Machine Scheduling Problems with Position-Dependent Availability Constraints

Andreas Hipp

Machine Scheduling Problems with Position-Dependent Availability Constraints

Dissertation

zur Erlangung des akademischen Grades

Doktor rerum politicarum (Dr. rer. pol.)

vorgelegt von

Andreas Walter Hipp

geb. in Oberhausen

Erstgutachter: Prof. Dr. Florian Jaehn

Zweitgutachter: Prof. Dr. Dominik Kreß

Tag der Disputation: 15.05.2024

Fakultät für

Wirtschafts- und Sozialwissenschaften

Professur für Betriebswirtschaftslehre,

insb. Operations Research und Management Science

HELMUT SCHMIDT UNIVERSITÄT

Universität der Bundeswehr Hamburg

In Memoriam

Prof. Dr. Rainer Leisten

„Ich weiß, dass ich nichts weiß.“

Sokrates (469 - 399 v. Chr.)

Acknowledgment

Mein Dank gilt zuallererst Prof. Dr. Florian Jaehn, meinem Betreuer und Chef, der es mir ermöglicht hat, mein Ziel der Promotion weiter zu verfolgen. Ebenso hat mich Prof. Dr. Dominik Kreß vom Beginn an stets fachlich und persönlich in meinem Vorhaben unterstützt, meine Dissertation erfolgreich zu beenden. Ohne meine Kolleg*innen Nicolas Fauvé, Finn Meissner, Frank Wiedra und Katrin Hofmann hätte es aber letztendlich niemals funktioniert. Vielen Dank für Eure Unterstützung! An dieser Stelle muss auch Prof. Dr. Rainer Leisten erwähnt werden, der mir damals an der Universität Duisburg-Essen den Weg zur Promotion ebnete. Ich werde Ihnen für immer dankbar sein!

Außerhalb des universitären Umfelds sind meine Freunde für die Unterstützung in all den Jahren hervorzuheben – nicht nur auf diese Arbeit bezogen, sondern auf alles, was ich tue und bin. Vielen Dank, dass ihr immer für mich da seid Cathi, Hanna, Kathrin, Miriam, Moritz, Scarlett und Thomas!

Zu guter Letzt konnte ich in all der Zeit auf die volle Rückendeckung meiner Familie zählen, die mich darin bestärkt hat zu promovieren. Den Stellenwert, den ihr für mich habt, kann ich nicht in Worte fassen. Danke, dass es euch gibt!

Hamburg, März 2024 Andreas Hipp

Abstract

Machine scheduling problems can be found in any practical environment from logistics to manufacturing, where tasks, called jobs, are completed by assigning them to resources, called machines. In this thesis, we assume that the execution of jobs deteriorates the machine to the point where it may not be in a state to process further jobs, or, more generally, restricts the availability of the machine. Time is often considered to be the main factor in machine deterioration, such as the time elapsed since the start of planning. However, with regard to practice, there are other important factors besides time that can have a significant impact on the machine state. In this work, we concentrate on factors based on the job positions in the scheduling sequence, summarized as position-dependent machine availability constraints. This characteristic can be found in any production or logistics environment where the assignment of the job to the machine determines the availability of the machine to process jobs, rather than the processing time.

First, a definition and the existing literature of the superordinate research field of problems with position-dependent availability is given: state-dependent machine availability. In the following, single machine scheduling problems with position-dependent maintenance or with jobs that have to be sequenced at fixed positions are analyzed regarding their complexity. These assumptions represent specific characteristics of position-dependent availability in scheduling. For each assumption, complexity results are presented for single machine scheduling problems minimizing completion time or due date related objective functions. In addition, polynomial-time algorithms are formulated for specific scheduling problems. Finally, we deal with the hybrid flow shop scheduling (HFS) problem that is characterized by a flow shop layout with at least one stage of processing with parallel machines. Priority rules and constructive heuristics are tested and evaluated on 1260 instances for the two-stage no-wait HFS problem with two identical machines on the first stage and one machine on the second stage, which is known to be $\mathcal{NP}$-hard. Furthermore, HFS problems with unrelated machines are assumed, considering setup times on the machines caused by different job families. Constructive heuristics are tested on 960 instances to study the influence of the number of setups on the results for completion time related objective functions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sequencing and *scheduling* are crucial parts of any regular decision making process in manufacturing and service industries. Until the 1990s, it was common to distinguish between two classic trends: Project Planning and Machine Scheduling (Pinedo (2022)). While project planning was primarily characterized by precedence constraints between project activities of a superordinate project[1] assuming no lack of resources, machine scheduling displayed specific scheduling problems depending on the machine and job situation. Since models nowadays focus on resource constraints in project planning and have high mathematical complexity and a more general validity in machine scheduling, both areas are being brought much closer together (Brucker and Knust (2012)).

As a result, scheduling includes any type of job-to-machine assignment under predefined job and/or machine restrictions to optimize one or more given objectives (Blazewicz et al. (2019)). A job is any type of resource-consuming task that has to be completed on that resource, here called a machine.

Sequencing, as part of the scheduling process, is the step of determining the exact order in which to assign each job to the available machine(s). If more than one machine is available in the production environment, a machine has to be selected for each job in the first step to determine the job order for each machine in the second step (Pinedo (2022)).

We do not claim to cover the entire area of scheduling in this work. For an overview of scheduling research, see, for example, the well-known works of Brucker and Knust (2012), Framiñán et al. (2014), Jaehn and Pesch (2019), or Pinedo (2022). Demeulemeester and

1 for a formal definition, see the DIN ISO definition in DIN (2015)

Herroelen (2006) concentrate on specific project scheduling models, Hopp and Spearman (2011) classify scheduling in the overall research field of Operations Management.

While numerous job and machine constraints are studied in scheduling, we concentrate on one type of machine restriction that is part of one of the newer areas of Operations Research and has not received the same attention as other scheduling areas so far (Sanlaville and Schmidt (1998)): Machine availability constraints. When we talk about machine availability, we mean any type of machine restriction regarding the availability of the machine to execute jobs, i.e., deterioration caused by assigned jobs that leads to wear and tear on the respective machine. This wear and tear can be "healed" by performing specific operations, referred to as maintenance operations, to improve the state of the machine and ensure its functionality to perform jobs. Since the late 1980s, researchers have been studying different types of maintenance and machine availability constraints (Sanlaville and Schmidt (1998)). With regard to maintenance operations, it can be distinguished between scheduling them in advance (see, e.g., Low et al. (2010) and Costa et al. (2016)), the restriction to perform them within defined periods (typical models are presented by Błażewicz et al. (2003) and Perez-Gonzalez and Framiñán (2018)), and completely arbitrary selectable time windows for scheduling maintenance operations – see Kubzin and Strusevich (2006) or Rustogi and Strusevich (2012a).

Besides these and other time-based types of machine deterioration and maintenance planning, a lot of real-world machine environments can be found in which the machine deteriorates because of other factors than time. Looking for instance at the usage of aircraft landing gear, it is obvious that the length of each flight does not influence the wear and tear of the wheels and landing gear, but the take-off and landing procedures themselves. This means that the deterioration, and therefore the maintenance planning, depends mainly on the number of flights and not on their length (see Sriram and Haghani (2003)). Similar characteristics can be found in the wear and tear of aircraft engines, as described by Ruther (2013), or in the wear and tear of batteries in vehicles with combustion engines (see Crompton (2000)).

In this thesis, we concentrate on this case of non time-related factors of wear and tear by assuming that machine deterioration is related to the number of jobs assigned, regardless of their processing times on the machine. This means that each job causes the

same deterioration. After a certain number of jobs have been processed, the wear and tear of the machine prevents further jobs from being executed, and a maintenance operation must be performed to restore the machine's functionality. Maintenance operations are characterized by a fixed length and must be performed when the machine is worn out or at any time before. Regardless of their position in the sequence, maintenance operations completely restore the machine. In the following, we refer to this type of maintenance planning as *position-dependent* (*pd*) maintenance, which was introduced and first studied by Drozdowski et al. (2017). In Operations Research, only a few studies have examined *pd* maintenance planning so far. Therefore, in this work, we concentrate on single machine scheduling problems. Single machine layouts may be the simplest layout in machine scheduling, but are very important at the same time when the focus is on understanding the specifics and properties of problems.

This thesis is divided into three main parts. The first part aims to provide the fundamentals for our research. In Chapter 2, we introduce the main notation that is applied in all the models and solution approaches presented. Our problems are embedded in the well-known Graham notation introduced by Graham et al. (1979). We present here the relevant entries for this notation to clearly identify each scheduling configuration. In addition, a clear and newly composed notation is introduced, which is added to the aforementioned Graham notation in order to properly describe non-time based deterioration and maintenance specifics. In Chapter 3, the topic of non-time based deterioration factors in scheduling is formally elaborated and embedded in the research field of machine availability in scheduling. Existing work as well as the models studied in this thesis are analyzed and put into context.

In the second part, displayed by chapters 4 and 5, we concentrate on single machine models that include *pd* maintenance planning or related availability constraints. Here, we study completion time and due date related objective functions, which are typically examined in machine scheduling because of their theoretical and practical relevance (see, e.g., Framiñán et al. (2014)). To make significant statements about the characteristics of these problems, we focus on their runtime complexity. In Section 4.1, the runtime complexity of a set of single machine scheduling layouts with *pd* maintenance is observed by primarily orienting on the respective cases with continuous machine availability.

In addition, in sections 4.2 and 4.3, we present two dynamic programs to solve specific single machine problems to minimize the sum of weighted completion times and the number of late jobs in polynomial time. The problems are analyzed in terms of their properties for optimal solutions, and the runtime of the algorithms is determined. Numerical examples demonstrate their approach and operating principle. Chapter 5 includes the special case of jobs that have to be scheduled at predefined fixed positions. This characteristic is closely related to the previous case of *pd* maintenance. We again present complexity results for scheduling models including typical job characteristics and objective functions as well as a dynamic program for the single machine layout with fixed job positions and uniform due dates to minimize total weighted late work. Again, the runtime complexity is determined, followed by an example to demonstrate the approach of the algorithm.

In the last part, we leave the field of position-dependent machine availability and address related availability constraints regarding job scheduling. In addition, hybrid flow shop scheduling problems instead of single machine layouts are studied in the remaining chapters. First, Chapter 6 outlines the two-stage hybrid flow shop problem with the no-wait constraint. This means, waiting times of jobs between successive machine stages are not allowed. Since this problem is known to be $\mathcal{NP}$-hard for the objective functions makespan and total completion time, heuristic approaches, such as classical priority rules, are implemented and evaluated. In Chapter 7, we study hybrid flow shop scheduling problems with setup times. Here, jobs belong to classes that require setups to adjust special machine configurations for the job classes. The availability of the machine to perform jobs is therefore limited by the characteristics of the jobs. Since the problem is $\mathcal{NP}$-hard, a constructive heuristic is formulated and evaluated for various objective functions.

Finally, the thesis is completed by a detailed conclusion and outlook for future work in Chapter 8.

Each chapter can be assigned to a paper that has been published, accepted, or is in progress of being submitted. The assignment of these papers to their respective chapters of this thesis is shown in Appendix A.

Chapter 2

Framework

Scheduling problems and their solutions aimed at optimizing one or more objectives consume a lot of time for any decision maker in real-world production environments. Therefore, mathematical formulations displaying the required type of resources, here machines, and jobs together with the appropriate model structure and solution approach are one of the key factors for effective decision making. (Pinedo (2022))

In this chapter, we present the tools that are required in this thesis to deal with a wide range of scheduling problems, to formulate suitable models, and to generate an appropriate approach to finding (optimal) solutions. In Section 2.2, we concentrate on the notation used in the models that are explicitly studied. Where further notation is required, especially in Chapter 3, it is introduced on the spot. Since we focus on the runtime complexity of the models we study, we give a brief overview of complexity in scheduling at the end of this chapter.

