orders $-w_j$ nondecreasingly is optimal for $\sum w_j C_j$.*

*(ii) Any $\sigma \in \mathrm{Sym}(n)$ that orders $d_j$ nondecreasingly is optimal for $\sum T_j$ and $L_{\max}$.*

*(iii) Any $\sigma \in \mathrm{Sym}(n)$ is optimal for $\sum C_j$ and $C_{\max}$.*

*Proof.*

(i) This follows from Theorem 3.15 by setting $d_j = r_j = 0$.

(ii) The claim for $\sum T_j$ follows from Theorem 3.15 by setting $w_j = 1$ and $r_j = 0$. The claim for $L_{\max}$ follows from Theorem 3.16 by setting $r_j = 0$.

(iii) This follows from Theorem 3.14. Alternatively, the claim for $\sum C_j$ follows from Theorem 3.15 by setting $d_j = r_j = 0$ and $w_j = 1$, and the claim for $C_{\max}$ follows from Theorem 3.16 by setting $d_j = r_j = 0$. □

Now we present an instance for which no permutation schedule is optimal. The example can be applied on various objective functions.

**Example 3.20.** Consider an instance given by $m = 3$, $n = 2$, $b_1 = p_1 = b_3 = p_3 = 1$, $b_2 = p_2 = 2$, $r_1 = 0$, $r_2 = 1$, $d_1 = 6$ and $d_2 = 5$. First, we show that no optimal permutation schedule exists for $L_{\max}$. Figure 3.1 shows a feasible schedule which is not a permutation schedule. The gray box indicates that the second job has not been released yet.

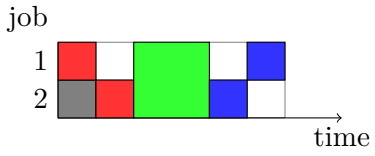


Figure 3.1.: A non-permutation schedule for the instance of Example 3.20.

In contrast, Figure 3.2 shows all batch-active permutation schedules, since the only two decisions are which of the two jobs is processed first and whether to batch on the second machine or not. Using Lemma 2.10, one of them must be optimal, if there exists an optimal permutation schedule.

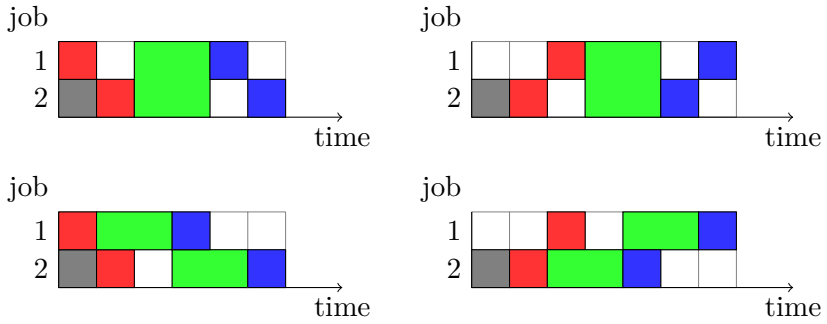


Figure 3.2.: All possible permutation schedules without unnecessary idle time for the instance of Example 3.20.

However, the schedule in Figure 3.1 achieves an objective value of zero, while all schedules in Figure 3.2 have a positive objective value. Hence, there is no optimal permutation schedule for this instance. The same example shows that there is no optimal permutation schedule for $\sum T_j$, $\sum w_j T_j$, $\sum U_j$ and $\sum w_j U_j$.

Choosing weights $w_1 = 1$, $w_2 = 3$ leads to an instance where no permutation schedule is optimal for $\sum w_j C_j$, since the schedule in Figure 3.1 achieves an objective value of $6 \cdot 1 + 5 \cdot 3 = 21$, while all schedules in Figure 3.2 have an objective value of at least 22.

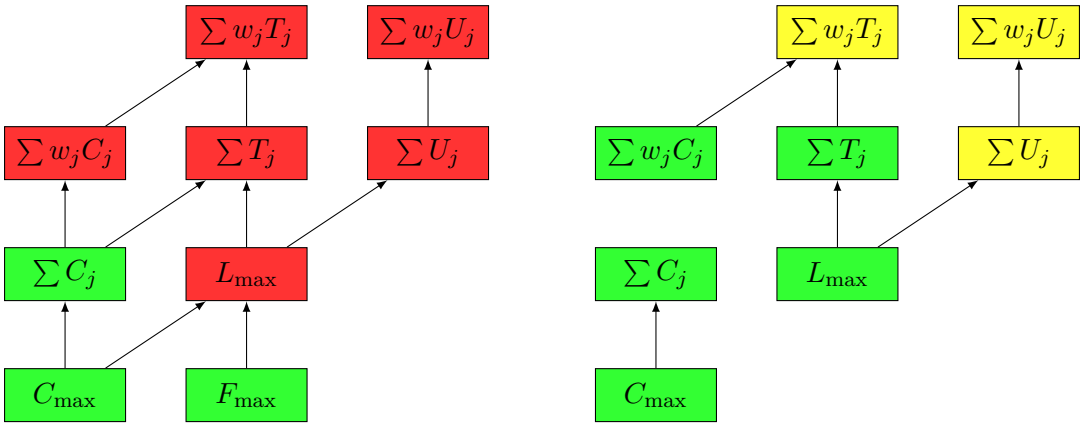


Figure 3.3.: Existence of optimal permutation schedules with release dates (left) and without release dates (right).

Figure 3.3 summarizes the results of this section. All objectives marked in green allow for

determining an optimal permutation by sorting the jobs with respect to release dates, due dates, or weights, as shown in Corollaries 3.18 and 3.19. If an objective is marked in red, this means that there are examples where there does not exist an optimal permutation schedule, as shown in Example 3.20. For the objectives marked in yellow, there exists an optimal permutation schedule due to Theorem 3.13, but it remains unclear how to determine an optimal permutation. We will see later (compare Corollary 5.6 in Chapter 5) that we can sometimes still make use of permutation schedules in these cases.

From now on, we will often concentrate on problem variants marked in green in Figure 3.3. In other words, we restrict ourselves to permutation schedules, and suppose that we know an optimal permutation. This allows us to assume that the jobs are indexed in a way such that they are processed in the order $1, 2, \ldots, n$ on all machines. For later reference, we summarize this in the following assumption.

**Assumption 3.21.** *There exists an optimal schedule in which the jobs are processed in the order* $1, 2, \ldots, n$ *on all machines.*

## 3.4. Batchings in Permutation Schedules

In this section we investigate how a fixed job permutation influences the choice of batchings. We make some general observations, define two specific batchings in the context of permutation schedules, and count how many possible batchings there are on each machine.

**Definition 3.22.** A batching is called *ordered*, if all its batches contain only subsequent numbers and the batches in the batching are ordered consistently with the job numbers contained in the batches.

**Observation 3.23.** A schedule is a permutation schedule ordered by $\mathrm{id}_{[n]}$, if and only if all corresponding batchings are ordered.

**Observation 3.24.** An ordered batching is uniquely defined by specifying the last job in each batch. Therefore, we identify an ordered batching $\mathcal{B}^{(i)}$ on machine $i \in [m]$ with a vector $z_i = (z_{ij})_{j \in [n-1]}$, where $z_{ij} = 1$, if and only if $j$ is the last job in its batch. Note that $j = n$ is definitely the last job in its batch. That is why $z_i$ contains only $n - 1$ components.

The following two specific ordered batchings play a special role when scheduling a PFB. They have been introduced in [IG86] first, and have been used by Sung et al. in [SY97] and [SKY00].

**Definition 3.25.**

(i) The *Last-Only-Empty (LOE) batching* on a machine $i$ is defined as the ordered batching which only uses full batches, except for the last batch, which contains

less jobs if $b_i$ does not divide $n$. Formally,

$$B_l^{(i)} = \begin{cases} \{(l-1)b_i+1, (l-1)b_i+2, \ldots, lb_i\}, & \text{if } 1 \leq l < \lceil \frac{n}{b_i} \rceil, \\ \{(l-1)b_i+1, (l-1)b_i+2, \ldots, n\}, & \text{if } l = \lceil \frac{n}{b_i} \rceil. \end{cases}$$

(ii) Similarly, the *First-Only-Empty (FOE) batching* on machine $i$ is defined as the ordered batching which only uses full batches, except for the first batch. Formally, for $k = \lceil \frac{n}{b_i} \rceil$,

$$B_l^{(i)} = \begin{cases} \{1, 2, \ldots, n-(k-1)b_i\}, & \text{if } l = 1, \\ \{n-(k-l+1)b_i+1, \ldots, n-(k-l)b_i\}, & \text{if } 1 < l \leq k. \end{cases}$$

Next, we count the number of feasible ordered batchings on a machine, depending on $n$ and the batch capacity $b$ of this machine. In [SY97], these numbers are given as coefficients of a generating function. We will see that they can be expressed in terms of the $b$-step Fibonacci numbers [NPW].

**Definition 3.26.** The *$b$-step Fibonacci numbers* are given by the following recursively defined sequence $(F_k^{(b)})_{k \in \mathbb{Z}}$:

$$F_k^{(b)} = 0, \text{ for } k \leq 0, \qquad F_1^{(b)} = 1, \qquad F_k^{(b)} = \sum_{l=1}^{b} F_{k-l}^{(b)}, \text{ for } k > 1.$$

**Observation 3.27.** For $b = 1$ it holds that $F_k^{(1)} = 1$ for all $k \geq 0$. For $b = 2$ we obtain the ordinary Fibonacci numbers. Moreover, one can inductively show that $2^{k-1} \geq F_k^{(b)} \geq F_k^{(b')}$ if $b \geq b'$ and $k > 1$. Since the Fibonacci numbers grow exponentially in $k$, this shows that the $b$-step Fibonacci numbers with $b \geq 2$ grow exponentially as well.

Now we establish the connection of the $b$-step Fibonacci numbers to the number of feasible ordered batchings on a machine.

**Theorem 3.28.** *On a machine with batch capacity $b$ there are exactly $F_{n+1}^{(b)}$ feasible ordered batchings of $n \in \mathbb{N}$ jobs. In particular, the number of feasible ordered batchings is exponential in $n$ for $b \geq 2$ and bounded by $2^{n-1}$.*

*Proof.* We use induction on $n$. If $n = 0$ or $n = 1$, then we only have one possibility and $F_1^{(b)} = F_2^{(b)} = 1$. Now let $n > 1$. Suppose the first batch has size $k \in [1, b']$ where $b' := \min\{n, b\}$. By induction, there are $F_{n-k+1}^{(b)}$ possibilities for batching the remaining $n-k$ jobs. Summing up over all possible values of $k$, the total number of feasible batches is given by

$$\sum_{k=1}^{b'} F_{n-k+1}^{(b)} = \sum_{k=1}^{b} F_{n-k+1}^{(b)} = F_{n+1}^{(b)},$$

where the first equality follows because $F_k^{(b)} = 0$ for $k \leq 0$. □

# 4. Mixed-Integer Programs

This chapter deals with mixed-integer programming (MIP) formulations for scheduling a PFB. Buscher and Shen [BS10] propose mixed-integer programs for flow shops with parallel batching. Their setting does not require job-independent processing times and differs in some other details from our PFB. We adapt it to our setting, which yields our first formulation. After this, we propose a second formulation using a different mechanism to ensure feasibility. Both formulations reflect a general PFB with a regular objective function without requiring Assumption 3.21.

Finally, we present a third formulation, which concentrates on permutation schedules, making use of Assumption 3.21. By fixing a job permutation, the number of binary variables can be reduced from $\mathcal{O}(mn^2)$ in each of the two general formulations to $\mathcal{O}(mn)$.

All formulations assume that the objective function is regular. Therefore, the constant $M = \max_{j\in[n]} r_j + n\sum_{i=1}^{m} p_i$ from Lemma 2.12 can be used as an upper bound for any completion time.

## 4.1. Adapting the Formulation of Buscher and Shen

Buscher and Shen [BS10] give two formulations for a non-proportionate flow shop of batching machines, i.e. processing times may be job-dependent. The processing time of a batch is defined as the maximum processing time of a job contained in it.

Their first formulation is tailored to so-called consistent batches, i.e. the batches formed on the first machine have to be used for the entire flow shop, while their order might still be changed between two machines. In contrast, their second formulation matches our situation of inconsistent batches, i.e. batches can be formed independently on all machines.

Moreover, in their setup, all batch capacities $b_i$ are equal to a global constant $S$. However, by modifying the corresponding constraint, one can adapt this to our situation with machine-dependent batch capacities.

Additionally, their formulation does not incorporate release dates. This issue can be fixed by adding corresponding constraints.

Finally, they use quadratic constraints in their formulation to enforce binary decisions. We replace them by typical *big-M* constraints, using the constant $M$ of Lemma 2.12.

In total, by slightly modifying the mixed-integer program for inconsistent batches given in [BS10, Section 2.2], one can obtain a general formulation for a PFB. It contains binary variables $\delta_{ijb}$, $i \in [m]$, $j \in [n]$, $b \in [n]$, which determine whether job $j$ is processed in the $b$-th batch on machine $i$. Moreover, it involves continuous variables $c_{ij}$, $i \in [m]$, $j \in [n]$ denoting the completion time of job $j$ on machine $i$, as well as $t_{ib}$, $i \in [m]$, $b \in [n]$, denoting the completion time of the $b$-th batch on machine $i$. Then, the constraints of our first formulation are given by:

$$\sum_{j=1}^{n} \delta_{ijb} \leq b_i, \qquad i \in [m],\ b \in [n], \tag{4.1a}$$

$$\sum_{b=1}^{n} \delta_{ijb} = 1, \qquad i \in [m],\ j \in [n], \tag{4.1b}$$

$$t_{i(b+1)} \geq t_{ib} + p_i, \qquad i \in [m],\ b \in [n-1], \tag{4.1c}$$

$$t_{1b} \geq r_j + p_1 - M(1 - \delta_{1jb}), \qquad j \in [n],\ b \in [n], \tag{4.1d}$$

$$t_{(i+1)b} \geq c_{ij} + p_{i+1} - M(1 - \delta_{(i+1)jb}), \qquad i \in [m-1],\ j \in [n],\ b \in [n], \tag{4.1e}$$

$$c_{ij} \geq t_{ib} - M(1 - \delta_{ijb}), \qquad i \in [m],\ j \in [n],\ b \in [n], \tag{4.1f}$$

$$\delta_{ijb} \in \{0, 1\}, \qquad i \in [m],\ j \in [n],\ b \in [n], \tag{4.1g}$$

$$c_{ij} \in \mathbb{R}, \qquad i \in [m],\ j \in [n], \tag{4.1h}$$

$$t_{ib} \in \mathbb{R}, \qquad i \in [m],\ b \in [n]. \tag{4.1i}$$

In this formulation, (4.1a) ensures that the batch sizes do not exceed the batch capacities. By (4.1b), each job has to be assigned to exactly one batch on each machine. Constraint (4.1c) guarantees that a machine has finished the previous batch when it starts a new one. Constraints (4.1d) and (4.1e) make sure that all jobs of a batch are available when the batch is started. Constraint (4.1f) relates job and batch completion times. Note that this formulation allows that the job completion time is larger than the corresponding batch completion time. However, this is fine, because it only introduces unnecessary delay and we restrict ourselves to regular objective functions. Finally, (4.1g) – (4.1i) define the ranges of the variables.

Additionally, our experiments have shown that adding the following constraints helps solvers to find lower bounds, although these constraints are already implied by the previous constraints:

$$c_{ij} \geq r_j + \sum_{i'=1}^{i} p_{i'}, \qquad i \in [m],\ j \in [n], \tag{4.1j}$$

$$t_{i1} \geq \min\{r_j \mid j \in [n]\} + \sum_{i'=1}^{i} p_{i'}, \quad i \in [m]. \tag{4.1k}$$

They basically reflect that at least the processing times $p_1, \ldots, p_i$ have to elapse, before a job or batch can be completed on the $i$-th machine.

## 4.2. An Alternative General Formulation

In this section we give a second mixed-integer program that models a general PFB. We introduce binary variables $z_{ijj'} \in \{0,1\}$ for each $i \in [m]$, $j, j' \in [n]$, $j \neq j'$, such that $z_{ijj'} = 1$, if and only if job $j$ is processed strictly before job $j'$ on machine $i$. Hence, if $j$ and $j'$ are processed in different batches, then exactly one of the two variables $z_{ijj'}$ and $z_{ij'j}$ has value one. If they belong to the same batch on machine $i$, then both are zero. There is no case in which both have value one. Additionally, the formulation contains continuous variables $c_{ij}$, $i \in [m]$, $j \in [n]$ denoting the completion time of job $j$ on machine $i$. The constraints of the MIP are:

$$
\begin{aligned}
& c_{1j} \geq r_j + p_1, && j \in [n], && \text{(4.2a)}\\
& c_{(i+1)j} \geq c_{ij} + p_{i+1}, && i \in [m-1],\ j \in [n], && \text{(4.2b)}\\
& c_{ij'} - c_{ij} \geq p_i - M(1 - z_{ijj'}), && i \in [m],\ j, j' \in [n],\ j \neq j', && \text{(4.2c)}\\
& c_{ij'} - c_{ij} \leq M z_{ijj'}, && i \in [m],\ j, j' \in [n],\ j \neq j', && \text{(4.2d)}\\
& \sum_{j' \in [n] \setminus \{j\}} z_{ijj'} + z_{ij'j} \geq n - b_i, && i \in [m],\ j \in [n], && \text{(4.2e)}\\
& z_{ijj'} \in \{0,1\}, && i \in [m],\ j, j' \in [n],\ j \neq j', && \text{(4.2f)}\\
& c_{ij} \in \mathbb{R}, && i \in [m],\ j \in [n]. && \text{(4.2g)}
\end{aligned}
$$

Here, (4.2a) and (4.2b) ensure that a job is available at each machine before it is processed there. By (4.2c), the completion time of two jobs $j$ and $j'$ on machine $i$ must differ by at least $p_i$, if they are not batched together. Note that this constraint also guarantees that not both $z_{ijj'}$ and $z_{ij'j}$ can have value one, because in this case $c_{ij'} - c_{ij}$ would have to be at least $p_i$ and at most $-p_i$ at the same time. Constraint (4.2d) ensures that $z_{ijj'}$ has to take value one, if $j$ is completed before $j'$ on machine $i$. In order to understand (4.2e), recall that $z_{ijj'} + z_{ij'j}$ takes value one, if and only if $j$ and $j'$ are not batched together on machine $i$, and value zero otherwise. Therefore, (4.2e) guarantees for each job $j$ that at least $n - b_i$ jobs are not batched together with $j$, which is equivalent to limiting the batch size to $b_i$. Finally, (4.2f) and (4.2g) define the ranges of the variables.

The formal correctness of this MIP formulation is shown by the following lemma, which is proven in the appendix in Section A.2.

**Lemma 4.1.** *Consider completion times $c_{ij}$, $i \in [m]$, $j \in [n]$. They define a feasible schedule $S$, if and only if there are binary values $z_{ijj'} \in \{0,1\}$, $i \in [m]$, $j, j' \in [n]$, $j \neq j'$, such that (4.2a) to (4.2g) are satisfied.*

## 4.3. A Formulation for a Fixed Permutation

In this section, we focus on PFB variants that satisfy Assumption 3.21. Thus, we can restrict ourselves to schedules in which the jobs are processed in the order $1, 2, \ldots, n$ on

all machines. For this case we give a simpler mixed-integer program. We define binary variables $z_{ij}$, $i \in [m]$, $j \in [n-1]$, as in Observation 3.24. This means that $z_{ij} = 1$, if and only if $j$ is the last job in its batch, i.e. jobs $j$ and $j+1$ are processed in different batches on machine $i$. Again, we have continuous variables $c_{ij}$, $i \in [m]$, $j \in [n]$, each of them denoting the completion time of job $j$ on machine $i$. Then the following constraints characterize a feasible permutation schedule:

$$c_{1j} \geq r_j + p_1 \qquad j \in [n], \tag{4.3a}$$
$$c_{(i+1)j} \geq c_{ij} + p_{i+1}, \quad i \in [m-1],\ j \in [n], \tag{4.3b}$$
$$c_{i(j+1)} \geq c_{ij} + p_i z_{ij}, \quad i \in [m],\ j \in [n-1], \tag{4.3c}$$
$$c_{i(j+1)} \leq c_{ij} + M z_{ij}, \quad i \in [m],\ j \in [n-1], \tag{4.3d}$$
$$\sum_{k=0}^{b_i-1} z_{i(j+k)} \geq 1, \qquad i \in [m],\ j \in [n-b_i], \tag{4.3e}$$
$$z_{ij} \in \{0,1\}, \qquad i \in [m],\ j \in [n-1], \tag{4.3f}$$
$$c_{ij} \in \mathbb{R}, \qquad i \in [m],\ j \in [n]. \tag{4.3g}$$

As in the general formulation (4.2), the constraints (4.3a) and (4.3b) ensure that a job is available at each machine before it is processed there. By (4.3c) and (4.3d), the finishing times of two consecutive jobs on machine $i$ are equal, if they are batched together, and is separated by at least $p_i$ otherwise. Constraint (4.3e) enforces that all batch sizes on machine $i$ are limited to $b_i$. Finally, (4.3f) and (4.3g) define the ranges of the variables.

Once again, the formal correctness of this MIP formulation is shown by the following lemma, which is proven in the appendix in Section A.3.

**Lemma 4.2.** *Consider completion times $c_{ij}$, $i \in [m]$, $j \in [n]$. They define a feasible permutation schedule $S$ ordered by the identity function, if and only if there are binary variables $z_{ij} \in \{0,1\}$, $i \in [m]$, $j \in [n-1]$, such that (4.3a) to (4.3g) are satisfied.*

Observe that this formulation involves only $\mathcal{O}(mn)$ binary variables. In contrast, the two formulations that model the general case involve $\mathcal{O}(mn^2)$ binary variables.

## 4.4. Modeling Objective Functions

This section explains how standard objective functions (compare Section 2.2) can be modeled via mixed-integer programming. Depending on the objective function, we add $n$ additional variables $C_j$, $F_j$, $L_j$, $T_j$ or $U_j$, for $j \in [n]$. They result in some new constraints. $C_j$, $F_j$ and $L_j$ are affine in the completion times $c_{mj}$:

$$C_j = c_{mj},$$
$$F_j = c_{mj} - r_j,$$
$$L_j = c_{mj} - d_j.$$

Tardiness can be modeled by adding the following constraints:

$$\begin{aligned} T_j &\geq c_{mj} - d_j, \\ T_j &\geq 0. \end{aligned}$$

Finally, unit penalties are realized as follows:

$$\begin{aligned} MU_j &\geq c_{mj} - d_j, \\ U_j &\in \{0, 1\}. \end{aligned}$$

A (weighted) sum objective function can now be directly expressed as a linear combination of those new variables. For bottleneck objective functions, another single variable representing the maximum has to be introduced. For instance, in order to minimize the makespan, introduce a variable $C_{\max}$ and add the constraints

$$C_{\max} \geq C_j$$

for all $j \in [n]$. The same construction works for any other bottleneck objective function.

In case of formulation (4.3), which is limited to permutation schedules, we can directly set $C_{\max} = c_{mn}$.

All in all, a mixed-integer program for scheduling a PFB can be constructed by combining one of the three formulations (4.1), (4.2), and (4.3) with the desired objective.

# 5. A Job-Wise Dynamic Program

In this chapter we present a dynamic program that finds an optimal schedule for a PFB, if Assumption 3.21 is satisfied. If the number $m$ of machines is constant, then the runtime of the algorithm is $\mathcal{O}(n^{q(m)})$ where $q$ is a quadratic polynomial. This proves that several PFB variants with fixed $m$ can be scheduled in polynomial time.

First we generalize an observation made in [Bap00, Section 2.1]. We show that it suffices to consider a limited set of possible completion times.

**Lemma 5.1.** *In a batch-active schedule for a PFB it holds that $c_{ij} \in \mathcal{C}_i$ for all $i \in [m]$, $j \in [n]$, where*

$$\mathcal{C}_i = \left\{ r_{j'} + \sum_{i'=1}^{i} \lambda_{i'} p_{i'} \;\middle|\; j' \in [n],\ \lambda_{i'} \in [n] \text{ for } i' \in [i] \right\}.$$

*In particular, $|\mathcal{C}_i| \leq n^{i+1} \leq n^{m+1}$.*

*Proof.* Fix some $i \in [m]$ and $j \in [n]$. Let $B_j$ be the batch of job $j$ on machine $i$. Since the schedule is batch-active, the following holds: $B_j$ is either started at the completion time of the previous batch $B_j^-$ on this machine or at the arrival time of one of the jobs of $B_j$. In the former case, all jobs $j' \in B_j^-$ satisfy that $c_{ij} = c_{ij'} + p_i$. In the latter case, there is a job $j'' \in B_j$ such that $c_{ij} = c_{(i-1)j''} + p_i$, where we write $c_{0j''} = r_{j''}$ for convenience. The claim follows inductively by observing that the former case can happen at most $n - 1$ times in a row, since there are at most $n$ batches on each machine. □

Note that this lemma does not require Assumption 3.21 yet. However, from now on, we suppose that Assumption 3.21 holds and the objective function is regular. Hence, by Lemmas 2.10 and 5.1, we may assume the existence of an optimal schedule which

- is a permutation schedule in which the jobs are processed in the order $1, 2, \ldots, n$ on all machines and
- satisfies $c_{ij} \in \mathcal{C}_i$ for all $i \in [m]$ and $j \in [n]$.

Therefore, for the remainder of this chapter, we restrict ourselves to such schedules, unless stated otherwise.

We also require that the objective function is either a sum or a bottleneck function. Both cases can be handled simultaneously by writing the objective function as $f(C) = \bigoplus_{j=1}^{n} f_j(C_j)$, where $\bigoplus \in \{\sum, \max\}$. We also use the symbol $\oplus$ as a binary operator.

The restriction to permutation schedules allows us to proceed in a job-wise manner as we will see in a moment. Given an instance $I$ of a PFB that satisfies Assumption 3.21 and a job $j \in [n]$, let $I_j$ be the modified instance which contains only the jobs $1, \ldots, j$. Next, we define the variables $g$ of our dynamic program. Let

$$g(j, t_1, t_2, \ldots, t_m, k_1, k_2, \ldots, k_m)$$

be the minimum objective value of a feasible schedule for $I_j$ under the constraint that for each machine $i$ it holds $c_{ij} = t_i$ and job $j$ is contained in a batch with $k_i$ jobs (including $j$). If no such schedule exists, the value of $g$ is defined to be $+\infty$.

The next lemma yields the starting values to calculate $g$ for $j = 1$.

**Lemma 5.2.** *For $j = 1$, $t_i \in \mathcal{C}_i$, and $k_i \in [b_i]$, $i \in [m]$, the starting values of $g$ are given by*

$$g(1, t_1, \ldots, t_m, k_1, \ldots, k_m) = \begin{cases} f_1(t_m), & \text{if } (*), \\ +\infty, & \text{otherwise,} \end{cases}$$

*where $(*)$ consists of the following conditions:*

*(i)* $k_i = 1$ *for all* $i \in [m]$,

*(ii)* $t_1 \geq r_1 + p_1$, *and*

*(iii)* $t_{i+1} \geq t_i + p_{i+1}$ *for all* $i \in [m-1]$.

*Proof.* The conditions of $(*)$ are necessary for the existence of a feasible schedule for instance $I_1$, since $I_1$ consists only of one job and there must be enough time to process this job on each machine. Conversely, if (i) to (iii) are satisfied, then the schedule defined by $c_{i1} = t_i$ is a feasible schedule for $I_1$ with objective value $f_1(C_1) = f_1(c_{m1}) = f_1(t_m)$. □

Now we turn to the recurrence formula to calculate $g$ for $j > 1$ from the values of $g$ for $j - 1$.

**Lemma 5.3.** *Given Assumption 3.21, for $j > 1$, $t_i \in \mathcal{C}_i$, and $k_i \in [b_i]$, $i \in [m]$, the values of $g$ are determined by*

$$g(j, t_1, \ldots, t_m, k_1, \ldots, k_m) = \begin{cases} \min\{f_j(t_m) \oplus g(j-1, t'_1, \ldots, t'_m, k'_1, \ldots, k'_m) \mid (**)\}, & \text{if } (*), \\ +\infty, & \text{otherwise,} \end{cases}$$

*where the minimum over the empty set is defined to be $+\infty$. Here, $(*)$ consists of the conditions*

*(i)* $t_1 \geq r_j + p_1$ *and*

*(ii)* $t_{i+1} \geq t_i + p_{i+1}$ *for all* $i \in [m-1]$,

*and* $(**)$ *consists of the following conditions for all* $i \in [m]$*:*

*(iii)* $t'_i \in \mathcal{C}_i$,

*(iv)* $k'_i \in [b_i]$,

*(v) if* $k_i = 1$, *then* $t'_i \leq t_i - p_i$, *and*

*(vi) if* $k_i > 1$, *then* $t'_i = t_i$ *and* $k'_i = k_i - 1$.