2.1 Problem Classification

Typically, scheduling problems can be classified according to the information of the planning horizon, and the nature of the variables.

If all information of the planning horizon – all relevant characteristics of the machines and jobs – is known in advance, i.e. at the beginning of the optimization process, the problem is called *static*. In *dynamic* problems, however, some or all of the information about the parameters is not known in advance. Furthermore, a distinction is made between

deterministic and *stochastic* problems. If no non-deterministic descriptions of parameters are part of the scheduling problem, it is a deterministic problem. Otherwise, it is referred to as a stochastic problem. Here, not all information about the outcome of a decision is available at the time the decision has to be made. (Blazewicz et al. (2019))

As the majority of models in scheduling research are based on static, deterministic approaches (Leisten and Rajendran (2015)), we only consider static, deterministic scheduling problems, too. Furthermore, following basic assumptions hold for all the problems we tackle in the course of this work (Framiñán et al. (2014)):

- The sets of jobs and machines are known in advance.
- No dependencies between jobs are given.
- Each job can only be processed by one machine.
- The set of machines is available for the whole planning horizon.
- Each machine can only process one job at a time.
- Transportation times within the machine environment are negligible.
- The inventory buffers in front of each machine are unlimited.

2.2 Basic Notation

All of the scheduling problems investigated in this thesis are considered to be finite in the number of jobs and machines. In general, the number of jobs is given by $n \in \mathbb{N}$, where j denotes a job. If a machine environment with more than one machine is given, i defines a machine of a set with a total of $m \in \mathbb{N}$ machines.

According to the classification scheme of Graham et al. (1979), deterministic scheduling problems can be described with an $\alpha|\beta|\gamma$ triplet where α defines the machine layout, β the model constraints, and γ the objective function. In accordance with the main topic of this thesis, we add β-features by a four-field tuple $[\delta, \epsilon, \zeta, \eta]$ in our classification scheme as it is done by Briskorn et al. (2024). δ represents the machine state, ϵ the type of deterioration caused by jobs on the machine, the characteristics of maintenance operations are described by ζ, and η displays potential influences of the machine state to

its performance. This notation allows full integration into the notation by Graham et al. (1979) such that $\alpha|\beta,[\delta,\epsilon,\zeta,\eta]|\gamma$ specifies a machine scheduling model with the machine environment according to α, machine availability and maintenance features according to $[\delta,\epsilon,\zeta,\eta]$, further side constraints according to β, and objectives according to γ.

Since we study theoretical models of a group of scheduling problems that, to the best of our knowledge, has not been analyzed intensively so far, we focus on single machine scheduling problems in chapters 4 and 5, represented by $\alpha = 1$. Nevertheless, we also present flow shop, parallel machine problems, and hybrid flow shop problems in chapters 3, 6, and 7. Flow shops are described by $\alpha = Fm$ whereas $m \geq 2$ declares the number of machines in the system, here in the flow shop. A flow shop layout is characterized by a series of machines, in which all jobs have to be processed in the same order regarding the machines. In parallel machine problems, $m \geq 2$ machines are arranged in parallel, but each job can only be assigned to one of them. The machines are distinguished on the basis of their speed, i.e. we can have identical ($\alpha = Pm$), related ($\alpha = Qm$), or unrelated machines ($\alpha = Rm$). Hybrid flow shop settings combine flow shop layouts with sequential machine stages with parallel machine layouts on at least one of the stages. The respective notation is introduced in Chapter 6. For more detailed information about the different types of machine environments, see, e.g., Pinedo (2022).

Dealing with model and job constraints in the scheduling environment, only some of the most typical model constraints are considered in this thesis because of the mentioned reason that only a few scheduling problems with *pd* maintenance or related constraints have been studied so far. All relevant β-field entries are summarized and explained in Table 2.1. For instance, jobs can be characterized by a weight to display their relevance for the studied objective. Further, jobs might have release dates if not all jobs are available at the beginning of the planning horizon. For the case of jobs with fixed positions in the final sequence, studied in Chapter 5, we require the novel β-field entry fp that is introduced by Calinescu et al. (2017) and Jaehn (2024) who are the first to investigate this job feature to the best of our knowledge. The models in chapters 6 and 7 address two specific job characteristics: the no-wait constraint and setup times. $no-wait$ describes the case that jobs are not allowed to wait between subsequent stages of operation. Setup times ST are based on the assumption that jobs can be grouped into job families that require the same

setups on the machine. When two jobs from different families are processed in sequence, a setup must be performed to prepare the machine.

Next, we detail fundamental features in machine scheduling subject to maintenance applying the introduced notation for the β-field of the Graham notation, i.e. the $[\delta, \epsilon, \zeta, \eta]$-tuple. Here we distinguish between how the state of a machine is represented, how this state is deteriorated and maintained, and how it affects the machine's performance. The following paragraphs describing the four β-entries are from Briskorn et al. (2024).

Tab. 2.1: Constraints and assumptions examined in layouts of this thesis

β-entry	Term	Description
$d_j = d$	uniform due dates	All jobs of the relevant job set have the same due date $d_j = d$ with $d \in \mathbb{N}$.
fp	jobs with fixed position	A subset of jobs with given fixed positions in the final sequence.
$no - wait$	no-wait constraint	The start time of a job on a stage of operation must be equal to its completion time on the previous stage.
$p_j = p$	uniform processing time	All jobs of the relevant job set have the same processing time $p_j = p$ with $p \in \mathbb{N}$.
$pmtn$	preemption	Job preemption is allowed. Jobs do not have to be finished once started, but can be interrupted and continued arbitrarily. The amount of processing of a preempted job remains.
r_j	release date	Job j is ready to be executed at a machine at its release date $r_j \in \mathbb{N}_0$.
$ST_{sd,f}$	sequence-dependent setup times	Setup times that must be performed on the machine when performing two jobs from different job families f and f'.
w_j	weight	Job j is characterized by weight $w_j \in \mathbb{N}$ that displays the value of this job.

Machine State: δ-entry

A machine state reflects all relevant information about the condition a machine is in. In machine scheduling models, the state might include whether the machine is currently operational and how long it can operate before maintenance becomes necessary.

Obviously, in real world settings a machine state can often be accurately described only by the states of its components, each of which may be difficult to quantify. However, to the best of our knowledge, within simplifying scope of machine scheduling models, the most general perception of a machine state in the literature is reflected by a single number s. A higher number then reflects a better condition the machine is in. Notably, such a number is often not explicitly introduced in the literature but the setting at hand can be captured by considering such a number.

There might be an upper bound $\overline{s}$ or a lower bound $\underline{s}$ (or both) for a machine state, reflecting whether the machine is in perfect shape or on the verge of breaking down. If the actual state is equal to the upper bound and the lower bound, maintenance operations cannot improve the state any further, and the machine will break down if any further deterioration is caused before a maintenance is conducted, respectively.

Models in the literature differ in their instantiations of the lower bound and the upper bound. We reflect this by having $\delta = [\underline{s}, \overline{s}]$ as the first field in our classification scheme. Note that $\underline{s}$ and $\overline{s}$ might take specific numerical values or $-\infty$ and ∞, respectively, or might not be specified. For example, $\delta = [0, 5]$, $\delta = [0, \infty]$, and $\delta = [\underline{s}, \overline{s}]$ describe settings where the state is at least 0 and at most 5 in each instance, at least 0 (but not bounded from above) for each instance, and specified by the instance itself, respectively. Since a finite lower bound can be chosen arbitrarily (by adjusting the upper bound accordingly), we restrict ourselves to considering only two specific values for the lower bound in the following, namely 0 and $-\infty$.

Deterioration: ϵ-entry

Focusing on models where the machine state over time depends on the schedule, we find exclusively models in the literature where jobs trigger deterioration. Drawing on a number representing the state as sketched in the explanations for the machine state, this number would be decreased by jobs. Often the state can be assumed to deteriorate uniformly while a job is being processed. The extent to which the state is deteriorated may or may not depend on the job. We find three common manifestations in the literature. First, the deterioration might be *fixed*, i.e., processing a job deteriorates the state by a constant ω. Second, the deterioration might be *linear in the job's processing time*, i.e., processing a

job deteriorates the state by a constant ω^p per time unit of processing time, such that the deterioration caused by job j with processing time p_j is $p_j \cdot \omega^p$. Finally, the deterioration might be *individual* and not (necessarily) correlated with any other property of the job. In this case we denote it as ω_j. The last case can clearly be seen as a generalization of the former two.

We reflect these three manifestations by having $\epsilon = \omega$, $\epsilon = \omega^p$, and $\epsilon = \omega_j$, respectively, as the second field in our classification scheme.

Maintenance: ζ-entry

Since numbers represent states as outlined in the paragraph *Machine state*, these numbers are increased by maintenance operations. In the literature we find different assumptions about the duration of maintenance operations and their impact on the state.

First, we can distinguish between cases where they are independent and those where they are not. We say that they are independent if neither *the duration of a maintenance depends on the state the machine is in when it starts, nor on the state it leaves, nor vice versa*. Alternatively, if both are dependent, e.g., the machine state when starting the maintenance operation or the machine state when completing it (possibly both) determines its duration, or for a given machine state when starting the maintenance, its duration determines the state it leaves the machine in.

Second, maintenance operations might be restricted to *bring the machine into perfect shape*. Note that in this case a maintenance duration is either constant or determined by the state the machine is in when the maintenance starts. Alternatively, in a more flexible setting, the *target state* to be achieved by conducting a maintenance operation may deviate from the maximum state.

We reflect the four corresponding manifestations by having $\zeta = (dep, full)$ with $dep \in \{yes, no\}$ describing whether or not durations of maintenance operations and their impacts on the state are dependent or not and $full \in \{yes, no\}$ describing whether a maintenance necessarily brings the machine in perfect shape (yes) or not (no). Note that $full = yes$ allows to bring the machine to a perfect state, but it is not required.

Note that all four manifestations are meaningful. Independent of $full$, $dep = no$ might be found in settings where maintenance operations require a considerable setup time (overshadowing any variable time component), and $dep = yes$ is meaningful in settings where the variable time component has significant impact. On the other hand, $full = yes$ might be motivated by a substantial organizational burden of maintenance operations, such that the state shall be maximized as a matter of policy whenever a maintenance is conducted, while $full = no$ allows flexibility, e.g., when splitting a maintenance into multiple operations does involve significant additional cost.

Performance: η-entry

The state a machine is in might influence the machine's performance. In this case, it is typically assumed that *deterioration slows down the machine* and that the processing time of a job is a non-increasing function of the state at the job's start time. Alternatively, *processing time of each job does not depend on the state of the machine*, which can be seen as a special case of the former.

We represent these two settings by having $\eta = p_j$ and $\eta = p_{j,s}$ where p_j stands for processing times depending only on jobs (if any) and $p_{j,s}$ stands for processing times additionally depending on state s at a job's start time.

In this thesis, we do not work with all possible combinations of this tuple. Table 2.2 summarizes all β-entries of the $[\delta, \epsilon, \zeta, \eta]$-tuple that we require in the course of our scheduling models, especially in Chapter 4. These entries describe our case of *pd* maintenance planning and the respective type of machine deterioration.

Tab. 2.2: β-field entries concerning the machine state and maintenance planning in models with pd maintenance

entry	**Term**	**Description**
δ-entry		
$[0, \overline{s}]$	range of feasible machine state	Machine state is at least 0 and at most a given value $\overline{s} \in \mathbb{N}$.
ϵ-entry		
ω	deterioration factor	Deterioration level of each job is fixed by a constant $\omega \in \mathbb{N}$.
ζ-entry		
(n, y)	independent, full maintenance	Durations of maintenance operations do not depend on the machine state and they completely restore the machine.
η-entry		
p_j	independent processing time	Processing time $p_j \in \mathbb{N}$ of job j does not depend on the state of the machine.