*Proof.* The conditions of $(*)$ are necessary for the existence of a feasible schedule for instance $I_j$, since there must be enough time to process job $j$ on each machine. Therefore, $g$ takes the value $+\infty$, if $(*)$ is violated. For the rest of the proof, assume that $(*)$ is satisfied.

For fixed values $t_i$ and $k_i$ consider a corresponding schedule $S$ for instance $I_j$, i.e. one which satisfies $c_{ij} = t_i$ and the batch of job $j$ has size $k_i$ for all $i \in [m]$. Then $S$ naturally defines a schedule $S'$ for instance $I_{j-1}$ by ignoring job $j$. Suppose that $S'$ corresponds to values $t'_i$ and $k'_i$ (i.e. $c_{i(j-1)} = t'_i$ and the batch of job $j-1$ has size $k'_i$ for all $i \in [m]$). Due to Assumption 3.21, on each machine $i \in [m]$ we have two possibilities: Either job $j$ is batched into the batch of job $j-1$, or a new batch is formed, consisting only of job $j$. In the former case, it must hold that $1 < k_i = k'_i + 1$ and $t'_i = t_i$, while the latter case requires $k_i = 1$ and $t_i \geq t'_i + p_i$, since the machine is occupied by the batch of job $j-1$ until then. These properties are exactly reflected by conditions (v) and (vi). Conditions (iii) and (iv) define the ranges for $t'_i$ and $k'_i$.

Hence, for given values $t_i$ and $k_i$, a corresponding schedule $S$ can be found, if and only if there is a schedule $S'$ corresponding to values $t'_i$ and $k'_i$ such that $(**)$ is satisfied. Among all these possibilities, we may choose the one which yields the cheapest objective function. This results in the claimed equation. □

Using these formulas we can prove the main theorem of this chapter.

**Theorem 5.4.** *Let a PFB instance with a constant number of* $m$ *machines be given such that Assumption 3.21 is satisfied. Suppose the objective function is regular and has the form* $f(C) = \bigoplus_{j=1}^{n} f_j(C_j)$ *with* $\bigoplus \in \{\sum, \max\}$. *Then an optimal schedule can be found in time* $\mathcal{O}(n^{m^2+5m+1})$.

*Proof.* Using Lemmas 5.2 and 5.3, one can calculate the values of $g$ for all $j \in [n]$ and all $t_i \in \mathcal{C}_i$, $k_i \in [b_i]$, $i \in [m]$. For a fixed job $j$ the number of these values is

$$\begin{aligned} & |\mathcal{C}_1| \cdot |\mathcal{C}_2| \cdot \ldots \cdot |\mathcal{C}_m| \cdot b_1 \cdot b_2 \cdot \ldots \cdot b_m \\ \leq\ & n^2 \cdot n^3 \cdot \ldots \cdot n^{m+1} \cdot n^m \\ =\ & n^{\frac{(m+1)(m+2)}{2}-1+m} \\ =\ & n^{\frac{m^2}{2}+\frac{5m}{2}}. \end{aligned}$$

In total, this results in at most $n^{\frac{m^2}{2}+\frac{5m}{2}+1}$ values to be calculated for all $j$. For calculating one of these values, one takes a minimum over at most all values for $j-1$. Hence, the total time to compute all values of $g$ is bounded by

$$\mathcal{O}(n^{\frac{m^2}{2}+\frac{5m}{2}+1} \cdot n^{\frac{m^2}{2}+\frac{5m}{2}}) = \mathcal{O}(n^{m^2+5m+1}).$$

Due to Lemma 5.1, the objective value can then be determined as the minimum of all values of $g$ for $j = n$. If for each value of $g$ we additionally store a reference to the previous value, we can reconstruct the optimal schedule by following these references in a backtracking manner. This does not increase the asymptotic runtime, since computing the values is more expensive than backtracking. □

The previous theorem shows that many variants of a PFB with constant $m$ are solvable in polynomial time. This is summarized in the next corollary.

**Corollary 5.5.** *The following problems can be solved in polynomial time, where $Fm$ denotes a flow shop of a constant number of $m$ machines:*

- $Fm \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid C_{\max}$
- $Fm \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid \sum C_j$
- $Fm \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid F_{\max}$
- $Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid L_{\max}$
- $Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid \sum T_j$
- $Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid \sum w_j C_j$.

*Moreover, the problems*

- $Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid C_{\max}$
- $Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid \sum C_j$

*can be solved in pseudo-polynomial time.*

*Proof.* Using Theorem 5.4, we have to check which PFB variants satisfy Assumption 3.21. Referring to Chapter 3, these are exactly the instances listed in the statement of this corollary. For most of the variants, the runtime $\mathcal{O}(n^{m^2+5m+1})$ is polynomial in the encoding length of the instance. However, this is not the case for the last two variants, because they are high-multiplicity problems. Therefore, the dynamic program is only pseudo-polynomial for them. □

Apart from the PFB variants mentioned so far, there is another variant that does not satisfy Assumption 3.21, but for which the dynamic program can still be applied. This is covered by the next corollary.

**Corollary 5.6.** *The problem*

- $Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid \sum U_j$

*can be solved in polynomial time.*

*Proof.* Referring to Chapter 3, for this PFB variant there always exists an optimal permutation schedule, but it is a priori not clear how to find the optimal permutation. Suppose the jobs are indexed by the earliest-due-date rule, i.e. $d_1 \leq d_2 \leq \cdots \leq d_n$. We aim to minimize the number of late jobs. Note that the only distinctive feature of the jobs is the due date. Therefore, if it is possible to find a schedule with at most $k$ late jobs, then we may assume that these are among the jobs $1, 2, \ldots, k$ which have the smallest due dates. Moreover, we may assume that the remaining $n - k$ jobs are processed in earliest-due-date order, i.e. in the order $k+1, k+2, \ldots, n$. Therefore, we only need to try the $n$ permutations $k+1, k+2, \ldots, n, 1, 2, \ldots, k$ for each possible value of $k \in [n]$. Hence, we succeed by applying the dynamic program $n$ times. □

**Remark 5.7.** In fact, the procedure in the proof of Corollary 5.6 can be enhanced by applying a binary search on $k$. This way, it suffices to apply the dynamic program $\mathcal{O}(\log(n))$ times.

# 6. A Multi-Criteria Point of View

In the previous chapter we constructed an algorithm which proceeds job-wise. However, if the number of machines grows, this algorithm becomes very slow. In this chapter we choose a different approach. The idea is to compute, machine by machine, a sufficiently large set of possible partial schedules such that an optimal schedule is among them when we arrive at the last machine. This yields an algorithm which is pseudo-polynomial for a fixed number of jobs $n$. Moreover, in Chapter 7, we show that in the case $n = 2$, this algorithm is even polynomial.

## 6.1. Efficiency and Non-Dominance

In order to determine which partial schedules are interesting to compute and store, we take a multi-criteria perspective. An introduction to multi-criteria optimization can be found in [Ehr05]. For a fixed machine $i \in [m]$, we consider each completion time $c_{ij}$ of a job $j \in [n]$ as its own objective function. This leads to the following definition:

**Definition 6.1.**

(i) A schedule $S$ *dominates* a schedule $S'$ at machine $i$, if $c_{ij}(S) \leq c_{ij}(S')$ for all $j \in [n]$ and strict inequality holds for at least one $j$.

(ii) A schedule which is not dominated by any other schedule at machine $i$ is called *efficient* (or *Pareto-optimal*) at machine $i$.

(iii) A vector $c_i = (c_{ij})_{j\in[n]} \in \mathbb{R}^n$ is called *non-dominated* at machine $i$, if it is the completion time vector of an efficient schedule at this machine.

The remainder of this section is devoted to collecting helpful properties about efficient schedules and non-dominated completion time vectors. The first property is usually called external stability:

**Lemma 6.2.** *Every feasible schedule which is not efficient at machine $i$ is dominated by an efficient schedule at machine $i$.*

*Proof.* Careful analysis of the set of feasible completion time vectors at machine $i$ shows that this set is $\mathbb{R}^n_{\leqq}$-compact, as defined in [Ehr05, Definition 2.13]. Then [Ehr05, Theorem 2.21] implies external stability. □

**Lemma 6.3.** *Let $S$ be a schedule that is efficient at machine $i$. Then there exists a schedule $S'$ with $c_i(S) = c_i(S')$ which is efficient at all machines $1, \dots, i$.*

*Proof.* For machines $i' = i-1, i-2, \dots, 1$, modify $S$ as follows: If $S$ is not already efficient at machine $i'$, then there exists a schedule $S''$ dominating $S$ at machine $i'$ by Lemma 6.2. Replace the partial schedule of $S$ up to machine $i'$ by the partial schedule of $S''$ up to machine $i'$, without modifying $S$ for the machines $i'+1, \dots, n$. The dominance relation ensures that $S$ remains feasible. After doing this for all $i' = i-1, i-2, \dots, 1$, $S$ satisfies the conditions stated for $S'$ in the claim. □

**Lemma 6.4.** *Let $S$ be a schedule that is efficient at all machines. Then $S$ is batch-active.*

*Proof.* Suppose there is a batch that can be started earlier without violating feasibility and without changing the order of batches. Let $i \in [m]$ be the machine on which this batch is processed. By starting the batch earlier, we obtain a schedule that dominates $S$ at machine $i$, contradicting the efficiency of $S$ there. □

**Lemma 6.5.** *If release dates and processing times are integer and $S$ is efficient at machine $i$, then $c_i(S) \in \mathbb{Z}^n$.*

*Proof.* Consider the restricted instance consisting of the machines $1, \dots, i$. By Lemma 6.3, we may assume without loss of generality that $S$ is efficient at all machines $1, \dots, i$. Then Lemma 6.4 yields that $S$ is batch-active. The claim follows using Lemma 5.1. □

**Lemma 6.6.** *With respect to any regular objective function, there is an optimal schedule which is efficient at each machine.*

*Proof.* Let $S$ be an optimal schedule, which exists due to Lemma 2.11. If $S$ is not efficient at machine $m$, then by Lemma 6.2 there exists a schedule $S'$ dominating $S$ at machine $m$ that is efficient there. Regularity of the objective function yields that $S'$ must be optimal, too. Hence, by possibly replacing $S$ by $S'$, we may assume that $S$ is efficient at machine $m$. Using Lemma 6.3, we can also assume that $S$ is efficient at all other machines. This already implies the claim. □

## 6.2. The Efficient-Partial-Schedule Algorithm

Lemma 6.6 justifies an algorithm which works as follows: For each machine $i \in [m]$, compute a set of efficient partial schedules $\mathcal{S}_i$ up to machine $i$ that represents all non-dominated vectors $c_i$. Given $\mathcal{S}_{i-1}$, $\mathcal{S}_i$ can be obtained by continuing each partial schedule in $\mathcal{S}_{i-1}$ by each feasible batching on machine $i$ in a batch-active way. Dominated options are eliminated. Algorithm 1 describes how this can be done.

**Algorithm 1** Efficient-Partial-Schedule Algorithm

**Input:** An instance of a PFB with regular objective function.
**Output:** A schedule minimizing the objective function.

```
1: S_0 := {"empty" schedule containing only release date information}.
2: for i ∈ [m] do
3:     S_i := ∅.
4:     for S ∈ S_{i-1} do
5:         for each feasible batching B on machine i do
6:             S' := batch-active continuation of S by B.
7:             Add S' to S_i.
8:             for S'' ∈ S_i \ {S'} do
9:                 if S'' dominates S' or c_i(S') = c_i(S'') then
10:                    Remove S' from S_i.
11:                    break.
12:                end if
13:                if S' dominates S'' then
14:                    Remove S'' from S_i.
15:                end if
16:            end for
17:        end for
18:    end for
19: end for
20: Among all schedules in S_m, return one which minimizes the objective function.
```

**Theorem 6.7.** *For any PFB instance with regular objective function, Algorithm 1 finds an optimal schedule. If release dates and processing times are integer, it can be implemented to run in time* $\mathcal{O}(mn(M^n 2^n n!)^2)$*, where* $M = \max_{j\in[n]} r_j + n\sum_{i=1}^{m} p_i$ *is the constant from Lemma 2.12. If the instance satisfies Assumption 3.21, then the runtime can be improved to* $\mathcal{O}(mnM^{2n}2^{2n})$*.*

*Proof.* By construction and Lemma 6.3, the algorithm finds all non-dominated vectors at all machines, as well as one efficient schedule belonging to each non-dominated vector. By Lemma 6.6, this suffices to find an optimal schedule.

For analyzing the running time, we first discuss, how the elements of $\mathcal{S}_i$ can be stored. One way of doing this is to store only the batch configuration and the resulting completion times on machine $i$, as well as a pointer to the previous element in $\mathcal{S}_{i-1}$. Then, each element takes $\mathcal{O}(n)$ storage and the final schedule can be reconstructed in $\mathcal{O}(nm)$ time.

Observe that by the Lemmas 2.12, 6.4, and 6.5 there are at most $M^n$ non-dominated vectors at each machine. An upper bound for the number of feasible batchings on each machine is given by $2^n n!$. This is because there are $n!$ options to choose a permutation

first, and then at most $2^n$ possible batchings corresponding to this permutation, referring to Theorem 3.28. Therefore, we investigate at most $M^n 2^n n!$ potential schedules for each machine. Hence, we perform at most $\mathcal{O}(m(M^n 2^n n!)^2)$ dominance checks in total. Each dominance check takes $\mathcal{O}(n)$ time. All other operations are not relevant asymptotically, resulting in the claimed total running time. If Assumption 3.21 is satisfied, we may fix a permutation throughout the whole algorithm. This allows the factor $n!$ to be dropped. □

**Remark 6.8.** Even if Assumption 3.21 is satisfied, the number of jobs $n$ occurs in the exponent of the runtime. Therefore Algorithm 1 is only interesting, if $n$ is very small. However, for fixed $n$, the runtime of Algorithm 1 is pseudo-polynomial.

The following example shows that the number of non-dominated completion time vectors can be exponential in $n$, even if we restrict ourselves to a fixed job permutation.