Tab. 2.3: Objective functions examined in this thesis

γ-entry	**Mathematical Term**	**Description**
C_{max}	$\max\limits_{j \in J} C_j$	Maximum completion time
$\sum C_j$	$\sum_{j=1}^{J} C_j$	Total completion time
$\sum w_j C_j$	$\sum_{j=1}^{J} w_j C_j$	Total weighted completion time
L_{max}	$\max\limits_{j \in J}(C_j - d_j)$	Maximum lateness
$\sum T_j$	$\sum_{j=1}^{J} \max\{0, C_j - d_j\}$	Total tardiness
$\sum w_j T_j$	$\sum_{j=1}^{J} (w_j \cdot \max\{0, C_j - d_j\})$	Total weighted tardiness
$\sum U_j$	$U_j = \begin{cases} 1, \ if \ C_j - d_j > 0 \\ 0, \ else \end{cases}$	Number of tardy jobs
$\sum w_j U_j$		Weighted number of tardy jobs
$\sum Y_j$	$\sum_{j=1}^{J} \min\{T_j, p_j\}$	Total late work
$\sum w_j Y_j$	$\sum_{j=1}^{J} (w_j \cdot \min\{T_j, p_j\})$	Total weighted late work

Finally, we focus on the objective functions studied in the following chapters. These are traditionally completion time or due date oriented, but can also imply variability oriented factors (see Leisten and Rajendran (2015) for findings on variability in scheduling).

Framiñán et al. (2014) give a detailed overview of the cluster of objective functions in scheduling research. The completion time of job j is denoted by C_j, and its potential due date is denoted by d_j. In Table 2.3, we show the objectives that we examine in the following. Besides the typical completion time oriented objectives, namely maximum completion time (or makespan) and total completion time, we also consider a variety of due date oriented objectives, ranging from minimizing maximum lateness, where only the job with the largest lateness determines the objective value, to minimizing the sum over all tardy jobs (in contrast to lateness L_j, tardiness T_j is defined by a lower bound of zero), to minimizing the number of tardy jobs, where binary variables indicate whether a job is completed on time or not. Since we are dealing with single objective problems, only one objective is considered to be optimized per scheduling layout.

2.3 Runtime Complexity in Scheduling

In the course of this thesis, we analyze different scheduling problems with the goal of making statements regarding properties of optimal solution schedules and their runtime complexity. The following paragraph gives a brief description of the topic *runtime complexity*, since we refer to the two classes $\mathcal{P}$ and $\mathcal{NP}$ consistently.

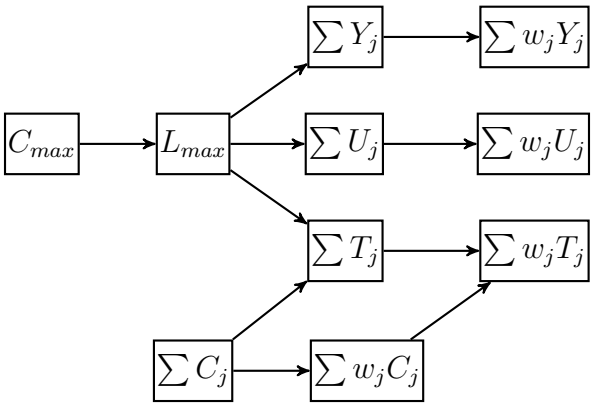
Runtime complexity, or computational complexity, refers to the complexity of the most efficient algorithm to solve a given scheduling problem to optimality. The concept of model complexity is based on the Turing Machine, introduced by Turing (1937), which describes, in a nutshell, a computer with a single processor that can solve exactly one problem (an instance of a decision problem) and has unlimited memory. A distinction must be made between a deterministic (DTM) and non-deterministic Turing Machine (NDTM), where an NDTM can be seen as a DTM with additional information about the solution of the given instance. With help of these two constructions, the two complexity classes $\mathcal{P}$ (problems with polynomial runtime) and $\mathcal{NP}$ (problems with non-deterministic polynomial runtime) can be introduced. A problem can be classified in $\mathcal{P}$ if the corresponding decision problem can be solved with the DTM in polynomial time. Problems of the class $\mathcal{NP}$ can be solved by the NDTM in polynomial time by adding the solution of the instance to the original information. It holds $\mathcal{P} \subseteq \mathcal{NP}$ if a problem can be solved by the DTM as well by the

NDTM in polynomial time without entries in the additional input. The other way around, a DTM generally solves in exponential time any problem of the class $\mathcal{NP}$ including the additional input that can be solved by the NDTM in polynomial time. Besides these classes, a decision problem $\mathcal{I}$ is identified as $\mathcal{NP}$-hard if all problems of the class $\mathcal{NP}$ can be reduced to $\mathcal{I}$ in polynomial time. If this original decision problem $\mathcal{I}$ is in $\mathcal{NP}$, it is $\mathcal{NP}$-complete[1]. (Jaehn and Pesch (2019))

For more detailed information, see Garey and Johnson (1979) which is a fundamental work for complexity theory in scheduling.

Based on the findings regarding runtime complexity in scheduling research, the objective functions presented in Table 2.3 can be classified in a hierarchy, shown in Figure 2.1. It displays possibilities to reduce an objective function to another, and thus to make statements about the runtime complexity of identical scheduling layouts, but with one of the objective functions mentioned. For example, if the problem of minimizing total completion is $\mathcal{NP}$-hard for a given layout, we know that minimizing total weighted completion time is $\mathcal{NP}$-hard, too. On the other hand, we cannot make a statement about a problem minimizing total completion time if we know that this layout minimizing total weighted completion time is $\mathcal{NP}$-hard.

Fig. 2.1: Complexity hierarchy of objective functions (Jaehn and Pesch (2019))



In the following chapters, we will focus on runtime classification schemes for various machine scheduling models, including *pd* maintenance and related constraints, applying the introduced extended Graham notation.

1 It is still open whether there are problems that are in $\mathcal{NP}$ but not in $\mathcal{P}$. The majority of researchers assume $\mathcal{P} \neq \mathcal{NP}$.

Chapter 3

Classification of Machine Availability Models in Scheduling

This chapter, which is from Briskorn et al. (2024), addresses the field of scheduling problems with machine availability constraints. In fact, we give a systematic overview of models that cover any type of state-dependent machine availability restriction caused by jobs or machines. For instance, the job position in the production sequence on a given machine can have an effect on processing times, deterioration levels of machines or the length of maintenance operations. After outlining the characteristics and demarcate this type of machine availability from the mass of adjacent research in Chapter 1 and Section 2.2, we present and categorize relevant literature dealing with any type of state-dependent machine availability and discuss relations between these categories.

3.1 Introduction

As already mentioned in Chapter 1, the majority of surveys studying machine availability in scheduling have in common that they primarily outline models dealing with time-dependent machine availability constraints and do not cover the density of non-time based restricted models as it is done in this work. Only Geurtsen et al. (2023) and Pei et al. (2023) mention effects on the machine state other than time, although they either also focus on time-dependent characteristics or the results on processing times instead of the machine.

Our focus, however, is on models in which the machines' states are deteriorated by processing jobs or are influenced by the schedule in any other way (possibly in addition to time). Note that in schedules without idle time, states depending exclusively on time can be seen as states being deteriorated continuously while processing jobs, so there is considerable overlap. Nevertheless, as long as idle time is feasible, the corresponding models are formally distinct. However, machines' states that are modified by jobs and maintenance operations, thus depend on the schedule itself, are mostly in rather recent work. The purpose of the chapter at hand is to categorize and summarize the existing work in this field.

Since we can expect machine states in settings with multiple machines to be independent of each other, we restrict ourselves to a single machine when describing machine states. However, in our survey of the literature, models with multiple machines are also covered.

The structure of this chapter is as follows. Using the introduced scheme to extend the Graham classification for machine scheduling models where the schedule impacts the machines' states (see Section 2.2), in sections 3.2 and 3.3, we summarize and categorize existing work by concentrating on properties and the runtime complexity of the models. Finally, we give a conclusion of the scope presented in this chapter in Section 3.4.

3.2 Processing Times Independent of the Machine State

In this section, we address problem settings where the processing time does not depend on the machines state ($\eta = p_j$). Within this class of problems we distinguish between settings where all jobs deteriorate a machine equally ($\epsilon = \omega$) in Section 3.2.1, where deteriorations are linear in the jobs' processing times ($\epsilon = \omega^p$) in Section 3.2.2, and where deteriorations are individual ($\epsilon = \omega_j$) in Section 3.2.3.

3.2.1 Fixed Machine Deterioration

In the paper by Briskorn et al. (2010), the authors consider an inventory constraint of a homogeneous good on classical scheduling problems. This means that each job, in addition to its classical characteristics, either delivers to or removes from inventory. The inventory

level is bounded from below but not from above. If we consider the inventory level as the state of the machine, and the jobs that deliver to the inventory as maintenance, there is an obvious connection between the two problem settings. However, when comparing the results, one has to keep in mind that, at least in general, jobs that add to the inventory contribute to the value of the objective function, while maintenance in our case does not. Nevertheless, some results are transferable, e.g., by choosing weight $w_j = 0$ or due date $d_j = \infty$ for a job j representing maintenance. These transferable results are summarized in Table 3.1, together with the following results of this section.

The first paper, to the best of our knowledge, to explicitly consider the case of $\epsilon = \omega$ is by Drozdowski et al. (2017). They consider various single machine problems, always with a maximum machine state, and fixed-duration maintenance that fully recovers the machine. In particular, they focus on preemption, which causes a faster deterioration of the machine. Thus, problems that are easy to solve when preemption is not allowed become hard. Following this line of research, Hipp (2024) and Hipp and Jaehn (2024b) add complexity results for the case of unique processing times with the number of tardy jobs and weighted sum of completion as the objective, respectively.

There is a very small stream in the literature (Calinescu et al. (2017), Jaehn (2024)) that considers scheduling problems where the position of certain jobs in the sequence is fixed in advance. These types of jobs are motivated, among other things, by maintenance operations that must be scheduled as late as possible. However, an important difference from the other problems listed here is the fact that these fixed-position jobs contribute to the respective objective function. These problems can be characterized using our notation framework, but this requires a rather complex objective function. Therefore, in Table 3.1, we will list these problems using the entry fp, which represents the fixed-position constraint of certain jobs.

Tab. 3.1: Complexity status of problems with given processing times and fixed deterioration

Problem	**Runtime**	**Notes**	**Source**
$1\vert\left([0,\infty],\omega,(n,n),p_j\right),w_j=w\vert\sum w_jC_j$	$\mathcal{O}(n\log n)$	inventory constrained scheduling	Briskorn et al. (2010), Theorem 3
$1\vert\left([0,\infty],\omega,(n,n),p_j\right),p_j=p\vert\sum w_jC_j$	$\mathcal{O}(n\log n)$	inventory constrained scheduling	Briskorn et al. (2010), Theorem 3
$1\vert\left([0,\infty],\omega,(n,n),p_j\right)\vert\sum w_jC_j$	open	inventory constrained scheduling	Briskorn et al. (2010), Table 1
$1\vert\left([0,\infty],\omega,(n,n),p_j\right),d_j=d\vert\sum U_j$	$\mathcal{O}(n^6)$	inventory constrained scheduling	Briskorn et al. (2010), Theorem 9
$1\vert\left([0,\infty],\omega,(n,n),p_j\right),p_j=p\vert\sum U_j$	$\mathcal{O}(n^3)$	inventory constrained scheduling	Briskorn et al. (2010), Theorem 8
$1\vert\left([0,\infty],\omega,(n,n),p_j\right)\vert\sum U_j$	open	inventory constrained scheduling	Briskorn et al. (2010), Table 3
$1\vert\left([0,\overline{s}],\omega,(n,y),p_j\right),pmtn\vert C_{\max}$	$\mathcal{O}(n)$		Drozdowski et al. (2017), Lemma 1
$1\vert\left([0,\overline{s}],\omega,(n,y),p_j\right),r_j\vert C_{\max}$	$\mathcal{O}(n^2)$	for small $\overline{s}$, runtime is $\mathcal{O}(n\log n)$	Drozdowski et al. (2017), Lemma 2
$1\vert\left([0,\overline{s}],\omega,(n,y),p_j\right),r_j,pmtn\vert C_{\max}$	$\mathcal{O}(n^2)$	for small $\overline{s}$, runtime is $\mathcal{O}(n\log n)$	Drozdowski et al. (2017), Lemma 3
$1\vert\left([0,\overline{s}],\omega,(n,y),p_j\right)\vert L_{\max}$	$\mathcal{O}(n\log n)$		Drozdowski et al. (2017), Lemma 4
$1\vert\left([0,\overline{s}],\omega,(n,y),p_j\right),pmtn\vert L_{\max}$	$\mathcal{O}(n\log n)$		Drozdowski et al. (2017), Lemma 5
$1\vert\left([0,\overline{s}],\omega,(n,y),p_j\right),r_j,pmtn\vert L_{\max}$	strongly NP-hard		Drozdowski et al. (2017), Lemma 6
$1\vert\left([0,\overline{s}],\omega,(n,y),p_j\right),p_j=p,r_j\vert\sum U_j$	$\mathcal{O}(n^8)$		Hipp (2024)
$1\vert\left([0,\overline{s}],\omega,(n,y),p_j\right)\vert\sum w_jC_j$	$\mathcal{O}(n\log n)$	maintenance as late as possible, Smith's rule	Hipp and Jaehn (2024b)
$1\vert\left([0,\overline{s}],\omega,(n,y),p_j\right),p_j=p,r_j\vert\sum w_jC_j$	$\mathcal{O}(n^{18})$		Hipp and Jaehn (2024b)
$1\vert fp\vert L_{\max}$	$\mathcal{O}(n^3)$	fp = fixed position	Jaehn (2024), Theorem 2
$1\vert fp\text{-}1\vert\sum U_j$	$\mathcal{O}(n^5)$	$fp-1$ = position of one job is fixed	Jaehn (2024), Theorem 5
$1\vert fp\vert\sum U_j$	open	fp = fixed position	Jaehn (2024), Table 1
$1\vert fp\vert\sum C_j$	$\mathcal{O}(n\log n)$	fp = fixed position	Jaehn (2024), Theorem 6
$1\vert fp\text{-}1\vert\sum w_jC_j$	FPTAS	$fp-1$ = position of one job is fixed	Calinescu et al. (2017), Lemma 7

3.2.2 Deterioration of Machine State Linear in Processing Time

To the best of our knowledge, only a few papers study models with constant processing times that cause machine deterioration according to ω^p. In these works, whose results are listed in Table 3.2, ω^p is usually referred to as *sum-of-processing-times-based* or *workload-dependent* processing times or deterioration. Furthermore, they have in common that maintenance operations depend on the machine state ($dep = yes$) and that maintenance fully restores the machine ($full = yes$).

Xu et al. (2015) consider a very special case of maintenance scheduling where exactly one maintenance can be assigned to the machine. This maintenance operation is fixed regarding its start time, but its length depends on the workload x before the maintenance. $f(x)$ is a non-decreasing function of the machine's workload that determines the length of the maintenance. The authors consider two assumptions for the machine's workload function, $f'(x) \geq 1$ and $0 < f'(x) < 1$, and present an exact algorithm and polynomial-time approximation scheme, respectively.

The same layout as Xu et al. (2015), but for minimizing total weighted flow time, is studied by Luo and Liu (2017). Here, an arbitrary workload function $f(x)$ is assumed. They present a $(2 + \epsilon)$-approximation and fully polynomial-time approximation scheme whose runtime depends, inter alia, on the completion time of binary search T_f (see, e.g., Cormen et al. (2022)).

A more general approach to workload-dependent maintenance scheduling is considered by Xu and Xu (2015) assuming job preemption and an arbitrary number of maintenance operations. They formulate a pseudo-polynomial algorithm that is directly affected by the minimum workload interval of the machine (the workload limits t_{min} and t_{max} are given).

Lee and Kim (2017) deal with a two machine flow shop scheduling problem where only the machine in the first stage deteriorates and requires maintenance. Since the simple two-machine flow shop minimizing total tardiness is already strongly $\mathcal{NP}$-hard (see Koulamas (1994)), the authors analyze the problem to formulate dominance properties and a branch-and-bound algorithm including a modified NEH heuristic (Nawaz et al. (1983)) to determine upper bounds for the algorithm.

In contrast to the aforementioned papers, the work of Hadidi et al. (2011) also includes deciding when to schedule maintenance operations in order to prevent machine breakdowns. The objective is to minimize the sum of costs of holding (HC), repair (MRP), and preventive maintenance (PMC), i.e.,

$$1|[[0,\bar{s}],\omega^p,(y,y),p_j]|HC+MRC+PMC.$$

Even though they do not analyze the problem regarding its complexity, and therefore we have decided not to list this work in Table 3.2, an interesting insight into the practical relevance of such models is given. In another work, Hadidi et al. (2015) modify and extend this layout by, inter alia, looking at other decision policies for their integrated approach.

Tab. 3.2: Complexity status of problems with given processing times and linear deterioration

Problem	**Runtime**	**Notes**	**Source**
$1\|f'(x)\geq 1,[[0,\bar{s}],\omega^p,(y,y),p_j]\|\sum C_j$	$\mathcal{O}(n^2)$	$f'(x)$: function of machine workload	Xu et al. (2015)
$1\|0<f'(x)<1,[[0,\bar{s}],\omega^p,(y,y),p_j]\|\sum C_j$	PTAS, $\mathcal{O}(n^{2k+1})$	$f'(x)$: function of machine workload, k: jobs of (k,k)-exchange	Xu et al. (2015)
$1\|pmtn,[[0,\bar{s}],\omega^p,(y,y),p_j]\|C_{max}$	$\mathcal{O}(\lceil\sum p_j/t_{min}\rceil)$	t_{min} = machine's minimum working interval	Xu and Xu (2015)
$F2\|m_1=[[0,\bar{s}],\omega^p,(y,y),p_j],m_2=\emptyset\|\sum T_j$		Branch-and-Bound	Lee and Kim (2017)
$1\|[[0,\bar{s}],\omega^p,(y,y),p_j]\|\sum w_jC_j$	$(2+\epsilon)$-approx., $\mathcal{O}(n\log f(S)/\epsilon^2+T_f\log f(S)\log S/\epsilon)$		Luo and Liu (2017), Theorem 1
$1\|[[0,\bar{s}],\omega^p,(y,y),p_j]\|\sum w_jC_j$	FPTAS, $\mathcal{O}(n^4\log f(S)/\epsilon^3+T_f\log f(S)\log S/\epsilon)$		Luo and Liu (2017), Theorem 2

3.2.3 Job Individual Machine Deterioration

The paper by Bock et al. (2012) discusses the complexity of several scheduling problems involving maintenance and individual deterioration. Thus, each job deteriorates the machine by a job-dependent amount ω_j, and maintenance operations need to be conducted to

ensure the machine's processing. Thereby maintenance operations improve the machine state either to perfect shape or partially. The processing times of jobs do not depend on the machine state. The authors consider different objective functions and distinguish between the full and target maintenance cases. The complexity results can be found in Table 3.3. Grigoriu and Briskorn (2017) provide a 5/4-approximation algorithm for the single machine problem with full maintenance and makespan minimization introduced by Bock et al. (2012). Tian et al. (2019) extend the problem from a single machine to parallel machines with partial maintenance operations. They propose a $(2\rho_{\max}+3)$-approximation algorithm for $\rho_{\max} = \max(\omega_j/p_j)$ to minimize makespan.

As already explained in Section 3.2.1, scheduling with inventory constraints bears strong parallels to the concepts of scheduling with maintenance. Briskorn et al. (2010) give a general definition of a scheduling problem with inventory constraints and due dates. Thereby the processing times of jobs do not depend on the machines' states and arbitrary inventory modifications are allowed, indicating arbitrary deterioration and partial maintenance operations in terms of maintenance scheduling. They consider the objective functions $\sum w_j C_j$, $L_{\max}$, and $\sum U_j$ and show that the general versions of the considered problems are all (strongly) $\mathcal{NP}$-hard. Hence, they consider problem variants assuming that different jobs have certain parameter values in common. The results for the general problem and some polynomially solvable problem variants maintaining $\epsilon = \omega_j$ can be found in Table 3.3. In Briskorn et al. (2013), the authors present exact algorithms to solve the general problem for the objective $\sum w_j C_j$. The general problem of minimizing maximum lateness $L_{\max}$ can be solved by a branch-and-bound algorithm as presented in Briskorn and Leung (2013). In Morsy and Pesch (2015), the authors present a 2-approximation algorithm for a problem variant of the general problem with objective $\sum w_j C_j$.

Different from the problem definition in Briskorn et al. (2010), in Davari et al. (2020), an inventory scheduling problem is formulated to minimize the makespan $C_{\max}$ while the inventory level may not exceed a maximum level and all jobs have release dates. Interpreted as a maintenance scheduling problem, a set of mandatory maintenance operations, including their properties, is given and needs to be scheduled simultaneously with the jobs, where the maximum maintenance capacity is limited. The problem remains $\mathcal{NP}$-hard.

Tab. 3.3: Complexity status of problems with given processing times and individual deterioration