**Example 6.9.** Consider a PFB instance without release dates where the first two machines are specified by $p_1 = b_1 = 1$, $p_2 = 2$, and $b_2 = 3$. We restrict the job order to $1, 2, \ldots, n$ on all machines. On the first machine, we have no choice in our batching. The arrival time of job $j$ at the second machine is $c_{1j} = j$. There, we construct exponentially many non-dominated completion time vectors.

Consider partial schedules where the batching on the second machine uses a batch of size one for the first job and batches of sizes two or three for all other jobs. Let $\mathcal{S}$ be the set of batch-active schedules resulting from such a batching. Figure 6.1 illustrates two different schedules belonging to $\mathcal{S}$ in the case $n = 7$. We prove our claim that there are exponentially many non-dominated completion time vectors at the second machine using the following three steps:

(i) All schedules in $\mathcal{S}$ belong to distinct completion time vectors.

(ii) All schedules in $\mathcal{S}$ are efficient at the second machine.

(iii) $|\mathcal{S}|$ is exponential in $n$.

For proving (i), consider a schedule $S \in \mathcal{S}$. By $p_2 = 2$ and the fact that, except for the first batch, each batch on the second machine has size at least two, the second machine is already idle when the last job of any batch arrives. Hence, any batch will be started exactly upon arrival of its last job. Therefore, it holds $c_{2j}(S) = j + 2$, if and only if $j$ is the last job in a batch. This implies (i) by Observation 3.24.

Next, we show (ii). Assume that the completion time vector $c_2 := c_2(S)$ of some $S \in \mathcal{S}$ is dominated by another vector $c_2' := c_2(S')$ for some schedule $S'$ (which does not need to be contained in $\mathcal{S}$). For each job $j$ that is the last job in its batch in $S$, it must hold that $c_{2j}' \leq c_{2j} = j + 2$. This can only be achieved if $j$ is also the last job in its batch in $S'$. Therefore, the batching $\mathcal{B}'$ of $S'$ on the second machine is at least as fine as the batching $\mathcal{B}$ of $S$, where we call a batching finer than another batching, if the first one can be constructed from the second one solely by splitting batches. Since $c_2$ and $c_2'$ must

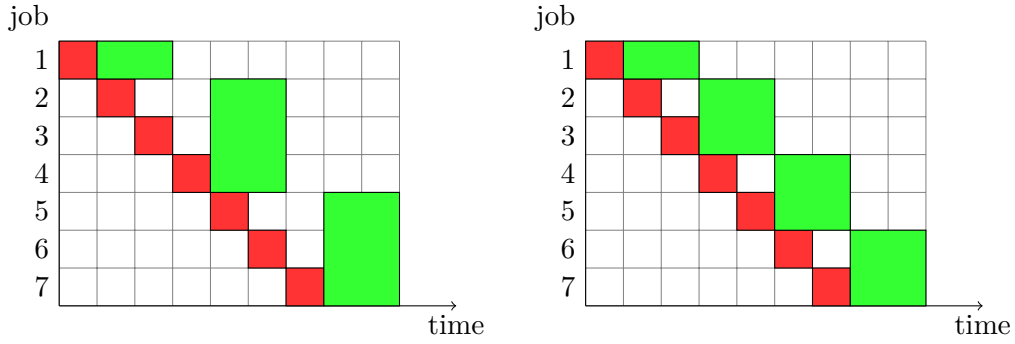


Figure 6.1.: Two efficient schedules for $n = 7$ in Example 6.9.

be different, $\mathcal{B}'$ is even strictly finer than $\mathcal{B}$. Hence, there exists a batch $B_*$ in $\mathcal{B}$ which is split into at least two batches in $\mathcal{B}'$. Let $j_1$ be the last job in the batch before $B_*$ and $j_2$ the last job in $B_*$. Then we obtain $c'_{2j_2} - c'_{2j_1} = c_{2j_2} - c_{2j_1} = j_2 + 2 - (j_1 + 2) = j_2 - j_1 \leq 3$, since $B_*$ has at most size 3. On the other hand, $c'_{2j_2} - c'_{2j_1} \geq 2p_2 = 4$, since $B_*$ is split into at least two batches in $S'$. This contradiction proves efficiency of $S$.

Finally, we show (iii). For convenience, suppose $n = 6k + 1$ for some $k \in \mathbb{N}$. Split the jobs $2, \ldots, n$ into $k$ groups of six subsequent jobs. An element of $\mathcal{S}$ can be constructed by deciding for each of the groups, whether we use two batches of size three (as in the left schedule in Figure 6.1) or three batches of size two (as in the right schedule in Figure 6.1). Thus, $|\mathcal{S}| \geq 2^k = (2^{\frac{1}{6}})^{n-1}$, which is exponential in $n$.

# 7. Special Cases

In this chapter we prove that some special cases of a PFB can be scheduled in polynomial time. First, we show that the Efficient-Partial-Schedule algorithm introduced in the previous chapter runs in polynomial time for $n = 2$. Afterwards, we concentrate on constant batch capacities and constant processing times. Finally, we investigate the high-multiplicity problem to minimize the makespan in a two-machine PFB without release dates.

## 7.1. Two Jobs

First, we focus on a PFB with only two jobs, i.e. the problem

$$F \mid n = 2, r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid f.$$

We show that it can be solved in polynomial time for any regular objective function $f$. We prove this by showing that for $n = 2$, Algorithm 1 can be implemented to run in $\mathcal{O}(m^2)$ time. First, we define a set $P$ by

$$P := \{\pm p_i \mid i \in [m]\} \cup \{0, r_2 - r_1\}.$$

Observe that $|P| \leq 2m + 2$. This set helps us to bound the number of non-dominated completion time vectors per machine, as shown by the next three lemmas.

**Lemma 7.1.** *Consider a PFB instance with $n = 2$. For any batch-active schedule it holds that $c_{i2} - c_{i1} \in P$ for all $i \in [m]$.*

*Proof.* We use induction on $i$. The induction start is settled by setting $c_{0j} = r_j$ for $j = 1, 2$. For the induction step, let $i \in [m]$. If the two jobs are batched together on machine $i$, then the claim holds trivially, since the difference is zero. Hence, assume they are not batched together.

First, suppose $c_{i2} > c_{i1}$. Due to batch-activeness, both jobs must be started as soon as possible on machine $i$. In formulas this means $c_{i1} = p_i + c_{(i-1)1}$ and $c_{i2} = p_i + \max\{c_{i1}, c_{(i-1)2}\}$. Hence, depending on which of the two terms determines the maximum, $c_{i2} - c_{i1} \in \{p_i, c_{(i-1)2} - c_{(i-1)1}\}$. This settles the claim, since it holds $c_{(i-1)2} - c_{(i-1)1} \in P$ by induction.

The case $c_{i2} < c_{i1}$ can be handled analogously. □

**Lemma 7.2.** *Consider a PFB instance with $n = 2$. If $S$ is an efficient schedule at machine $i$, then $c_{i2} - c_{i1} \in P$ for all $i \in [m]$.*

*Proof.* Due to Lemma 6.3, we may assume that $S$ is efficient at all machines up to machine $i$. Then, Lemma 6.4 implies that $S$ is batch-active up to machine $i$. The claim follows by Lemma 7.1. □

**Lemma 7.3.** *If $n = 2$, then there are at most $2m + 2$ non-dominated completion time vectors per machine.*

*Proof.* Let $c_i$ and $c'_i$ be two distinct non-dominated completion time vectors at machine $i \in [m]$. Then it either holds that $c_{i1} < c'_{i1}$ and $c_{i2} > c'_{i2}$ or it holds that $c_{i1} > c'_{i1}$ and $c_{i2} < c'_{i2}$. In particular, $c_{i2} - c_{i1} \neq c'_{i2} - c'_{i1}$. The claim follows by Lemma 7.2. □

Using this, the factor $M^n$ in the runtime of Algorithm 1 in Theorem 6.7 can be replaced by $\mathcal{O}(m)$. This results in a running time of $\mathcal{O}(m^3)$ for $n = 2$. In the following we show how this can be improved to $\mathcal{O}(m^2)$ by a slight modification of the algorithm.

The main idea of this improvement is to do less dominance checks. Instead of keeping sets $\mathcal{S}_i$ of non-dominated options, we keep at most one schedule $S_i(t)$ for each machine $i$ and each completion time difference $t \in P$. This schedule $S_i(t)$ is the best schedule found so far that satisfies $c_{i2} - c_{i1} = t$. The full procedure is summarized in Algorithm 2.

**Theorem 7.4.** *For any PFB instance with regular objective function and $n = 2$, Algorithm 2 finds an optimal schedule. It can be implemented to run in time $\mathcal{O}(m^2)$.*

*Proof.* In contrast to Algorithm 1, this algorithm may also find and continue dominated options. Nevertheless, as in Theorem 6.7, it inductively follows by Lemma 6.3 that all non-dominated completion time vectors are among them. Therefore, by Lemma 6.6, an optimal schedule will be found.

For analyzing the running time, note that we can store the partial schedules in the same way as in Theorem 6.7: Only store the last batching and a reference to the previous partial schedule. Since $n = 2$, this representation has constant size, and the final schedule can be reconstructed in time $\mathcal{O}(m)$.

The initialization of Algorithm 2 consists of the first two nested loops and takes $\mathcal{O}(m^2)$ time.

Moreover, note that the innermost loop of Algorithm 2 has only at most three iterations, since there are only three batchings for $n = 2$. Hence, the whole loop takes only constant time. It will be called $\mathcal{O}(m^2)$ times due to the outer two for-loops. Hence, the total running time is $\mathcal{O}(m^2)$. □

**Algorithm 2** Efficient-Partial-Schedule Algorithm for $n = 2$

**Input:** An instance of a PFB with regular objective function and $n = 2$.
**Output:** A schedule minimizing the objective function.

**for** $i \in [m] \cup \{0\}$ **do**
  **for** $t \in P$ **do**
    $S_i(t) :=$ **null**.
  **end for**
**end for**
$S_0(r_2 - r_1) =$ "empty" schedule containing only release date information.
**for** $i \in [m]$ **do**
  **for** $t \in P$ such that $S_{i-1}(t) \neq$ **null do**
    **for** each feasible batching $\mathcal{B}$ on machine $i$ **do**
      $S' :=$ batch-active continuation of $S_{i-1}(t)$ by $\mathcal{B}$.
      $t' := c_{i2}(S') - c_{i1}(S')$.
      **if** $S_i(t') =$ **null** or $c_{i1}(S') < c_{i1}(S_i(t'))$ **then**
        $S_i(t') := S'$.
      **end if**
    **end for**
  **end for**
**end for**
Among all schedules $S_m(t)$, $t \in P$, return one which minimizes the objective function.

## 7.2. Constant Processing Times or Constant Batch Capacities

The purpose of this section is to briefly mention that scheduling most PFB variants becomes easy, if there is either a constant processing time $p_i = p$, or a constant batch capacity $b_i = b$ for all machines $i \in [m]$. The reason for this is that in both cases the flow shop can be reduced to a single machine, using the reduction procedure proposed by Sung et al. [SKY00]. Scheduling a single batching machine with job-independent processing times can be achieved in polynomial time for almost all standard objective functions except for weighted tardiness, compare [Bap00]. For $\sum w_j T_j$, this question is still open.

## 7.3. Two Machines without Release Dates

This section deals with the specific problem

$$F2 \mid p_{ij} = p_i, p\text{-batch}, b_i \mid C_{\max},$$

i.e. with makespan minimization in a two-machine PFB without release dates. As mentioned before, Ahmadi et al. [AADT92] show that the problem can be solved in time

$\mathcal{O}(n)$ by applying LOE batching on the first machine and FOE batching on the second machine. However, remember that this problem has the high-multiplicity property, since the jobs are identical and do not have to be encoded separately, compare Section 2.4. Hence, the runtime $\mathcal{O}(n)$ is only pseudo-polynomial. In this section we build upon the results of [AADT92] and show that finding the minimum makespan is equivalent to solving an unbounded knapsack problem with two variables, which is solvable in polynomial time as proven in [HW76].

We are going to use the following fact from [AADT92]:

**Theorem 7.5.** *In the case $m = 2$, the optimal makespan of a PFB without release dates is given by*

$$C_{\max} = \max\left\{ \left\lceil \frac{n-(k-1)b_2}{b_1} \right\rceil p_1 + kp_2 \;\middle|\; k = 1, \ldots, \left\lceil \frac{n}{b_2} \right\rceil \right\}.$$

*Proof.* See [AADT92, Section 2.3]. □

Evaluating this formula takes $\mathcal{O}(\lceil \frac{n}{b_2} \rceil)$ time, which is not polynomial in the length of a binary encoded input. The next theorem shows that the optimal makespan can be computed in polynomial time:

**Theorem 7.6.** *In the case $m = 2$, the optimal makespan of a PFB without release dates can be determined in polynomial time.*

*Proof.* First observe that for two natural numbers $r, s \in \mathbb{N}$, it holds that $\lceil \frac{r}{s} \rceil = \lfloor \frac{r+s-1}{s} \rfloor$, since both values are equal to the unique integer among the $s$ fractions $\frac{r}{s}, \frac{r+1}{s}, \ldots, \frac{r+s-1}{s}$. Hence, from Theorem 7.5, we get:

$$\begin{aligned} C_{\max} &= \max\left\{ \left\lfloor \frac{n-(k-1)b_2+b_1-1}{b_1} \right\rfloor p_1 + kp_2 \;\middle|\; k = 1, \ldots, \left\lceil \frac{n}{b_2} \right\rceil \right\} \\ &= \max\left\{ lp_1 + kp_2 \;\middle|\; l \leq \frac{n-(k-1)b_2+b_1-1}{b_1} \text{ and } k, l \in \mathbb{N} \right\} \\ &= \max\left\{ lp_1 + kp_2 \mid (l-1)b_1 + (k-1)b_2 \leq n-1 \text{ and } k, l \in \mathbb{N} \right\}. \end{aligned}$$

By substituting $z_1 = l - 1$ and $z_2 = k - 1$, this is equal to the objective value of the following two-item unbounded knapsack problem:

$$\begin{aligned} \max\ & (z_1+1)p_1 + (z_2+1)p_2 \\ \text{s.t. } & z_1 b_1 + z_2 b_2 \leq n-1 \\ & z_1, z_2 \in \mathbb{N}_0. \end{aligned}$$

Such a problem can be solved in polynomial time as shown in [HW76]. Thus, the same holds for our problem of finding the optimal makespan in the case $m = 2$. □

This shows that the evaluation version and the decision version of

$$F2 \mid p_{ij} = p_i, p\text{-batch}, b_i \mid C_{\max}$$

are solvable in polynomial time. For the optimization version the answer is less clear. We cannot explicitly output all start or completion times of all jobs, since these are $\mathcal{O}(n)$ numbers, and thus too many. However, referring to [AADT92], it is clear what the optimal schedule looks like: It uses LOE batching on the first machine and FOE batching on the second machine. Moreover, each batch on the first machine is processed as early as possible and each batch on the second machine as late as possible without violating the previously computed optimal makespan $t$. Hence, we obtain the following formulas, which can be evaluated in constant time for a fixed job $j \in [n]$, once $t$ is known:

$$\begin{aligned} c_{1j} &= p_1 \left\lceil \frac{j}{b_1} \right\rceil, \\ c_{2j} &= t - p_2 \left( \left\lceil \frac{n-j+1}{b_2} \right\rceil - 1 \right). \end{aligned}$$

Outputting these formulas as a compact schedule representation can be seen as solving the optimization version of the problem in polynomial time.