Problem	**Run time**	**Notes**	**Source**
$1\|[[0,\infty],\omega_j,(n,n),p_j]\|\sum C_j$	strongly $\mathcal{NP}$-hard	inventory constrained scheduling; $J = J^+ \cup J^-$; $I^*(J^{-/+}) \in \{0,1\}$ equals 1, if the parameter (*) is constant for all jobs with negative/positive inventory modification; d_j: due dates	Briskorn et al. (2010), Theorem 1
$1\|[[0,\infty],\omega_j,(n,n),p_j]\|\sum w_jC_j$	strongly $\mathcal{NP}$-hard		Briskorn et al. (2010), Theorem 2
$1\|I^w(J^-) = 1, I^p(J^+) + I^w(J^+) + I^\omega(J^+) \geq 2, [[0,\infty],\omega_j,(n,n),p_j = p]\|\sum w_jC_j$	$\mathcal{O}(n \log n)$		Briskorn et al. (2010), Theorem 3
$1\|I^w(J^-) = I^\omega(J^+) = 1, I^p(J^+) = I^w(J^+) = 0, [[0,\infty],\omega_j,(n,n),p_j = p]\|\sum w_jC_j$	open		Briskorn et al. (2010)
$1\|[[0,\infty],\omega_j,(n,n),p_j]\|L_{\max}$	strongly $\mathcal{NP}$-hard		Briskorn et al. (2010), Theorem 4
$1\|I^p(J^+) + I^\omega(J^+) \geq 1, [[0,\infty],\omega_j,(n,n),p_j]\|L_{\max}$	$\mathcal{O}(n^2 \log(\sum p_j))$		Briskorn et al. (2010), Theorem 5
$1\|[[0,\infty],\omega_j,(n,n),p_j]\|\sum U_j$	$\mathcal{NP}$-hard		Briskorn et al. (2010), Theorem 6
$1\|I^d(J^+) = 1, I^p(J^+) + I^\omega(J^+) \geq 1, [[0,\infty],\omega_j,(n,n),p_j = p]\|\sum U_j$	$\mathcal{O}(n^3)$		Briskorn et al. (2010), Theorem 8
$1\|I^d(J^-) = I^p(J^+) = 1; [[0,\infty],\omega_j,(n,n),p_j = p]\|\sum U_j$	$\mathcal{O}(n^6)$		Briskorn et al. (2010), Theorem 9
$1\|r_j,[[0,\bar{s}],\omega_j,(n,n),p_j]\|C_{\max}$	strongly $\mathcal{NP}$-hard	inventory constrained scheduling; release dates r_j	Davari et al. (2020), Theorem 1
$1\|[[0,\bar{s}],\omega_j,(y,y),p_j]\|\sum C_j$	$\mathcal{NP}$-hard	Even for $p_j = p$ and $p_j = \omega_j$	Bock et al. (2012), Theorem 3
$1\|[[0,\bar{s}],\omega_j,(y,n),p_j]\|\sum C_j$	$\mathcal{O}(n \log n)$	If $p_j < p_i$ implies $p_j + \omega_j \leq p_i + \omega_i$	Bock et al. (2012), Theorem 10
$1\|[[0,\bar{s}],\omega_j,(y,y),p_j]\|\{C_{\max}, L_{\max}\}$	o. $\mathcal{NP}$-hard, $\mathcal{O}(n^2(\bar{s})^2 \cdot \log(\sum(p_j + \omega_j)))$	Even for $p_j = p$ and $p_j = \omega_j$	Bock et al. (2012), Theorem 5,6,7
$1\|[[0,\bar{s}],\omega_j,(y,n),p_j]\|\{C_{\max}, L_{\max}\}$	$\mathcal{O}(n \log n)$		Bock et al. (2012), Theorem 11
$1\|[[0,\bar{s}],\omega_j,(y,y),p_j]\|\sum U_j$	$\mathcal{NP}$-hard	Even for $p_j = p$ and $p_j = \omega_j$	Bock et al. (2012), Theorem 8
$1\|[[0,\bar{s}],\omega_j,(y,n),p_j]\|\sum U_j$	$\mathcal{O}(n \log n)$	If $p_j < p_i$ implies $p_j + \omega_j \leq p_i + \omega_i$	Bock et al. (2012), Theorem 12

3.3 Processing Times Depending on the Machine State

In this section, we address problem settings where the processing time depends on the machines' states ($\eta = p_{js}$). Within this class of problems we distinguish between settings where all jobs deteriorate a machine equally ($\epsilon = \omega$) in Section 3.3.1, and where deteriorations are linear in the jobs' processing times ($\epsilon = \omega^p$) in Section 3.3.2. In Section 3.3.1, we further differentiate the models with respect to the jobs' processing time characteristics and whether the deterioration is job-dependent or not. We do not explicitly consider the case where deterioration is individual ($\epsilon = \omega_j$), but the operating principals of the models in sections 3.3.1 and 3.3.1 are closely related to the idea of individual deterioration.

3.3.1 Fixed Machine Deterioration

The scheduling of jobs whose (deterministic) processing time is not a constant has been considered in various contexts in the literature. A classical scheduling problem considers related parallel machines, i.e., machines that differ only in speed (Chekuri and Khanna (2001); Jansen et al. (2020)). Similarly, some models consider that the speed of a single machine can be adjusted, e.g., in speed scaling (Albers et al. (2014); Bampis et al. (2015)). In this context, there is a large body of literature dealing with a rather simple maintenance scheduling model, which has been proposed by Lee and Leon (2001), among others. Thereby, a single maintenance operation can potentially be scheduled within the entire planning horizon. It is assumed that the maintenance operation decreases the processing times of all subsequent jobs, but obstructs the machine, causing a delay in job processing equal to the duration of the maintenance. This trade-off needs to be evaluated and decisions need to be made regarding both, job sequencing and maintenance scheduling. These effects can be classified in the context of the machine's maintenance level as follows. The machine's maintenance level is binary and only becomes one after the maintenance has been performed. The processing of jobs has no impact on the level, i.e., we have a constant deterioration of $\omega_j = 0$ per job j. Mosheiov and Sidney (2010) extend the problem to maintenance operations whose duration increase with their start time. This assumption is consistent with the understanding of a deteriorating machine over time. All the problems studied are shown to be solvable in polynomial time. Other authors deal with this type of

maintenance scheduling problems and include further properties such as random machine breakdowns (Lee and Lin (2001)) resulting in three possible maintenance levels – initial, after maintenance operations and breakdowns, position-dependent maintenance duration, and due-window assignments (Mor and Mosheiov (2012); Mor and Mosheiov (2015)). The problem is extended to a parallel machine environment in Wang et al. (2011) but remains polynomially solvable for a fixed number of machines m. Yang et al. (2014) state the problem of potential single maintenance operations for parallel machine setting. The processing times still depend on the fact whether a maintenance operation was previously scheduled on the machine and are further controllable by job compression. The problem can be solved in $\mathcal{O}(n^{m+3})$.

Motivated by machine deterioration, and thus approaching the scope of this paper, the processing time of a job may depend on the starting time of the job (Cheng et al. (2004)). In this section, we will consider situations in which the processing time of a job depends on its position in the schedule. In fact, there are only a few papers on this topic. A significant part of the literature concentrates on learning effects, which propose that the processing time of jobs decreases the later they are scheduled (Biskup (1999)). Conversely, another perspective in the literature explores deterioration effects, which propose that job processing time increases as the job position advances due to machine wear. Although learning effects are not in our scope, problems from both settings can be sorted into the model presented in this paper. Thereby, the state of the machine counts the number of jobs processed so far and thus constantly deteriorates per job. Furthermore, the literature is divided according to whether maintenance operations are involved or not. Thereby, the impact of maintenance operations on the machine state depends on the specific problem at hand.

In most papers it is assumed that each job has a base processing time p_j which may be modified by its position and therefore depends on the state of the machine. The literature can be categorized more specifically based on whether the position's effect on a job's processing time is depending on the job. The remainder of this section will follow the proposed division.

Position-Dependent, Job-Independent Deterioration of Machine State

First, we consider problem formulations with position-dependent processing times. Thus, the influence of the position on the processing time is job-independent. The first (to the best of our knowledge) to consider processing times that depend on the number of jobs processed is Gawiejnowicz (1996). He analyzes decreasing machine speed (deterioration), increasing machine speed (learning effect), and arbitrary speed functions. We restrict ourselves to the first. While these deterioration functions are very general (requiring only that processing times increase), Mosheiov (2005) considers a polynomial increase of a job's processing time with increasing positions in the schedule. If p_j is the processing time of job j if it were scheduled first, then its processing time at position r in the schedule is given by $p_j r^a$ for some $a > 0$. Maintenance operations are not taken into account, so according to our notation, the state of the machine never improves during the processing time. He makes use of knowledge from problems with learning functions (Mosheiov (2001)).

The additional consideration of precedence constraints is introduced by Gordon et al. (2008). They analyze not only the polynomial processing times $p_j r^a$ described above (with positive results only for $a = 1$), but also exponential processing times in the form of $p_j \gamma^{r-1}$, where $\gamma > 1$ is a parameter.

Kuo and Yang (2008) do not require, but allow, maintenance operations to speed up processing times. They also introduce a constraint that limits the number of maintenance operations. Since the number of maintenance operations is naturally bounded by n (if no preemption is allowed), any algorithm solving a problem with such a constraint can be used up to n times to solve the corresponding unrestricted problem. Their result (see Table 3.4) is generalized by Rustogi and Strusevich (2012a). The latter distinguishes between group-dependent and group-independent deterioration, where a group refers to the jobs scheduled between two maintenance operations. However, group-independent deterioration simply refers to the case where a machine is fully recovered after maintenance ($full = yes$). Group-dependent deterioration can then be modeled by a partial recovery of the machine.

Xue et al. (2014) consider general position-dependent processing times that may include learning effects and deterioration of a job within an instance. However, they are also restricted to problems that include only learning effects (not considered here) and de-

terioration, respectively. Unlike previous approaches, they minimize a linear combination of makespan, sum of earliness, and sum of tardiness.

The works of Montoya-Torres et al. (2021) and Montoya-Torres et al. (2023) are not listed in Table 3.4 because they do not provide complexity results. However, they present mixed integer programming models (MIPs) and heuristics for multi-machine problems with deterioration. Both papers justify their model by workers who represent the machines and whose productivity decreases due to fatigue or stress. Montoya-Torres et al. (2021) first considers the problem

$$Pm|\left([-\infty,\infty],\omega,-,p_{js}=p_j r^a\right)|C_{\max}$$

with $a > 0$ and proposes a MIP. Further MIPs are presented for extensions, namely different but related machine (or worker) deterioration factors $a_i > 0$ for each machine i,

$$Pm|\left([-\infty,\infty],\omega,-,p_{js}=p_j r^{a_i}\right)|C_{\max},$$

and breaks for workers, which corresponds to machine maintenance,

$$Pm|\left([0,\overline{s}],\omega,[n,n],p_{js}=p_j r^{a_i}\right)|C_{\max}.$$

Montoya-Torres et al. (2023) analyze the problem

$$Pm|\left([-\infty,\infty],\omega,-,p_{js}=\lceil p_j r^a\rceil\right)|T_{\max},$$

which is at least binary $\mathcal{NP}$-hard, since for infinitesimal a it can be reduced to $P2||C_{\max}$, which is $\mathcal{NP}$-hard.

Tab. 3.4: Complexity status of problems with state dependent processing times and fixed deterioration

Problem	Runtime	Notes	Source
$1\vert([0,\overline{s}],\omega,(n,y),p_{js})\vert C_{\max}$	$\mathcal{O}(n\log n)$	for $\overline{s}\geq n$	Gawiejnowicz (1996), Theorem 2
$1\vert([0,\overline{s}],\omega,(n,y),p_{js})\vert C_{\max}$	$\mathcal{O}(n\log n)$	for $\overline{s}<n$	Gawiejnowicz (1996), Theorem 4
$1\vert([-\infty,\infty],\omega,-,p_{js}=p_jr^a)\vert C_{\max}$	$\mathcal{O}(n\log n)$	r: pos. in schedule, $a>0$, LPT-rule	Mosheiov (2005), p. 885
$1\vert([-\infty,\infty],\omega,-,p_{js}=p_jr^a)\vert\sum C_j$	$\mathcal{O}(n^3)$	r: pos. in schedule, $a>0$, solution is V-shaped	Mosheiov (2005), p. 884, Theorem 1
$1\vert([-\infty,\infty],\omega,-,p_{js}=p_jr),SP-prec\vert C_{\max}$	$\mathcal{O}(n\log n)$	r: pos. in schedule SP-prec: series parallel precedence constraints	Gordon et al. (2008), Theorem 3
$1\vert\left([-\infty,\infty],\omega,-,p_{js}=p_j\gamma^{r-1}\right),SP-prec\vert C_{\max}$	$\mathcal{O}(n\log n)$	r: pos. in schedule, $\gamma>1$, SP-prec: series parallel precedence constraints	Gordon et al. (2008), Theorem 6
$1\vert\left([-\infty,\infty],\omega,-,p_{js}=p_j\gamma^{r-1}\right),prec\vert\sum C_j$	$\mathcal{O}(n\log n)$	r: pos. in schedule, $\gamma\geq 2$	Gordon et al. (2008), Theorem 7
$1\vert([-\infty,\infty],\omega,(n,y),p_{js}=p_jr^a)\vert C_{\max}$	$\mathcal{O}(n^2)^*$	r: pos. in schedule, $a>0$ see next line for generalization	Kuo and Yang (2008), Theorem 3
$1\vert([-\infty,\infty],\omega,(n,y),p_{js})\vert C_{\max}$	$\mathcal{O}(n^2)$		Rustogi and Strusevich (2012a), p. 795
$1\vert([-\infty,\infty],\omega,(n,n),p_{js})\vert C_{\max}$	$\mathcal{O}(n^2)$		Rustogi and Strusevich (2012a), Theorem 3
$1\vert([0,\overline{s}],\omega,(n,y),p_{js}=p_jr^a)\vert\alpha C_{\max}+\beta\sum E_j+\gamma\sum T_j$	$\mathcal{O}(n^2\log n)$	r: pos. in schedule, $a>0$,	Xue et al. (2014), Theorem 3

*The underlying paper states a runtime of $\mathcal{O}(n\log n)$, but Rustogi and Strusevich (2012a) correct to $\mathcal{O}(n^2)$.

Position-Dependent, Job-Dependent Deterioration of Machine State

Next, we focus on approaches where the actual processing times of jobs are influenced not only by the position, but also by job-dependent factors.

First, we consider problems with no maintenance operations involved and collect some complexity results in Table 3.5.

Again, there is a significant body of research that focuses primarily on learning effects. However, the work of Mosheiov and Sidney (2003) and Mosheiov and Sidney (2005), for example, provides approaches and results that can be used for research that takes into account deterioration effects. Mosheiov and Sarig (2008) address a scheduling problem

with completely arbitrary position-dependent processing times that are given numerically for each job and each position, so there are no limitations to learning or deterioration effects, combined with a due window assignment problem. They propose an $\mathcal{O}(n^3)$ algorithm. Yu et al. (2014) provide an overview of the complexity of scheduling problem involving parallel machines that are either identical or unrelated and position-dependent processing times $p_{ijr} = f_{ij}(i, j, r)$ for different properties of f_{ij}. The primary objective in their considered scenarios is to reduce the total load. The results are listed in Table 3.5.

In Zhao and Tang (2010), a very general interpretation of position-dependent and job-dependent processing times is given by $p_{js} = p_j r^{\alpha_j}$ for a job j scheduled at position r and a job-dependent deteriorating factor $\alpha_j > 0$. The authors consider both problems, with and without maintenance scheduling, and prove them to be polynomially solvable. Some complexity results for variants of the no-maintenance problem are gathered in Table 3.5. Various problem formulations dealing with the maintenance case and position-dependent and job-dependent processing times are investigated differing in properties and restrictions of the maintenance operations. There may be restrictions on the number of maintenance operations that may be conducted or the duration of a maintenance operation may depend on its start time or on how many maintenance operations have already been conducted within the schedule. The latter two approaches reduce or increase the duration, again incorporating deterioration or learning effects. As an example, Yang and Yang (2010a) deal with problem variants of the problem introduced by Zhao and Tang (2010) that differ in the maintenance duration. More results and problem variants can be found in Table 3.6.

Tab. 3.5: Complexity status of problems with state dependent processing times and job-dependent deterioration factors without maintenance

Problem	**Run time**	**Notes**	**Source**
$1\|[[0,\bar{s}],\omega,-,p_{js}=p_j r^{\alpha_j}]\|C_{\max}$	$\mathcal{O}(n^3)$	r: pos. in schedule, $\alpha_j > 0$	Zhao and Tang (2010), Lemma 1
$Rm\|[[0,\bar{s}],\omega,-,p_{ijs}=f_{ij}(i,j,r)]\|TL$	$\mathcal{O}(mn^3)$	r: pos. in schedule, f_{ij} non-decreasing in r, TL: total load	Yu et al. (2014), Theorem 1
$Pm\|[[0,\bar{s}],\omega,-,p_{js}=f_j(j,r)]\|TL$	$\mathcal{O}(n^3)$	r: pos. in schedule, f_j non-decreasing in r, TL: total load	Yu et al. (2014), Theorem 2
$Rm\|[[0,\bar{s}],\omega,-,p_{ijs}=f_{ij}(i,j,r)]\|TL$	$\mathcal{O}(n^{m+2})$	r: pos. in schedule, $f_{i,j}$ non-monotonic, TL: total load	Yu et al. (2014), Theorem 4
$1\|S_{psd},[[1,\bar{s},\omega,-,p_{js}=p_j r^{\alpha_j}]\|\sum(\alpha E_j+\beta T_j+\eta d_1+\delta D)$	$\mathcal{O}(n^3)$	r: pos. in schedule, S_{psd} = past-sequence dependent setup times, $\alpha,\beta,\eta,\delta > 0$, E_j: earliness, T_j: tardiness, $D = d_2 - d_1$: due window	Yang and Yang (2012), Theorem 1
$Rm\|$resource $,[[-1,\bar{s}],\omega,-,p_{ijs}=p_{ij}r^{\alpha_{ij}}-b_{ij}u_{ij}]\|TC$	$\mathcal{O}(n^{m+2})$	r: pos. in schedule, TC = total cost; u_{ij}: resource allocation	Hsu and Yang (2014), Theorem 1,2
$Rm\|$resource$,[[-1,\bar{s}],\omega,-,p_{ijs}=(p_{ij}r^{\alpha_{ij}}/u_{ij})^k]\|TC$	$\mathcal{O}(n^{m+2})$	r: pos. in schedule, TC = total cost; u_{ij}: resource allocation	Hsu and Yang (2014), Theorem 3,4

Tab. 3.6: Complexity status of problems with state dependent processing times and job-dependent deterioration factors with maintenance (MA)

Problem	**Run time**	**Notes**	**Source**
$1\|\text{single MA}, m = b + ct, [[0, \bar{s}], \omega, (n, y), p_{js} = p_j r^{a_j}] \| \sum(\alpha E_j + \beta T_j + \gamma q)$	$\mathcal{O}(n^4)$	m: duration of the MA, $b > 0, c \geq 0$, r: pos. in schedule, $\alpha, \beta, \gamma \geq 0$, E_j: earliness, T_j: tardiness, q: common flow allowance	Ji et al. (2014), Theorem 1
$1\|m_i = t_0, [[0, \bar{s}], \omega, (n, y), p_{js} = p_j r^{\alpha_j}] \| C_{\max}$	$\mathcal{O}(n^4)$	m_i: duration of the i-th MA, t_0: basic MA duration, r: pos. in schedule, $\alpha_j > 0$	Zhao and Tang (2010), Theorem 1
$1\|m_i = t_0 i^b, [[0, \bar{s}], \omega, (n, y), p_{js} = p_j r^{\alpha_j}] \| C_{\max}$	$\mathcal{O}(n^4)$	m_i: duration of the i-th MA, t_0: basic MA duration, r: pos. in schedule, $\alpha_j > 0$, $b < 0$	Yang and Yang (2010a), Theorem 1
$1\|m_i = t_0 + \beta t_i, [[0, \bar{s}], \omega, (n, y), p_{js} = p_j r^{\alpha_j}] \| C_{\max}$	$\mathcal{O}(n^5)$	m_i: duration of the i-th MA, t_0: basic MA duration, t_i runtime before the $i-$th MA, r: pos. in schedule, $\alpha_j > 0$, $\beta \geq 0$	Yang and Yang (2010a), Theorem 3
$Rm\|MA \leq k, m_{i,l} = \alpha_l + \beta_l t_{il}, [[0, \bar{s}], \omega, (n, y), p_{jlr} = p_{jl} f_{jl}(r)] \| \sum C_j$	$\mathcal{O}(n^{m+k+2})$	k: maximum number of MAs, m_i: duration of the i-th MA on machine l, t_{il}: run time before the $i-$th MA on machine l, r: pos. in schedule, $\alpha_l > 0$, $\beta_l \geq 0$, $f_{jl}(r) \geq 1$	Yang (2013), Theorem 1
$1\|MA \leq k_0, m_i = a + bt_i, [[0, \bar{s}], \omega, (n, y), p_{js} = p_j r^{\alpha_j}] \| \sum C_j$	$\mathcal{O}(n^{k_0+3})$	m_i: duration of the i-th MA, t_i runtime before the $i-$th MA, r: pos. in schedule, k_0: maximum number of MAs, $\alpha_j, a>0$, $b \geq 0$	Yang and Yang (2010b), Theorem 1
$1\|m_i = t_i [[0, \bar{s}], \omega, (n, y), p_{js} = p_j g_j(r)] \| C_{\max}$	$\mathcal{O}(n^4)$	m_i: duration of the i-th MA, r: pos. in schedule	Rustogi and Strusevich (2012b), Theorem 6
$1\|PRM = k, m_i = at_i + b, [[1, \bar{s}], \omega, (n, y), p_{js} = f_j(p_j, r)] \| \alpha C_{\max} + \beta \sum E_j + \gamma \sum T_j$	$\mathcal{O}(n^4)$	m_i: duration of the i-th MA, t_i runtime before the $i-$th MA, r: pos. in schedule, PRM: maximum number of jobs between two MAs, $\alpha, \beta, \gamma > 0$, $a \geq 0$, E_j: earliness, T_j: tardiness	Xue et al. (2014), Theorem 1

3.3.2 Deterioration of Machine State Linear in Processing Time

When dealing with problems assuming machine state-dependent processing times, we can find only a few works that also deal with maintenance scheduling (see Table 3.7). In Zhang et al. (2017), processing time p_j of job j is time-dependent (deterioration rate β_j multiplied by time t) and can be interpreted as state-dependent only if there are no idle times on the machine. This type of deteriorating processing time is not part of our scope. Nevertheless, this study is interesting when looking at state-dependent deterioration, because maintenance scheduling is workload-dependent with workload $x = \sum p_j$ of jobs scheduled before the maintenance operation and convex non-decreasing function $f(x)$. They prove the $\mathcal{NP}$-hardness of this single machine layout for minimizing makespan and total completion time.

Another work dealing with a kind of maintenance scheduling and state-dependent processing times is conducted by Salama and Srinivas (2021). They assume a tool change that has to be performed in order to execute jobs. These tools are deteriorated by the scheduled jobs' processing times. Furthermore, the actual processing time, which is limited by a lower bound, is affected by a deterioration rate of the machine α_j^D (slowing down jobs) and the actual state of the tools α_j^U (speeding up jobs). The objective is a combination of energy, tool change, and tardiness costs. After formulating a mixed integer linear programming model (MILP), in a second step two simulated-annealing based heuristics are generated (an adaptive neighborhood search is added to the second heuristic) and compared with each other.

The paper by Gordon et al. (2008) is already covered in Section 3.3.1 as one that does not deal with maintenance scheduling. In addition to the layouts mentioned above, the authors also study problems with processing time dependent deterioration. The actual processing time depends entirely on the sum of processing times of already processed jobs – in the paper called *cumulative deterioration*. Minimizing makespan in this layout is examined with and without precedence constraints. The same layout is tackled by Dolgui et al. (2012). In their work, they analyze the characteristics and add, e.g., learning effects to the problem layout. In the course of their research, they formulate polynomial algorithms, but do not give specific runtime analyses for their approaches.

Liu et al. (2013) present another work assuming workload-dependent processing times without any type of maintenance scheduling. A two-agent approach is studied where the agents (machines) compete to complete the jobs. Two different types of processing time based equations are considered to determine the actual processing time, a purely cumulative approach and an approach that includes the total processing times.

An interesting and noteworthy problem of Seif et al. (2018) deals with a permutation flow shop problem assuming a fuzzy bi-objective function of minimizing total tardiness and maintenance costs. The processing times are based on the actual machine state and lead to maintenance scheduling, so we obtain the scheduling problem

$$Fm|prmu, [[0, \bar{s}], \omega^p, (y, y), p_{js} = \lambda^h \cdot p_{i[j]}]| \sum T_j,\ y_{i[j]k}(SP_{ik} + e_{ik}WF_k).$$

The authors formulate a fuzzy multi-objective linear programming based on the work of Liang (2006) and solve it with a standard genetic algorithm. It is not listed in Table 3.7 because no statements are made about the runtime complexity to solve the problem.