# 8. Lower Bounds and Approximations

As mentioned earlier, the complexity status of scheduling most PFB variants is open. In particular, no polynomial algorithms are known. Therefore, in this chapter, we aim to find efficient approximation algorithms. We focus on those cases where Assumption 3.21 is satisfied.

## 8.1. A Lower Bound with Three Interpretations

In this section we develop a lower bound $c^*_{ij}$ for each completion time $c_{ij}$, $i \in [m]$, $j \in [n]$, in a permutation schedule processing the jobs in the order $1, 2, \ldots, n$. Afterwards, we show that this lower bound can be interpreted in three different ways.

We start by collecting some properties that hold for the completion times of such a schedule. Firstly, thanks to the fixed permutation, we have

$$c_{i(j+1)} \geq c_{ij} \tag{8.1}$$

for all $i \in [m]$, $j \in [n-1]$. Secondly, due to the processing time on each machine, we obtain

$$c_{(i+1)j} \geq c_{ij} + p_{i+1} \tag{8.2}$$

for all $i \in [m-1]$, $j \in [n]$. Thirdly, the batch capacity on each machine in combination with the fixed permutation yield

$$c_{i(j+b_i)} \geq c_{ij} + p_i \tag{8.3}$$

for all $i \in [m]$, $j \in [n-b_i]$.

Our lower bound builds upon these properties (8.1) to (8.3). For convenience, we define as starting values

$$c^*_{0j} = r_j$$

for all $j \in [n]$ and

$$c^*_{ij} = -\infty$$

for $i \in [m]$ and $j \leq 0$. Then, we recursively define

$$c^*_{ij} = \max\{c^*_{i(j-1)}, p_i + c^*_{(i-1)j}, p_i + c^*_{i(j-b_i)}\}.$$

The next lemma states that these values are in fact a lower bound for the completion times of a permutation schedule for a PFB.

**Lemma 8.1.** *If a schedule for a PFB is ordered by the identity permutation, then it satisfies $c_{ij} \geq c^*_{ij}$ for all $i \in [m]$ and $j \in [n]$.*

*Proof.* Inductively apply (8.1) to (8.3). □

Next, we turn to the promised three interpretations.

### Interpretation 1: Hybrid Flow Shop

A *hybrid flow shop* is a flow shop variant where some (or all) of the $m$ stages consist of multiple parallel machines, instead of a single one. Each job has to be processed on one of these parallel machines at each stage. For a literature review about hybrid flow shop problems we refer to [RVR10].

**Notation 8.2.** When considering hybrid flow shops, an index $i \in [m]$ does not denote a single machine, but a stage consisting of $m_i$ parallel machines $M^{(i)}_1, \ldots, M^{(i)}_{m_i}$.

For each PFB instance we consider a corresponding *proportionate hybrid flow shop* (PHF) instance as follows: Machine $i$ of the PFB is replaced by $m_i = b_i$ parallel machines in the PHF. These machines can only process one job at a time, i.e. they have batch capacity 1. The processing time of each job remains $p_i$ on a machine at stage $i$.

Observe that this PHF can simulate any PFB schedule by starting all parallel machines at a stage $i$ simultaneously. Hence, any feasible completion times $c_{ij}$, $i \in [m]$, $j \in [n]$, of a PFB are feasible for the corresponding PHF as well.

The next theorem shows that an optimal PHF schedule is determined by the values $c^*_{ij}$, $i \in [m]$, $j \in [n]$, which is the first of our three interpretations.

**Theorem 8.3.** *Given a PFB instance, the values $c^*_{ij}$, $i \in [m]$, $j \in [n]$, are the completion times of an optimal permutation schedule of the corresponding PHF instance with respect to any regular objective function.*

*Proof.* Since there are no batching machines involved in a PHF, there is no need to wait for the arrival of other jobs in order to achieve a fuller batch. Therefore, an optimal permutation schedule for the PHF can be achieved by starting each job $j \in [n]$ at each stage $i$ as soon as the following three conditions are satisfied:

- The previous job $j-1$ has been started at stage $i$, in order to ensure the permutation.
- The job $j$ has finished stage $i-1$ (or has been released, if $i=0$).
- There is a machine available for processing job $j$ at stage $i$. Due to the fixed permutation, this is the case as soon as job $j-b_i$ has finished stage $i$.

Putting these conditions together, the earliest possible completion time of job $j$ at stage $i$ can be calculated recursively by

$$c_{ij} = \max\{c_{i(j-1)}, p_i + c_{(i-1)j}, p_i + c_{i(j-b_i)}\}.$$

This is exactly the same formula as in the definition of the values $c^*_{ij}$, $i \in [m]$, $j \in [n]$. □

### Interpretation 2: LP-Relaxation

A standard way to obtain lower bounds for an optimization problem is to compute an optimal solution to the LP-relaxation of a mixed-integer programming formulation of the problem. It turns out that doing this for our mixed-integer program (4.3) leads exactly to the values $c^*_{ij}$, $i \in [m]$, $j \in [n]$.

**Theorem 8.4.** *For any regular objective function, the values $c^*_{ij}$, $i \in [m]$, $j \in [n]$, together with some values $z_{ij} \in [0,1]$, $i \in [m]$, $j \in [n-1]$, are an optimal solution to the LP-relaxation of the MIP (4.3).*

*Proof.* We first show feasibility. Define

$$z_{ij} = \min\left\{1, \frac{c^*_{i(j+1)} - c^*_{ij}}{p_i}\right\} \in [0,1]$$

for all $i \in [m]$, $j \in [n-1]$. We show for each constraint that it is satisfied by $c^*$ and $z$. Constraints (4.3a) and (4.3b) are fulfilled due to the definition of $c^*$. Constraint (4.3c) holds by the definition of $z$. For (4.3d) consider two cases: If $z_{ij} = 1$, then (4.3d) is satisfied due to the big $M$ on the right-hand side. If $z_{ij} = \frac{c^*_{i(j+1)} - c^*_{ij}}{p_i}$, then (4.3d) holds due to $M > p_i$. Finally, for (4.3e), consider two cases again: If one of the variables $z_{i(j+k)}$ has value 1, then the constraint is trivially satisfied. If not, then it holds that $z_{i(j+k)} = \frac{c^*_{i(j+k+1)} - c^*_{i(j+k)}}{p_i}$ for all $k$. Hence, the left-hand side is a telescoping sum that equals $\frac{c^*_{i(j+b_i)} - c^*_{ij}}{p_i}$. By the definition of $c^*$, this fraction is at least 1. Thus, feasibility follows.

For proving optimality, let $c_{ij}$, $i \in [m]$, $j \in [n]$, be the $c$-values of any other feasible solution. By (4.3c), $c$ satisfies (8.1). By (4.3a) and (4.3b), $c$ satisfies (8.2). Finally, by (4.3c) and (4.3e), $c$ satisfies (8.3). Hence, it inductively follows that $c_{ij} \geq c^*_{ij}$ for $i \in [m]$, $j \in [n]$. Thus, with respect to any regular objective function, $c^*$ is an optimal solution to the LP-relaxation. □

Observe that combining Theorems 8.3 and 8.4 yields:

**Corollary 8.5.** *With respect ot any regular objective function, an optimal permutation schedule for a PHF can be found by solving the LP-relaxation of (4.3).*

### Interpretation 3: Generalizing the Knapsack Problem of Theorem 7.6

Let us focus on makespan minimization for a PFB without release dates. As shown in the proof of Theorem 7.6, for the case $m = 2$, the minimum makespan is given by the objective value of the following knapsack problem:

$$\begin{aligned} \max\ & (z_1 + 1)p_1 + (z_2 + 1)p_2 \\ \text{s.t. } & z_1 b_1 + z_2 b_2 \leq n - 1, \\ & z_1, z_2 \in \mathbb{N}_0. \end{aligned}$$

Let us consider a generalization of this to $m$ machines, namely

$$\max \sum_{i=1}^{m} p_i(z_i + 1) \tag{8.4a}$$

$$\text{s.t. } \sum_{i=1}^{m} b_i z_i \leq n - 1, \tag{8.4b}$$

$$z_i \in \mathbb{N}_0, i \in [m]. \tag{8.4c}$$

Unfortunately, the objective value of (8.4) is not equal to the optimal makespan, as the following example shows.

**Example 8.6.** Consider a PFB instance without release dates consisting of three machines and two jobs (i.e. $m = 3$, $n = 2$), where $p_1 = b_1 = p_3 = b_3 = 1$ and $p_2 = b_2 = 2$. The only batching decision to make is whether to batch both jobs together on machine 2 or not. The two corresponding batch-active permutation schedules are illustrated in Figure 8.1. The makespan is 6 in both cases. However, the objective value of the knapsack problem (8.4) is 5. Hence, the objective value of the knapsack problem is not equal to the optimal makespan.

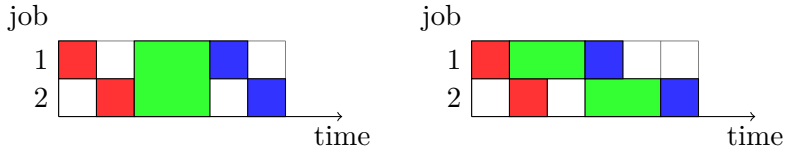


Figure 8.1.: The two possible schedules in Example 8.6.

However, a different connection between the PFB and the knapsack problem can be established, yielding our third interpretation of the values $c^*_{ij}$.

**Theorem 8.7.** *For a PFB without release dates, the objective value of (8.4) is equal to $c^*_{mn}$. In particular, (8.4) is a lower bound for the makespan.*

*Proof.* The recursive definition of $c^*_{ij}$ is a standard dynamic programming algorithm for solving an unbounded knapsack problem, compare e.g. [Wol98, Section 5.4.2]. □

Note that Example 8.6 and Theorem 8.7 imply that the lower bound $c^*_{ij}$ for $c_{ij}$ is not tight in general.

We conclude this section by comparing our lower bound (8.4) for the optimal makespan in a PFB without release dates to the following one, given in [SKY00]:

$$\max\left\{\left\lceil\frac{j}{b_1}\right\rceil p_1 + \sum_{i=2}^{k-1} p_i + \left\lceil\frac{n-j+1}{b_k}\right\rceil p_k + \sum_{i=k+1}^{m} p_i \;\middle|\; j\in[n], k\in[m]\right\}. \tag{8.5}$$

We show that (8.4) dominates this bound. Observe that both bounds are obtained by taking a maximum over integer linear combinations of the processing times. Hence, it suffices to prove that every combination occurring in (8.5) is feasible for the knapsack problem (8.4). Thus, let $j \in [n]$ and $k \in [m]$ be arbitrary. Using that $\lceil\frac{r}{s}\rceil \leq \frac{r+s-1}{s}$ for $r, s \in \mathbb{N}$, it holds that

$$\begin{aligned}
&\left(\left\lceil\frac{j}{b_1}\right\rceil - 1\right) b_1 + \left(\left\lceil\frac{n-j+1}{b_k}\right\rceil - 1\right) b_k \\
&\quad\leq \left(\frac{j+b_1-1}{b_1} - 1\right) b_1 + \left(\frac{n-j+1+b_k-1}{b_k} - 1\right) b_k \\
&\quad= (j-1) + (n-j) \\
&\quad= n-1.
\end{aligned}$$

Hence, the coefficients satisfy our knapsack constraint, which proves that our bound is not smaller than (8.5). Moreover, we can compute it in time $\mathcal{O}(nm)$ by the recursive definition of $c^*_{ij}$. Since the computation of (8.5) takes the same asymptotic runtime, our improvement of the bound comes along with no increase in computational cost.

## 8.2. The Full-Batch Algorithm

Our next goal is to investigate easy scheduling rules for a PFB. We examine whether these rules are constant factor approximations for different objective functions. Again we suppose that Assumption 3.21 holds and focus on schedules ordered by the identity permutation. For each batch on each machine, the main scheduling decision is how long to wait until the batch is started. Waiting for a longer time delays the jobs which would

have been available earlier, but it may allow the use of larger batches, reducing the completion times of jobs that arrive later at this machine. This section and the next one deal with two extreme strategies: Always waiting until a full batch of jobs is available, unless there are no more jobs, and immediately starting a batch when jobs are present and the machine is idle. It turns out that the first strategy can be arbitrarily bad, which we show in this section, while the second strategy yields a 2-approximation, which will be the topic of the subsequent section.

**Definition 8.8.** The *Full-Batch algorithm* for scheduling a PFB is defined by the following rule: On each machine use a LOE-batching in a batch-active way.

**Theorem 8.9.** *There exists no $\alpha \geq 1$ such that the Full-Batch algorithm is an $\alpha$-approximation for minimizing the makespan in a PFB.*

*Proof.* Let $\alpha \geq 1$. Without loss of generality, let $\alpha$ be integer. Construct a PFB instance without release dates as follows. Set $m = 10\alpha$, $n = 5\alpha$, $p_{2i} = b_{2i} = 1$, $p_{2i-1} = 2$ and $b_{2i-1} = n$ for $i \in [5\alpha]$.

Let $S$ be the schedule produced by the Full-Batch algorithm. Figure 8.2 illustrates $S$ on the first seven machines in the case $\alpha = 1$, i.e. $n = 5$. An odd machine waits until all jobs have been finished on the previous even machine, before it processes all of them in a single batch. This way, we obtain $C_{\max}(S) = 10\alpha + 5\alpha n = 10\alpha + 25\alpha^2$, where the first summand stems from the $5\alpha$ odd machines and the second summand from the $5\alpha$ even machines.