Tab. 3.7: Complexity status of problems with state dependent processing times and linear deterioration

Problem	**Runtime**	**Notes**	**Source**
$1\|[[-\infty,\infty],\omega^p,-,p_{js}=(1+\sum_{k=1}^{r-1}p_k)^A]\|C_{max}$	$\mathcal{O}(n\log(n))$	r: pos. in schedule	Gordon et al. (2008), p. 362
$1\|prec,[[-\infty,\infty],\omega^p,-,p_{js}=(1+\sum_{k=1}^{r-1}p_k)^1]\|C_{max}$	$\mathcal{O}(n)$	r: pos. in schedule	Gordon et al. (2008), Theorem 4
$1\|SP,[[-\infty,\infty],\omega^p,-,p_{js}=(1+\sum_{k=1}^{r-1}p_k)^2]\|C_{max}$	$\mathcal{O}(n\log(n))$	r: pos. in schedule, SP: series-parallel prec.	Gordon et al. (2008), Theorem 5
$1\|prec,[[-\infty,\infty],\omega^p,-,p_{js}=p_j(1+\sum_{k=1}^{r-1}p_{\pi(k)})^A]\|C_{max}$		polynomial solvable	Dolgui et al. (2012)
$1\|[[-\infty,\infty],\omega^p,-,p_{js}=(1+\sum_{l=1}^{r-1}p_{[l]}^G)p_j^G]\|\sum C_j^A : C_{max}^B \leq U$	$\mathcal{O}(n_A\log n_A + n_B)$	2-agent model (G=A,B)	Liu et al. (2013), Theorem 1
$1\|[[-\infty,\infty],\omega^p,-,p_{js}=(1+\frac{\sum_{l=1}^{r-1}p_{[l]}^G}{\sum_{l=1}^{n}p_l^G})p_j^G]\|\sum C_j^A : C_{max}^B \leq U$	$\mathcal{O}(n_A\log n_A + n_B)$	2-agent model (G=A,B)	Liu et al. (2013), Theorem 1
$1\|[[0,\bar{s}],\omega^p,(y,y),p_{js}=p_j+\beta_j t]\|C_{max}$	strongly $\mathcal{NP}$-hard	no. of MA arbitrary, 3-partition	Zhang et al. (2017), Theorem 3.2
$1\|[[0,\bar{s}],\omega^p,(y,y),p_{js}=p_j+\beta_j t]\|C_{max}$	$\mathcal{NP}$-hard in the ordinary sense	no. of MA fixed, subset product problem, see Ji et al. (2006)	Zhang et al. (2017), Theorem 3.3
$1\|[[0,\bar{s}],\omega^p,(y,y),p_{js}=p_j+\beta_j t]\|\sum C_j$	strongly $\mathcal{NP}$-hard	no. of MA arbitrary, 3-partition	Zhang et al. (2017), Theorem 5.1
$1\|[[0,\bar{s}],\omega^p,(y,y),p_{js}=p_j+\beta_j t]\|\sum C_j$	$\mathcal{NP}$-hard in the ordinary sense	no. of MA fixed, subset product problem, see Ji et al. (2006)	Zhang et al. (2017), Theorem 5.2
$1\|[[0,\bar{s}],\omega^p,(y,y),p_{js}=p_j+max\{\alpha_j^D a_j-\alpha_j^U(L-l_j),0\}]\|\mu^E\sum p_j+\mu^R\sum x_i+\mu^T\sum T_j$	$\mathcal{NP}$-hard	heuristic: simulated annealing	Salama and Srinivas (2021)

Besides those works, two papers, listed in Table 3.8, include batch or group scheduling characteristics in their scheduling models. Cheng et al. (2011) combine classical batch scheduling including setups between batches. Sum-of-processing-time-based deteriorating jobs, but no maintenance operations, are considered. Cheng et al. (2018) add maintenance scheduling to this type of problem, where the actual processing time is affected only by the total processing times of the jobs scheduled in the same batch.

Tab. 3.8: Complexity status of problems with batching, state dependent processing times and linear deterioration

Problem	**Runtime**	**Notes**	**Source**
$1\|S_{psd}[[-\infty,\infty],\omega^p,-,p_{js} = p_j(1+\sum_{l=1}^{r-1}\log p_{[l]})^a\|C_{max}$	$\mathcal{O}(n\log(n))$	r: pos. in schedule, S_{psd}= past-sequence-dependent setup times	Cheng et al. (2011), Theorem 1
$1\|S_{psd},[[-\infty,\infty],\omega^p,-,p_{js} = p_j(1+\sum_{l=1}^{r-1}\log p_{[l]})^a\|\sum C_j$	$\mathcal{O}(n\log(n))$	r: pos. in schedule, S_{psd}= past-sequence-dependent setup times	Cheng et al. (2011), Theorem 2
$1\|S_{psd},[[-\infty,\infty],\omega^p,-,p_{js} = p_j(1+\sum_{l=1}^{r-1}\log p_{[l]})^a\|\sum C_j^2$	$\mathcal{O}(n\log(n))$	r: pos. in schedule, S_{psd}= past-sequence-dependent setup times	Cheng et al. (2011), Theorem 3
$1\|S_{psd},agreeable,[[-\infty,\infty],\omega^p,-,p_{js} = p_j(1+\sum_{l=1}^{r-1}\log p_{[l]})^a\|\sum T_j$	$\mathcal{O}(n\log(n))$	r: pos. in schedule, S_{psd}= past-sequence-dependent setup times	Cheng et al. (2011), Theorem 4
$1\|S_{psd},agreeable,[[-\infty,\infty],\omega^p,-,p_{js} = p_j(1+\sum_{l=1}^{r-1}\log p_{[l]})^a\|L_{max}$	$\mathcal{O}(n\log(n))$	r: pos. in schedule, S_{psd}= past-sequence-dependent setup times	Cheng et al. (2011), Theorem 5
$1\|B,[[0,\bar{s}],\omega^p,(y,y),p_{js} = p_{ij}(1+\sum_{k=1}^{r-1}p_{i[k]})^{\alpha_i}]\|C_{max}$	$\mathcal{O}(n\log(n))$	r: pos. in schedule, B_i: batch	Cheng et al. (2018)
$1\|B,[[0,\bar{s}],\omega^p,(y,y),p_{js} = p_{ij}(1+\sum_{k=1}^{r-1}p_{i[k]})^{\alpha_i\geq 1}]\|\sum C_j$	$\mathcal{O}(n\log(n))$	r: pos. in schedule, B_i: batch	Cheng et al. (2018)
$1\|B,[[0,\bar{s}],\omega^p,(y,y),p_{js} = p_{ij}(1+\sum_{k=1}^{r-1}p_{i[k]})^{0<\alpha_i<1}]\|\sum C_j$	open; $\mathcal{O}(n^2\log(n))$, $\mathcal{O}(2^n)$	r: pos. in schedule, B_i: batch, SPT & V-shaped-based heuristics	Cheng et al. (2018)

3.4 Conclusion

In this chapter, we address traditional machine scheduling problems that are characterized by machine and/or job features leading to machine availability constraints on the execution of jobs. In contrast to the majority of works dealing with machine availability constraints, we concentrate on non-time related factors as the main cause of machine unavailability. We summarize these models including any type of non-time related machine availability constraint under the term *state-dependent machine availability*. In the course of this work, we categorize various sources of machine deterioration regarding the characteristics of jobs' processing times and their effect on machine wear and tear, and

how the machine can be improved by performing maintenance operations. For this reason, we extend the three-field notation introduced by Graham et al. (1979) to display the model characteristics that differ by the machine state, deterioration, maintenance operations, and the machine performance. While our goal is to present and relate the scope of state-dependent machine availability to other scheduling problems with restrictions of the machine state, our approach is to analyze the papers especially in terms of the runtime complexity to solve the problems. Most of the studied problems deal with single machine layouts. It is remarkable that the assumption of any type of state-dependent machine availability regularly leads to changes in runtime complexity compared to the respective layouts with continuous machine availability. In real-world production environments, the machine state often has an impact on the performance level of the machine since performing jobs deteriorates the machine and so affects its speed when executing subsequent jobs. In research, two main streams can be identified: Models where jobs' processing times are given regardless of the machine state, and models where processing times depend on the machine state. Furthermore, several types of deterioration can be found, which differ in their form of job dependency. Although most studies assume fixed deterioration (each job deteriorates the machine equally), or machine deterioration linear in job's processing times, some works deal with jobs that individually deteriorate the machine state, i.e., no direct processing time-to-deterioration-dependence is given. Overall, the research field of state-dependent machine availability in scheduling still has tremendous potential not only to fill theoretical gaps, but also to generate approaches for practical applications in production and logistics environments.

Chapter 4

Single Machine Scheduling Problems with Position-Dependent Maintenance

In this chapter, the aforementioned machine availability restrictions are considered in such a way that position-dependent (*pd*) maintenance operations must be performed on the machine after a defined number of scheduled jobs at the latest. This type of maintenance can be found in any real-world setting where machine wear and tear depends primarily on the number of scheduled jobs rather than on processing time-dependent factors. For the sake of the text flow, we use *pd* instead of $[0, \overline{s}], \omega, (n, y)$ (introduced in Chapter 2) in the remaining chapter when referring to *pd* maintenance in the Graham notation. Following the work of Hipp and Jaehn (2024b), in Section 4.1 we outline the runtime complexity of several single machine scheduling problems with *pd* maintenance and study their characteristics. In Section 4.2, the single machine scheduling problems of minimizing total weighted completion time and total tardiness of jobs are addressed separately, taking into account release dates and uniform processing times for all jobs. A dynamic program is formulated that solves these single machine problems to optimality in $\mathcal{O}(n^{18})$. Finally, we present the findings of Hipp (2024), where we consider the single machine scheduling problem with *pd* maintenance of minimizing the number of late jobs taking into account ready times, due dates, and uniform processing times.

4.1 Single Machine Problems with Position-Dependent Maintenance

In this section, *pd* maintenance is considered in single machine scheduling layouts. Maintenance operations must be performed after a certain number of jobs, but can also be performed before the machine is completely worn out. Each job causes the same level of deterioration on the machine. In the main layout, studied in Section 4.2, job preemptions (*pmtn*) are not allowed and jobs have uniform processing times. Job j is characterized by weight w_j and release date r_j at which it can be assigned to the machine at the earliest. Besides the practical relevance, our work is motivated by the paper of Drozdowski et al. (2017), who examine several single machine layouts with *pd* maintenance operations regarding their time complexity for the objective functions makespan C_{max} and maximum lateness L_{max} (see Table 4.1). Drozdowski et al. (2017) demonstrate that introducing *pd* machine deterioration and maintenance requirements can completely change the characteristics of scheduling problems.