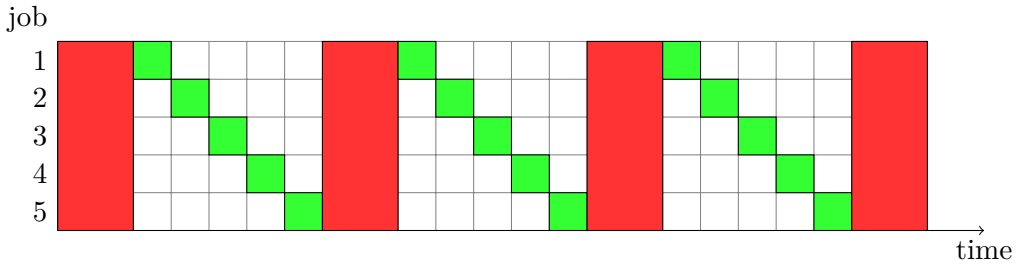


Figure 8.2.: Schedule $S$ up to the seventh machine produced by the Full-Batch algorithm in the case $\alpha = 1$, i.e. $n = 5$.

In contrast, let $S'$ be the schedule which does the following. All batches consist of a single job only. On the first machine, each job is started two time steps after the previous job, i.e. job $j$ is started at time $2j - 2$. On all remaining machines, jobs can be started immediately upon arrival, since no processing time is larger than two. Figure 8.3 illustrates $S'$ on the first seven machines in the case $\alpha = 1$, i.e. $n = 5$. We obtain $C_{\max}(S') = 2n - 2 + 15\alpha = 25\alpha - 2$, because it takes $2n - 2$ time steps until the last job is started on the first machine, and $15\alpha$ more time steps for processing the last job on all machines.

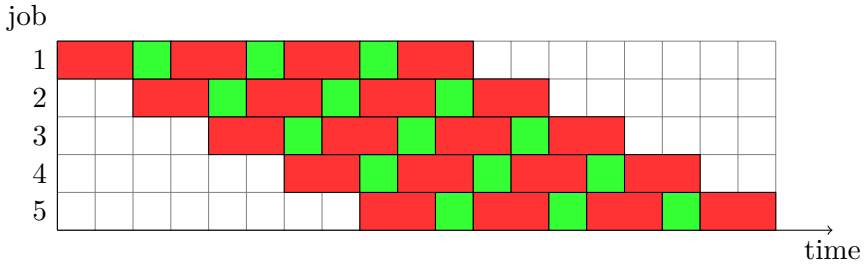


Figure 8.3.: Schedule $S'$ up to the seventh machine produced by using only batches of size one in the case $\alpha = 1$, i.e. $n = 5$.

Hence, in total,

$$\frac{C_{\max}(S)}{\min_{S^*} C_{\max}(S^*)} \geq \frac{C_{\max}(S)}{C_{\max}(S')} = \frac{10\alpha + 25\alpha^2}{25\alpha - 2} > \frac{25\alpha^2}{25\alpha} = \alpha.$$

Therefore, the Full-Batch algorithm cannot be an $\alpha$-approximation for any constant $\alpha \geq 1$. □

**Remark 8.10.** It can be shown that the Full-Batch algorithm is a $\min\{m+1, n\}$-approximation for minimizing the makespan. Moreover, if there are no release dates, this even improves to a $\min\{m, n\}$-approximation. We prove these two facts in the appendix in Section A.4. This also shows that the example in the previous theorem only works because $n$ and $m$ grow with increasing $\alpha$.

The schedule produced by the Full-Batch algorithm can be computed in $\mathcal{O}(mn)$ time in a machine-wise manner.

## 8.3. The Never-Wait Algorithm: A 2-Approximation

In the last section we saw that the Full-Batch algorithm can have an arbitrarily bad approximation ratio. Now we study the opposite strategy, which is to start a batch as soon as possible. It turns out that this yields a 2-approximation for several objective functions including the makespan. Again, we stick to PFB variants that satisfy Assumption 3.21, allowing us to focus on schedules ordered by the identity permutation.

**Definition 8.11.** The *Never-Wait algorithm* for scheduling a PFB is defined as follows: Each machine starts a new batch as soon as it is idle and there are jobs available at the machine. The size of the batch will be as large as possible, i.e. the minimum of the batch capacity of the machine and the number of available jobs at this time.

The completion times produced by the Never-Wait algorithm can be bounded in terms of the lower bound of Section 8.1, as shown by the next theorem.

**Theorem 8.12.** *Consider a PFB variant satisfying Assumption 3.21. For $i \in [m]$ and $j \in [n]$, let $c_{ij}$ be the completion time of job $j$ on machine $i$ resulting from the Never-Wait algorithm and $c^*_{ij}$ be the bound of Section 8.1. Then it follows that*

$$c_{ij} \leq c^*_{ij} + \sum_{i'=1}^{i} p_{i'}.$$

*Proof.* We use a simultaneous induction on $i$ and $j$. The induction start for $i = 0$ is settled by $c_{0j} = r_j = c^*_{0j}$.

Now let $i \in [m]$ and let $j \in [n]$ be a job which is scheduled in the first batch on machine $i$ by the Never-Wait algorithm. Then it holds $c_{ij} = c_{i1} = c^*_{i1} \leq c^*_{ij}$ where the second equality follows because the first job is processed continuously without breaks between machines in the Never-Wait algorithm, which implies that its completion time is optimal.

Finally, let $i \in [m]$ and let $j \in [n]$ be a job which is not scheduled in the first batch on machine $i$ by the Never-Wait algorithm. Let $B_j$ be the batch containing $j$ and $B_j^-$ be the directly preceding batch on machine $i$. We distinguish two cases:

**Case 1**: $j$ was not available at machine $i$ when $B_j^-$ was started. If $j$ arrives at $i$ while $B_j^-$ is processed, then $B_j$ will be started immediately at the time of completion of $B_j^-$. Otherwise, due to definition of the Never-Wait algorithm, $B_j$ will be started exactly at the time when $j$ arrives. In both cases it holds

$$c_{ij} \leq c_{(i-1)j} + 2p_i \overset{\text{ind.}}{\leq} c^*_{(i-1)j} + p_i + \sum_{i'=1}^{i} p_{i'} \leq c^*_{ij} + \sum_{i'=1}^{i} p_{i'}.$$

**Case 2**: $j$ was already available at machine $i$ when $B_j^-$ was started. Then the only reason for $j$ not being included in $B_j^-$ is that the batch capacity of machine $i$ was reached. Hence, $B_j^-$ must be a full batch, which implies that $j - b_i \in B_j^-$. Moreover, since $j$ is available, there cannot be any waiting time between $B_j^-$ and $B_j$. Hence,

$$c_{ij} = c_{i(j-b_i)} + p_i \overset{\text{ind.}}{\leq} c^*_{i(j-b_i)} + p_i + \sum_{i'=1}^{i} p_{i'} \leq c^*_{ij} + \sum_{i'=1}^{i} p_{i'}. \qquad \square$$

Using this, we obtain that the Never-Wait algorithm is a 2-approximation for several PFB variants.

**Corollary 8.13.** *The Never-Wait algorithm is a 2-approximation for the following PFB variants:*

- $F \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid C_{\max}$
- $F \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid F_{\max}$

- $F \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid \sum C_j$
- $F \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid \sum F_j$
- $F \mid p_{ij} = p_i, p\text{-batch}, b_i \mid \sum w_j C_j$
- $F \mid p_{ij} = p_i, p\text{-batch}, b_i \mid \sum w_j F_j$

*Proof.* First, note that all these variants satisfy Assumption 3.21. Let $S^*$ be a schedule ordered by the identity permutation that is optimal for one of those variants. Let $S$ be the schedule produced by the Never-Wait algorithm. Using Theorem 8.12 it follows for a fixed job $j \in [n]$ that

$$\begin{aligned} C_j(S) &\leq c^*_{mj} + \sum_{i=1}^{m} p_i \\ &\leq C_j(S^*) + C_j(S^*) = 2C_j(S^*). \end{aligned}$$

Similarly, for the flow times we obtain

$$\begin{aligned} F_j(S) &= C_j(S) - r_j \\ &\leq c^*_{mj} + \sum_{i=1}^{m} p_i - r_j \\ &\leq C_j(S^*) - r_j + \sum_{i=1}^{m} p_i \\ &\leq F_j(S^*) + F_j(S^*) = 2F_j(S^*). \end{aligned}$$

These two inequalities prove the claim for all six problem variants. □

As for the Full-Batch algorithm, the schedule produced by the Never-Wait algorithm can be computed in $\mathcal{O}(mn)$ time in a machine-wise manner.

The next example shows that the Never-Wait algorithm does not have a better approximation ratio than 2.

**Example 8.14.** We show that there is no $\varepsilon > 0$ such that the Never-Wait algorithm is a $(2-\varepsilon)$-approximation for the makespan.

For some $\varepsilon > 0$, consider the instance given by $n = 2$, $m = 2$, $b_1 = p_1 = 1$, $b_2 = 2$ and $p_2 = 3\lceil \frac{1}{\varepsilon} \rceil$ without release dates. The Never-Wait algorithm schedules two separate batches on the second machine, resulting in $c_{22} = 1 + 2p_2$. The corresponding schedule in the case $\varepsilon = 1$ is illustrated on the left side of Figure 8.4. On the other hand, if a single batch is used on the second machine, we obtain $c_{22} = 2 + p_2$, as illustrated on the right side of Figure 8.4. Hence,

$$\frac{c_{22}}{c^*_{22}} \geq \frac{1 + 2p_2}{2 + p_2} = 2 - \frac{3}{2 + p_2} > 2 - \frac{1}{\lceil \frac{1}{\varepsilon} \rceil} \geq 2 - \varepsilon.$$

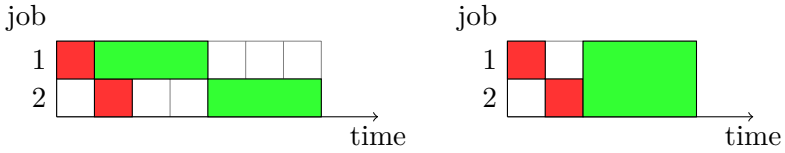


Figure 8.4.: Schedules produced by the Never-Wait algorithm (left) and by using a single batch on the second machine (right) in the case $\varepsilon = 1$ in Example 8.14.

## 8.4. Asymptotic 1-Approximation of the Makespan for $n \to \infty$

In this section we demonstrate that the approximation ratio of the Never-Wait algorithm becomes arbitrarily good (i.e. close to 1), if the number of jobs tends to infinity.

Consider a PFB instance with a large number $n$ of jobs. Let $i \in [m]$ be an arbitrary machine. The optimal makespan must be at least $\lceil \frac{n}{b_i} \rceil p_i$. This number becomes arbitrarily large, if $n$ tends to infinity.

In contrast, observe that the absolute gap between the completion time of a job in the schedule produced by the Never-Wait algorithm and the optimal completion time of this job is at most $\sum_{i=1}^{m} p_i$ by Theorem 8.12. The same holds for the absolute makespan gap of the Never-Wait algorithm. This bound on the gap stays constant, even when $n$ tends to infinity.

Putting these things together, we obtain that, with respect to the makespan, the approximation ratio of the Never-Wait algorithm tends to 1, if $n$ tends to infinity.

# 9. An Online Version: Jobs Arriving over Time

In practice, scheduling decisions must often be made without knowledge about future job arrivals. Therefore, in this chapter we consider an online version of a PFB. We show results from the literature about a single batching machine and develop an online algorithm for a two-machine PFB that has the best possible competitive ratio. For a survey about online scheduling on single and parallel batching machines we refer to [TFY14].

We assume that a job $j$ is unknown to the algorithm until its release date $r_j$. In particular, the total number $n$ of jobs remains unknown until the end of the scheduling process. Again we focus on problem variants that satisfy Assumption 3.21, which allows us to restrict ourselves to schedules ordered by the identity permutation. We also assume that this orders the jobs such that $r_1 \leq r_2 \leq \cdots \leq r_n$.

The standard tool to measure the performance of online algorithms is *competitive analysis*, compare e.g. [BEY98]. This concept is introduced in the following definition.

**Definition 9.1.** A deterministic online algorithm $ALG$ is called $\alpha$*-competitive* for some $\alpha \geq 1$, if it satisfies $\frac{ALG(I)}{OPT(I)} \leq \alpha$ for all instances $I$, where $ALG(I)$ is the objective value of the solution produced by $ALG$ and $OPT(I)$ is the objective value produced by an optimal offline algorithm. The *competitive ratio* of $ALG$ is defined as the infimum over all $\alpha$ such that $ALG$ is $\alpha$-competitive.

Using the results of Section 8.3, we obtain:

**Theorem 9.2.** *The Never-Wait algorithm is 2-competitive with respect to* $C_{\max}$, $F_{\max}$, $\sum C_j$ *and* $\sum F_j$.

*Proof.* Observe that the Never-Wait algorithm is indeed an online algorithm. The 2-competitiveness follows from Corollary 8.13. □

**Remark 9.3.** Note that the considered online version is not meaningful without release dates. Therefore, the objective functions $\sum w_j C_j$ and $\sum w_j F_j$ are not included in the previous theorem. The PFB variants with release dates to minimize these two objective functions do not satisfy Assumption 3.21.

Observe that Theorem 9.2 and Example 8.14 show that the competitive ratio of the Never-Wait algorithm is exactly 2 for makespan minimization. Our next goal is to investigate whether we can obtain a lower competitive ratio with a different algorithmic strategy. First, we give a lower bound.

**Theorem 9.4.** *There is no deterministic online algorithm for makespan minimization in a PFB with a competitive ratio less than the golden ratio* $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$.

*Proof.* For a single batching machine with identical processing times, this claim is proven in [DPZ03, Theorem 3.1]. Note that in [ZCW01, Theorem 1] a similar proof is given that requires non-identical processing times. These proofs can be extended to a PFB with arbitrarily many machines by using tiny processing times on all machines except for the first one. □

Fang et al. [FLL11, Section 2] present an algorithm for a single batching machine that is $\varphi$-competitive, if all job-processing times are *grouped*, i.e. lie in an interval $[p, \varphi p]$ for some $p \in \mathbb{R}_{>0}$. Since this setting generalizes job-independent processing times, the same result holds for a single batching machine with job-independent processing times. In this case, the algorithm can be simplified as follows:

**Definition 9.5.** For a single batching machine with a job-independent processing time $p_1$, the *$\varphi$-Wait algorithm* is defined as follows: It does nothing until time $\frac{1}{\varphi}p_1 = (\varphi - 1)p_1 = \frac{\sqrt{5}-1}{2}p_1 \approx 0.618p_1$. After this point in time, it behaves like the Never-Wait algorithm.

We show that this algorithm is $\varphi$-competitive, using the lower bound $c^*$ proven in Section 8.1:

**Theorem 9.6.** *For a single batching machine with batch capacity $b_1$ and a job-independent processing time $p_1$, the $\varphi$-Wait algorithm is $\varphi$-competitive with respect to each of the two objective functions $C_{\max}$ and $\sum C_j$.*

*Proof.* In analogy to the flow shop notation, we denote the completion time of job $j \in [n]$ by $c_{1j}$, although we only deal with a single machine. Let $j \in [n]$ be arbitrary. Similar to the Never-Wait algorithm (see Theorem 8.12) one can inductively show that $c_{1j} \leq c^*_{1j} + p_1$. If $j$ is scheduled in the first batch, then the construction of the algorithm ensures $c_{1j} = \max\{\varphi p_1, r_j + p_1\} \leq \varphi c^*_{1j}$. If $j$ is not scheduled in the first batch, this can have two reasons. Either $r_j \geq (\varphi - 1)p_1$, or $j$ does not fit into the first batch due to $j > b_1$. In both cases it follows that $c^*_{1j} \geq \varphi p_1$. Then we obtain

$$c_{1j} \leq c^*_{1j} + p_1 \leq c^*_{1j} + \frac{1}{\varphi}c^*_{1j} = \varphi c^*_{1j}.$$

Having proven this property for all $j \in [n]$, the $\varphi$-competitiveness follows for $C_{\max}$ and $\sum C_j$. □

Now we construct an online algorithm for a two-machine PFB that is $\varphi$-competitive. Let $t = \varphi p_1 + (\varphi - 1)p_2$. This is the latest possible time at which the second machine must process its first batch, if we want it to complete a job released at time zero at time $\varphi(p_1 + p_2)$, which is the minimal completion time of such a job multiplied with $\varphi$. The idea of the following algorithm is to schedule machine 1 in a way such that as many jobs as possible have completed it at time $t$, while machine 2 waits until time $t$ and behaves like the Never-Wait algorithm after it.

**Definition 9.7.** For a 2-machine PFB the *t-Switch* algorithm is defined as follows: Machine 1 starts a batch of as many jobs as possible at each of the following points in time: $t - \lfloor \frac{t}{p_1} \rfloor p_1, \ldots, t - p_1, t, t + p_1, \ldots$. We refer to these points in time as *starting instants*. If there are no jobs available at a starting instant, machine 1 does not start a batch and stays idle until the next starting instant, even if jobs arrive in the meantime. Machine 2 stays idle until time $t$ and behaves like the Never-Wait algorithm afterwards.

Next, we prove some lemmas that help to show $\varphi$-competitiveness. Let $c_{1j}$ and $c_{2j}$, $j \in [n]$, be the completion times produced by the $t$-Switch algorithm and let $c^*_{1j}$, $c^*_{2j}$, $j \in [n]$, be the lower bound of Section 8.1.

**Lemma 9.8.** *For all $j \in [n]$, it holds that $c_{1j} \leq c^*_{1j} + p_1$.*

*Proof.* We use induction on $j$. The proof is similar to the proof of Theorem 8.12. Since machine 1 starts a batch regularly in intervals of $p_1$ time, the claim holds for all $j$ that belong to the first batch processed on machine 1. Now let $j$ be a job that is not processed in the first batch. Let $B_j$ be the batch that contains $j$ and $B_j^-$ be the directly preceding batch. We distinguish two cases:

**Case 1:** Job $j$ has not been released when $B_j^-$ was started. In this case, $B_j$ is started at the first starting instant $s$ that satisfies $s \geq r_j$. Hence, $c_{1j} = s + p_1 \leq r_j + 2p_1 \leq c^*_{1j} + p_1$.

**Case 2:** Job $j$ has already been released when $B_j^-$ was started. Then $B_j^-$ must have been a full batch, otherwise $j$ would have been included in it. Moreover, since $j$ is available when $B_j^-$ finishes, $B_j$ is started directly at the next starting instant after $B_j^-$. Hence, we obtain $c_{1j} = c_{1(j-b_1)} + p_1 \leq c^*_{1(j-b_1)} + 2p_1 \leq c^*_{1j} + p_1$ by induction. □

**Lemma 9.9.** *For all $j \in [n]$ with $c_{1j} \leq t$, it holds that $c_{2j} \leq c^*_{2j} + (\varphi - 1)(p_1 + p_2)$.*

*Proof.* First note that machine 2 starts working exactly at time $t$, if there is at least one job satisfying the assumtion of this lemma. We use induction on $j$. If $j$ belongs to the first batch processed on machine 2, it follows that

$$c_{2j} = t + p_2 = \varphi(p_1 + p_2) \leq c^*_{2j} + (\varphi - 1)(p_1 + p_2).$$

If $j$ does not belong to the first batch on machine 2, then all previous batches must have been full. Therefore, induction yields

$$c_{2j} = c_{2(j-b_2)} + p_2 \leq c^*_{2(j-b_2)} + p_2 + (\varphi - 1)(p_1 + p_2) \leq c^*_{2j} + (\varphi - 1)(p_1 + p_2). \qquad \square$$

**Lemma 9.10.** *For all $j \in [n]$, it holds $c_{2j} < c^*_{2j} + p_1 + p_2$.*

*Proof.* We use induction on $j$. Let $j$ be a job that belongs to the first batch processed on machine 2. If $c_{1j} \leq t$, then the claim for this $j$ follows by Lemma 9.9. If $c_{1j} > t$, then machine 2 already behaves like the Never-Wait algorithm when $j$ arrives. Hence, $j$ is started directly upon its arrival on machine 2, and the claim follows by Lemma 9.8.

After the first batch has been started, machine 2 behaves like the Never-Wait algorithm. Therefore, the induction step for jobs $j$ that do not belong to the first batch can be conducted as in Theorem 8.12, using Lemma 9.8. □

Now we are ready to prove $\varphi$-competitiveness of the $t$-Switch algorithm.

**Theorem 9.11.** *For a two-machine PFB, the $t$-Switch algorithm is $\varphi$-competitive with respect to the two objective functions $C_{\max}$ and $\sum C_j$.*

*Proof.* Consider a job $j \in [n]$. We distinguish two cases:

**Case 1:** $c^*_{1j} < t$. Using Lemma 9.8, it follows that $c_{1j} < t + p_1$. Since $c_{1j}$ must be a starting instant, we even obtain $c_{1j} \leq t$. Now Lemma 9.9 yields

$$c_{2j} \leq c^*_{2j} + (\varphi - 1)(p_1 + p_2) \leq c^*_{2j} + (\varphi - 1)c^*_{2j} = \varphi c^*_{2j}.$$

**Case 2:** $c^*_{1j} \geq t$. Then it follows that $c^*_{2j} \geq t + p_2 = \varphi(p_1 + p_2)$. Using Lemma 9.10, we obtain

$$c_{2j} \leq c^*_{2j} + p_1 + p_2 \leq c^*_{2j} + \frac{1}{\varphi} c^*_{2j} = \varphi c^*_{2j}.$$

Having proven this property for all $j \in [n]$, the $\varphi$-competitiveness follows for $C_{\max}$ and $\sum C_j$. □

In summary, we have presented $\varphi$-competitive algorithms for a PFB with one or two machines. For more machines, the 2-competitive Never-Wait algorithm is the best algorithm known. In all of those cases, $\varphi$ is a lower bound for the competitive ratio. We conjecture that this lower bound can be met by a carefully designed algorithm for a PFB with arbitrarily many machines.

# 10. Conclusions and Future Work

Motivated by a novel application area in modern pharmaceutical production, we studied the problem of scheduling a proportionate flow shop of batching machines (PFB) in this thesis. Except for one article by Sung et al. [SKY00] proposing heuristic methods, previous work on this subject was limited to flow shops consisting of at most three machines.

Many of the approaches discussed in this thesis rely on the fact that there is an optimal solution with a fixed job permutation throughout the entire flow shop. This is covered by the concept of permutation schedules that also plays an important role in [SKY00] and other flow shop literature. Therefore, in Chapter 3, we classified different PFB variants according to whether an optimal permutation schedule always exists and whether it is easy to find an optimal job permutation. Additional to these findings, we gave an idea of how to establish this for $\sum U_j$ in Corollary 5.6. An interesting open question is how to find optimal job permutations for a PFB without release dates to minimize $\sum w_j U_j$ and $\sum w_j T_j$.

Concerning the complexity of scheduling a PFB, this thesis extends the state-of-the-art in several ways. First of all, in Chapter 5 we showed that several PFB variants can be optimally scheduled in polynomial time for an arbitrary, but fixed, number of machines $m$, compare Corollary 5.5. For many other variants, it remains open whether this is possible.

Secondly, in Chapter 6 we presented the Efficient-Partial-Schedule algorithm that can be used to find an optimal schedule for a PFB with any regular objective function. In Chapter 7, we showed that this algorithm runs in polynomial time for $n = 2$ jobs. We conjecture that this is true for any constant number of $n$ jobs and leave it as an open question to prove or disprove this.

Thirdly, we pointed out that some PFB variants are high-multiplicity problems, making it even harder to find polynomial algorithms for them. In Section 7.3 we proved that the specific high-multiplicity problem to minimize the makespan in a 2-machine PFB without release dates is solvable in polynomial time. Further research could concentrate on other high-multiplicity variants, e.g. a PFB with more machines, or with the objective function $\sum C_j$.

In addition to investigating exact solution methods, we constructed a lower bound for the completion times in a PFB in Chapter 8. We investigated two contrary scheduling rules regarding their performance in terms of approximation ratios. It turned out that always

waiting for a full batch yields no constant approximation guarantee at all. On the other hand, never waiting for more jobs to be included in a batch is a 2-approximation with respect to several objective functions involving completion and flow times. Here, from our perspective, the most interesting open question is whether a better approximation ratio can be achieved, for example by finding some intermediate strategy.

Finally, in Chapter 9, we studied an online version of a PFB in which jobs are not revealed before their release date. We presented online algorithms for a PFB with one or two machines that reach the best possible competitive ratio of $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$. For more machines, there is still a gap to close between $\varphi$ and 2.

Other questions for further research stem from the application in the pharmaceutical industry. First of all, it is possible to place several parallel machines at certain stages in the flow shop, leading to the mathematical concept of a hybrid flow shop. A first strategic question is how many machines should be placed at the stages to make a certain throughput of jobs possible. A second question is how to schedule such a proportionate hybrid flow shop of batching machines and which of the results of this thesis carry over to this setting.

Another area for further research inspired by practical application involves uncertainties. In Chapter 9 we already dealt with the case in which future job arrivals are unknown. In practice, however, processing times might also be uncertain. Moreover, jobs may have to repeat one or several stages in the flow shop, e.g. because they unexpectedly fail a quality test. The first step towards mathematically covering those uncertainties would be to find appropriate models that meet the practical requirements. Possible approaches are paradigms from online, robust, and stochastic optimization, as well as combinations of these.

All in all, we believe that this thesis provides a decent starting point for detailed investigations of how to schedule a PFB. We hope that it encourages the reader to dive into this topic and find out more about it.

# A. Additional Proofs

In this appendix we provide some proofs which have been left out in the main text for the sake of keeping it readable and focused.

## A.1. An Alternative Proof of Lemma 3.11

In Section 3.2, we proved Lemma 3.11 by exploiting bottleneck Monge matrices in the context of a bottleneck assignment problem. The proof we present here does not need these properties. It applies the pigeonhole principle instead.

**Lemma 3.11.** *Let $u, v \in \mathbb{R}^n$ be ordered nondecreasingly and let $\sigma, \pi \in \mathrm{Sym}(n)$. Then it holds that*

*(i)* $\min_{j\in[n]} (v_j - u_j) \geq \min_{j\in[n]} (v_{\pi(j)} - u_{\sigma(j)})$ *and*

*(ii)* $\max_{j\in[n]} (v_j - u_j) \leq \max_{j\in[n]} (v_{\pi(j)} - u_{\sigma(j)})$.

*Alternative Proof.*

(i) Fix an index $j \in [n]$. Let

$$J_1 = \{j' \in [n] \mid \sigma(j') \geq j\}$$

and

$$J_2 = \{j' \in [n] \mid \pi(j') \leq j\}.$$

Then $|J_1| = n - j + 1$ and $|J_2| = j$. Hence, $|J_1| + |J_2| = n + 1$. Therefore, by the pigeonhole principle, $J_1 \cap J_2$ cannot be empty and contains an index $j'$. Using that $u$ and $v$ are ordered nondecreasingly, we obtain $u_j \leq u_{\sigma(j')}$ and $v_j \geq v_{\pi(j')}$. Hence,

$$v_j - u_j \geq v_{\pi(j')} - u_{\sigma(j')} \geq \min_{j\in[n]} (v_{\pi(j)} - u_{\sigma(j)}).$$

The claim follows by taking the minimum over all $j$.

(ii) The second part can be deduced from the first part in the same way as we deduced the first part from the second part in the original proof. □

## A.2. Formal Proof of Correctness for the MIP in Section 4.2

**Lemma 4.1.** *Consider completion times $c_{ij}$, $i \in [m]$, $j \in [n]$. They define a feasible schedule $S$, if and only if there are binary values $z_{ijj'} \in \{0,1\}$, $i \in [m]$, $j, j' \in [n]$, $j \neq j'$, such that (4.2a) to (4.2g) are satisfied.*

*Proof.* Suppose binary values $z_{ijj'} \in \{0,1\}$, $i \in [m]$, $j, j' \in [n]$, $j \neq j'$, exist such that (4.2a) to (4.2g) are satisfied. We show the properties of Definition 2.3. Properties (i) and (ii) are satisfied by (4.2a) and (4.2b).

For Property (iii), assume $c_{ij} < c_{ij'}$ for some $i \in [m]$, $j, j' \in [n]$. Then, $c_{ij'} - c_{ij} > 0$. Hence, by (4.2d), $z_{ijj'} = 1$. Therefore, (4.2c) implies $c_{ij'} - c_{ij} \geq p_i$. In the case $c_{ij} > c_{ij'}$, we analogously obtain $c_{ij} - c_{ij'} \geq p_i$. Thus, if $c_{ij}$ and $c_{ij'}$ are not equal, then they differ by at least $p_i$, which is (iii).

For Property (iv), let $c \in \mathbb{R}$ and $i \in [m]$. If no job finishes machine $i$ at time $c$, then (iv) is trivially satisfied. Otherwise, choose a job $j$ with $c_{ij} = c$. Let $j' \in [n] \setminus \{j\}$. Observe that (4.2c) implies that at most one of the two variables $z_{ijj'}$ and $z_{ij'j}$ can have value one. Therefore, (4.2e) ensures, that there are at least $n - b_i$ jobs $j'$ with $z_{ijj'} + z_{ij'j} = 1$. Moreover, (4.2c) ensures that none of these $n - b_i$ jobs has the same completion time as $j$. Hence, at most $b_i$ jobs complete at time $c = c_{ij}$, which is (iv).

Conversely, suppose the schedule $S$ belonging to the completion times is feasible. Define

$$z_{ijj'} = \begin{cases} 1, & \text{if } c_{ij} < c_{ij'}, \\ 0, & \text{otherwise.} \end{cases} \tag{A.1}$$

Constraints (4.2a) and (4.2b) are satisfied by Properties (i) and (ii) of Definition 2.3.

For (4.2c), consider two cases. Firstly, if $c_{ij} < c_{ij'}$, then by (iii), $c_{ij'} - c_{ij} \geq p_i$ and (4.2c) is satisfied. Otherwise, if $c_{ij} \geq c_{ij'}$, then by (A.1), $z_{ijj'} = 0$ and the constraint is fulfilled due to the big negative constant $M$ on the right-hand side.

Similarly, (A.1) ensures that (4.2d) is satisfied.

Finally, for each job $j$ and each machine $i$ there are at least $n - b_i$ jobs with a different completion time on machine $i$ by Property (iv). For each job $j'$ of those, (A.1) yields that either $z_{ijj'}$ or $z_{ij'j}$ has value one. This ensures (4.2e). □

## A.3. Formal Proof of Correctness for the MIP in Section 4.3

**Lemma 4.2.** *Consider completion times $c_{ij}$, $i \in [m]$, $j \in [n]$. They define a feasible permutation schedule $S$ ordered by the identity function, if and only if there are binary variables $z_{ij} \in \{0,1\}$, $i \in [m]$, $j \in [n-1]$, such that (4.3a) to (4.3g) are satisfied.*

*Proof.* Suppose binary values $z_{ij} \in \{0,1\}$, $i \in [m]$, $j \in [n-1]$ exist such that (4.3a) to (4.3g) are satisfied. Constraint (4.3c) implies $c_{i(j+1)} \geq c_{ij}$. Hence, $S$ must be a permutation schedule ordered by the identity function. It remains to show the properties of Definition 2.3.

Properties (i) and (ii) are satisfied by (4.3a) and (4.3b).

Observe that (4.3c) and (4.3d) imply that $c_{i(j+1)}$ and $c_{ij}$ are equal, if $z_{ij} = 0$, and differ by at least $p_i$, if $z_{ij} = 1$. Together with the fact that $S$ is ordered by the identity function, we obtain (iii).

To prove (iv), fix $c \in \mathbb{R}$ and $i \in [m]$. If no job finishes machine $i$ at time $c$, then (iv) is trivially satisfied. Otherwise, let $j$ be the minimal job such that $c = c_{ij}$. By (4.3e), there must be a $j' \in \{j, j+1, \ldots, j+b_i-1\}$ such that $z_{ij'} = 1$. Then, (4.3c) implies that $c_{i(j'+1)} > c_{ij'} \geq c_{ij}$. Using that $S$ is ordered by the identity function, we obtain that only the jobs $j, j+1, \ldots, j'$ can finish at time $c$. These are at most $b_i$ many, which shows (iv).

Conversely, let $S$ be a feasible permutation schedule ordered by the identity function. Define

$$z_{ij} = \begin{cases} 1, & \text{if } c_{ij} < c_{i(j+1)}, \\ 0, & \text{otherwise.} \end{cases} \tag{A.2}$$

Constraints (4.3a) and (4.3b) are satisfied by Properties (i) and (ii) of Definition 2.3.

If $c_{ij} < c_{i(j+1)}$, then by (iii) $c_{i(j+1)} \geq c_{ij} + p_i$. Hence, (4.3c) is satisfied. Moreover, in this case we have $z_{ij} = 1$, which implies (4.3d) due to the positive big $M$ on the right-hand side.

If $c_{ij} = c_{i(j+1)}$, then $z_{ij} = 0$ and (4.3c) and (4.3d) are satisfied as well.

The case $c_{ij} > c_{i(j+1)}$ cannot occur, because $S$ is a permutation schedule ordered by the identity function. Therefore, (4.3c) and (4.3d) are valid in any case.

For (4.3e), fix $i \in [m]$ and $j \in [n-b_i]$. Consider the $b_i+1$ jobs $j, j+1, \ldots, j+b_i$. Not all of them can have the same completion time on machine $i$ due to (iv). Hence, using that $S$ is a permutation schedule ordered by the identity function, there is a $j' \in \{j, j+1, \ldots, j+b_i-1\}$ with $c_{ij'} < c_{i(j'+1)}$. This implies $z_{ij'} = 1$ and (4.3e) follows. □

## A.4. An Approximation Guarantee for the Full-Batch Algorithm

In this section we show that the Full-Batch algorithm is a $\min\{m+1, n\}$-approximation for minimizing the makespan. Moreover, if there are no release dates, this improves to a $\min\{m, n\}$-approximation. Note that Assumption 3.21 is satisfied for makespan

minimization. Hence, we can restrict ourselves to schedules ordered by the identity permutation. For convenience, write $r_{\max} = \max_{j\in[n]} r_j$.

**Lemma A.1.** *For the schedule $S$ produced by the Full-Batch algorithm it holds that $C_{\max} \leq r_{\max} + \sum_{i=1}^{m} \lceil \frac{n}{b_i} \rceil p_i$.*

*Proof.* Note that $S$ is a batch-active schedule. Hence, in the time period between $r_{\max}$ and $C_{\max}$, there is no time span in which all machines are idle. Each machine $i$ processes exactly $\lceil \frac{n}{b_i} \rceil$ batches in $S$, taking $\lceil \frac{n}{b_i} \rceil p_i$ time. Hence, the claim follows. □

**Theorem A.2.** *The Full-Batch algorithm is an $n$-approximation for minimizing the makespan.*

*Proof.* Let $S$ be the schedule produced by the Full-Batch algorithm and $S^*$ be a schedule minimizing the makespan. Considering the job that is released last, one obtains that $C_{\max}(S^*) \geq r_{\max} + \sum_{i=1}^{m} p_i$. With Lemma A.1 we obtain

$$C_{\max}(S) \leq r_{\max} + \sum_{i=1}^{m} \left\lceil \frac{n}{b_i} \right\rceil p_i \leq r_{\max} + \sum_{i=1}^{m} np_i \leq n \left( r_{\max} + \sum_{i=1}^{m} p_i \right) \leq nC_{\max}(S^*). \qquad \square$$

**Theorem A.3.** *The Full-Batch algorithm is a $\min\{m+1, n\}$-approximation for minimizing the makespan. If there are no release dates, this improves to $\min\{m, n\}$.*

*Proof.* Let $S$ be the schedule produced by the Full-Batch algorithm and $S^*$ be a schedule minimizing the makespan. In $S^*$, each machine $i$ processes at least $\lceil \frac{n}{b_i} \rceil$ batches. Hence, $C_{\max}(S^*) \geq \lceil \frac{n}{b_i} \rceil p_i$. Moreover, it also holds that $C_{\max}(S^*) \geq r_{\max}$. Hence, using Lemma A.1 we obtain

$$C_{\max}(S) \leq r_{\max} + \sum_{i=1}^{m} \left\lceil \frac{n}{b_i} \right\rceil p_i \leq C_{\max}(S^*) + \sum_{i=1}^{m} C_{\max}(S^*) = (m+1)C_{\max}(S^*).$$

Moreover, if there are no release dates, then the summand $r_{\max}$ can be left out, yielding a factor of $m$ instead of $m+1$. The claim follows together with Theorem A.2. □

# B. Bibliography


[AADT92] J. H. Ahmadi, R. H. Ahmadi, S. Dasu, and C. S. Tang. Batching and scheduling jobs on batch and discrete processors. *Operations Research*, 40(4):750–763, 1992.

[Bap00] P. Baptiste. Batching identical jobs. *Mathematical Methods of Operations Research*, 52(3):355–367, 2000.

[BCGvdK05] N. Brauner, Y. Crama, A. Grigoriev, and J. van de Klundert. A framework for the complexity of high-multiplicity scheduling problems. *Journal of Combinatorial Optimization*, 9(3):313–323, 2005.

[BDM09] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, 2009.

[BEY98] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[BGH+98] P. Brucker, A. Gladky, H. Hoogeveen, M. Y. Kovalyov, C. N. Potts, T. Tautenhahn, and S. L. van de Velde. Scheduling a batching machine. *Journal of Scheduling*, 1(1):31–54, 1998.

[BK] P. Brucker and S. Knust. Complexity results for scheduling problems. `http://www.informatik.uni-osnabrueck.de/knust/class/`. Last visited on September 11, 2018.

[Bru07] P. Brucker. *Scheduling Algorithms*. Springer, 5th edition, 2007.

[BS10] U. Buscher and L. Shen. MIP formulations and heuristics for solving parallel batching problems. *Journal of Systems Science and Complexity*, 23(5):884–895, 2010.

[CKS10] A. Condotta, S. Knust, and N. V. Shakhlevich. Parallel batch scheduling of equal-length jobs with release and due dates. *Journal of Scheduling*, 13(5):463–477, 2010.

[DPZ03] X. Deng, C. K. Poon, and Y. Zhang. Approximation algorithms in batch processing. *Journal of Combinatorial Optimization*, 7(3):247–257, 2003.

[Ehr05] M. Ehrgott. *Multicriteria Optimization*. Springer, 2nd edition, 2005.

[FLL11] Y. Fang, P. Liu, and X. Lu. Optimal on-line algorithms for one batch machine with grouped processing times. *Journal of Combinatorial Optimization*, 22(4):509–516, 2011.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GJS76] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[GLLR79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.

[Gri03] A. Grigoriev. *High Multiplicity Scheduling Problems*. PhD thesis, Universiteit Maastricht, 2003.

[HLP52] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, 2nd edition, 1952.

[HS90] D. S. Hochbaum and R. Shamir. Minimizing the number of tardy job units under release time constraints. *Discrete Applied Mathematics*, 28(1):45–57, 1990.

[HS91] D. S. Hochbaum and R. Shamir. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Operations Research*, 39(4):648–653, 1991.

[HW76] D. S. Hirschberg and C. K. Wong. A polynomial-time algorithm for the knapsack problem with two variables. *Journal of the ACM*, 23(1):147–154, 1976.

[IG86] Y. Ikura and M. Gimple. Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 5(2):61–65, 1986.

[LUMV92] C.-Y. Lee, R. Uzsoy, and L. A. Martin-Vega. Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775, 1992.

[MS06] M. Mathirajan and A. I. Sivakumar. A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology*, 29(9):990–1001, 2006.

[NPW] T. Noe, T. I. Piezas, and E. W. Weisstein. Fibonacci n-step number. From MathWorld – A Wolfram Web Resource. `http://mathworld.wolfram.com/Fibonaccin-StepNumber.html`. Last visited on July 25, 2018.

[PK00] C. N. Potts and M. Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228–249, 2000.

[PSK13] S. S. Panwalkar, M. L. Smith, and C. Koulamas. Review of the ordered and proportionate flow shop scheduling research. *Naval Research Logistics (NRL)*, 60(1):46–55, 2013.

[RVR10] R. Ruiz and J. A. Vázquez-Rodríguez. The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1):1–18, 2010.

[SK03] C. S. Sung and Y. H. Kim. Minimizing due date related performance measures on two batch processing machines. *European Journal of Operational Research*, 147(3):644–656, 2003.

[SKY00] C. S. Sung, Y. H. Kim, and S. H. Yoon. A problem reduction and decomposition approach for scheduling for a flowshop of batch processing machines. *European Journal of Operational Research*, 121(1):179–192, 2000.

[SY97] C. S. Sung and S. H. Yoon. Minimizing maximum completion time in a two-batch-processing-machine flowshop with dynamic arrivals allowed. *Engineering Optimization*, 28(3):231–243, 1997.

[TFY14] J. Tian, R. Fu, and J. Yuan. Online over time scheduling on parallel-batch machines: A survey. *Journal of the Operations Research Society of China*, 2(4):445–454, 2014.

[Wol98] L. A. Wolsey. *Integer Programming*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley & Sons, 1998.

[ZCW01] G. Zhang, X. Cai, and C. K. Wong. On-line algorithms for minimizing makespan on batch processing machines. *Naval Research Logistics*, 48(3):241–258, 2001.

# C. Acknowledgments

Many people contributed directly or indirectly to the success of this thesis.

First of all, I would like to thank my supervisors Prof. Sven O. Krumke, Dr. Heiner Ackermann, Dr. Christian Weiß, and Dr. Sandy Heydrich. All of them participated enthusiastically in the discussions during our weekly meetings. This has been a key ingredient for the success of this thesis. In particular, I thank Prof. Sven O. Krumke for his great personal support during the last few years and his extraordinary commitment to all aspects of teaching at TU Kaiserslautern. Moreover, special thanks go to Dr. Heiner Ackermann who originally proposed this thesis topic and has been an excellent mentor for me at Fraunhofer ITWM.

I also would like to thank Oliver Bachtler, Tim Bergner, and Jens-Peter Joost for being part of our "in-official master thesis seminar". I received a lot of input and helpful comments in this context and was always impressed to follow the progress of the other thesis projects.

Furthermore, I am grateful to my proofreaders Julia Amann, Oliver Bachtler, Tim Bergner, Sandy Heydrich, and Christian Weiß for their invested time and helpful comments.

And finally, many thanks to Julia, to my family, friends, and fellow students for all the support I received on the way to my master's degree.

# D. Selbständigkeitserklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen als Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Christoph Hertrich
Kaiserslautern, den 12. Oktober 2018.