Tab. 4.1: Complexity results of Drozdowski et al. (2017)

objective \ layout	$\mathbf{1\|pd\|\gamma}$	$\mathbf{1\|pd, r_j\|\gamma}$	$\mathbf{1\|pd, pmtn\|\gamma}$	$\mathbf{1\|pd, r_j, pmtn\|\gamma}$
$\mathbf{C_{max}}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$
$\mathbf{L_{max}}$	$\mathcal{P}$	$\mathcal{NP}$-hard*	$\mathcal{P}$	$\mathcal{NP}$-hard*

*in the strong sense

Apart from Drozdowski et al. (2017), there are only a few papers in the literature that deal with *pd* maintenance operations or characteristics similar to those in this paper. This includes, for example, the work by Hipp (2024) presented in Section 4.3 in which he presents an algorithm for solving the single machine layout of minimizing the number of late jobs considering *pd* maintenance, uniform processing times, and ready times in polynomial time. Whereas Geurtsen et al. (2023) study the influence of positions of maintenance operations on their length in their survey, only time-dependent factors are assumed for machine deterioration and maintenance scheduling. Bock et al. (2012) and Grigoriu and Briskorn (2017) consider another type of maintenance that is related to *pd* maintenance. In their studies, jobs cause individual machine deterioration and mainte-

nance operations can be applied for two goals: (a) restoring or (b) improving the machine state. Machines have a defined upper limit of their state, which represents a status of full functionality. When this state is reached by performing a maintenance operation, we are in (a). Furthermore, in (b), the machine state can also be improved beyond the mentioned state of full functionality to reach a new performance level of the machine. Calinescu et al. (2017) focus on single machine scheduling problems, where a special job must be sequenced at a specific position in the schedule. This special job can be interpreted as a variant of *pd* maintenance scheduling, where the job displays a single *pd* maintenance operation, but fixed in its position in the final sequence and influencing the objective function value. A dynamic program with pseudo-polynomial runtime is formulated leading to a fully polynomial-time approximation scheme. Based on the work of Calinescu et al. (2017), Jaehn (2024) presents a more general case of jobs with fixed positions, which is addressed in Chapter 5. Here, a subset of special jobs must be scheduled at predefined positions in the final sequence. These jobs are described by the same characteristics as the remaining set of jobs with the same influence on the objective function. Finally, Mosheiov (2011), Agnetis and Mosheiov (2017), and Fiszman and Mosheiov (2018) provide relevant models that help understand the functionality of *pd* maintenance scheduling from a different perspective. No type of machine unavailability is directly addressed in their works, but *pd* processing times of jobs are considered that are not connected to typical learning (Biskup (1999)) or aging effects (Mosheiov (2005)). This means that the processing times are unique and depend on their position in the schedule so that the sequencing of each job is similarly significant as in our case of *pd* maintenance scheduling. These three papers show that proportionate flow shops, including, inter alia, the job characteristics described, are solvable in polynomial time. Table 4.2 gives an overview of the aforementioned papers dealing with any type of *pd* machine (un-)availability or related problems.

With regard to the single machine layout that is studied in Section 4.2, Baptiste (2000) also considers single machine scheduling problems with uniform processing times and release dates, but with continuous machine availability. He presents an $\mathcal{O}(n^7)$ dynamic programming algorithm for the single machine case and $\mathcal{O}(n^{(3m+4)})$ algorithms for parallel machine layouts with m machines to minimize total weighted completion time. In addition, Baptiste (2000) adapts this approach to minimize total tardiness.

Tab. 4.2: Relevant literature dealing with types of position-dependent maintenance

Specification	**Direct influence on objective function**	**Literature**
Job(s) with fixed position	Yes	Calinescu et al. (2017), Jaehn (2024)
Position-dependent processing times of jobs (with no specific function)	Yes	Agnetis and Mosheiov (2017), Fiszman and Mosheiov (2018), Mosheiov (2011)
Proportional deterioration depending on number of jobs	No	Drozdowski et al. (2017), Hipp (2024)
Specific deterioration per job, machine improvement possible	No	Bock et al. (2012), Grigoriu and Briskorn (2017)

The remainder of this section is organized as followed. The machine layout and optimization problem, which is addressed in Section 4.2, are presented before outlining properties of this specified scheduling problem. Finally, single machine problems considering *pd* maintenance, ready times, and/or job preemption are studied regarding their runtime complexity.

4.1.1 Problem Description

We are given a set of n jobs, each of which has to be scheduled on one machine. Each job $j \in \{1, \ldots, n\}$ is defined by its processing time $p_j = p$, which is identical for all jobs, a release date r_j, and a weight w_j. Without loss of generality (as jobs can be reindexed accordingly), we assume that $w_j \geq w_{j+1}$ for all $j \in \{1, \ldots, n-1\}$. Furthermore, we are given two positive integers k and s.

The problem is now to schedule the n jobs, i.e., we are to define each job's completion time C_j. A schedule is called *feasible* if the following constraints are met:

1. No two jobs j and j' (both from the set $\{1, \ldots, n\}$ and $j \neq j'$) must overlap. That is, $|C_j - C_{j'}| \geq p$ must hold.

2. For each $j \in \{1, \ldots, n\}$, $C_j - p \geq r_j$ must hold.

3. After scheduling at most k jobs, there must be an idle period on the machine for maintenance of at least s time units. This can be expressed as follows: Let π_j be the position of job j in the schedule (which is obviously unique), so that $(\pi_1^{-1}, \ldots, \pi_n^{-1})$ defines the job sequence of the schedule. Then for every set $A \in \{\{1, \ldots, k\}, \{2, \ldots, k+1\}, \ldots, \{n-k, \ldots, n-1\}\}$, there must be a $j \in A$ so that $C_{\pi_{j+1}^{-1}} - C_{\pi_j^{-1}} \geq p + s$.

Among the feasible schedules, we seek one which minimizes the weighted sum of completion times $\sum_{j=1}^{n} w_j C_j$ (also referred to as weighted flow time). Using the three field notation introduced by Graham et al. (1979), this problem can be expressed by $1|pd, r_j, p_j = p|\sum w_j C_j$. Note that feasibility can easily be obtained for any job sequence $(\pi_1^{-1}, \ldots, \pi_n^{-1})$ using the recursion $C_{\pi_1^{-1}} = \min\{0, r_{\pi_1^{-1}}\} + p$, $C_{\pi_j^{-1}} = \min\{C_{\pi_{j-1}^{-1}}, r_{\pi_1^{-1}}\} + p + s$ for all $j \in \{2, \ldots, n\}$. Table 4.3 shows the basic notation for our problem.

Tab. 4.3: Basic notation for $1|pd, r_j, p_j = p|\sum w_j C_j$

n	Number of jobs, $n \in \mathbb{N}$
j	Job index, $j \in \{1, \ldots, n\}$
k	Maximum number of jobs before maintenance must be scheduled, $k \in \mathbb{N}$
s	Length of maintenance operation, $s \in \mathbb{N}$
p	Processing time for all n jobs, $p \in \mathbb{N}$, $p_j = p$
r_j	Release date of job j, $r_j \in \mathbb{N}_0$
w_j	Weight of job j, $w_j \in \mathbb{N}$
C_j	Completion time of job j, $C_j \in \mathbb{N}$

4.1.2 Problem Analysis

Based on the description in Section 4.1.1, we analyze the problem $1|pd, r_j, p_j = p|\sum w_j C_j$ regarding its optimal solutions for any given instance before formulating an algorithm to solve it to optimality in Section 4.2. In general, optimal schedules of scheduling problems minimizing total weighted completion time are left-shifted, i.e., we have a semi-active schedule (Pinedo (2022)). In addition, the following properties can be formulated.

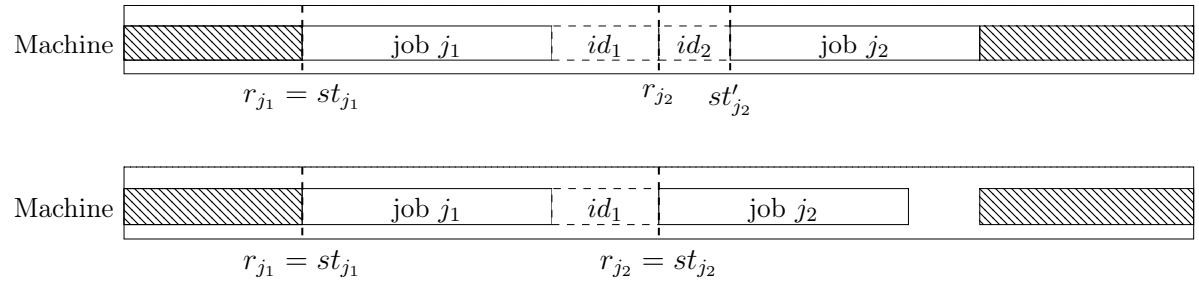
Proposition 4.1.1. *Every optimal schedule contains only idle times that end at some time instant t with a job j, which starts at t with $r_j = t$.*

To illustrate this property, we consider two jobs, j_1 and j_2 with $j_1, j_2 \in \{1, \ldots, n\}$, that are scheduled consecutively in the optimal sequence. We also consider a variant of this schedule, in which no maintenance is scheduled between j_1 and j_2 and a second variant in which a maintenance operation is scheduled between them. In the following, we show that in an optimal schedule, there are no idle times other than those that end with the start of a job on its release date.

Proof. We first investigate Proposition 4.1.1 for the case that no maintenance operation is scheduled between jobs j_1 and j_2. It is given $w_{j_1} > w_{j_2}$, r_{j_1}, and r_{j_2}.

Consider $r_{j_1} \leq r_{j_2}$. Since we know that every optimal schedule of the problem $1|\beta|\sum w_j C_j$ is left-shifted (Pinedo (2022)), job j_1 is scheduled as soon as possible. Assuming an idle machine, start time st_{j_1} of job j_1 is equal to r_{j_1}. Following this idea, job j_2 has to be scheduled at time $st_{j_2} = \max\{st_{j_1} + p, r_{j_2}\}$. If $r_{j_2} > st_{j_1} + p$ and no further job or maintenance operation is available to be scheduled, $st_{j_2} > st_{j_1} + p$ causing idle time on the machine. In an optimal schedule, this idle time $id_1 = r_{j_2} - (st_{j_1} + p)$ cannot be prevented. Job j_1 could be right-shifted to be completed at time st_{j_2} to prevent id_1, but this would obviously not lead to an optimal solution anymore. Any right-shifting of j_2 (in this scenario, no left-shifting of j_2 is feasible) with start time $s'_{j_2} > st_{j_2}$ leads to idle time $id_2 = s'_{j_2} - r_{j_2}$. In this case, the schedule is not optimal because $w_{j_2}(st_{j_2}+p) < w_{j_2}(s'_{j_2}+p)$ (see Gantt charts in Figure 4.1). For $r_{j_2} \leq st_{j_1} + p$, j_2 starts directly after j_1 is completed and in an optimal solution no idle time occurs at all.

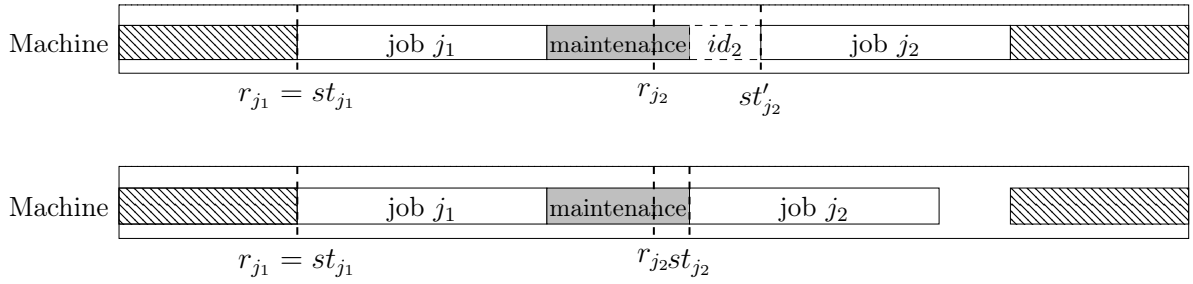
Fig. 4.1: Illustration of Proposition 4.1.1 without maintenance operation: Schedule, in which job j_2 can be shifted left to obtain an optimal subschedule and prevent idle time



For $r_{j_1} > r_{j_2}$ it is trivial to show that the same arguments hold. In an optimal solution, idle time only occurs because of the release date of the later sequenced job.

When a maintenance operation is sequenced in an optimal schedule between jobs j_1 and j_2, s as the length of the maintenance operation must be considered regarding feasible start times of the jobs and possible idle times. In general, the results for the case without performing a maintenance operation remain. Compared to the above considerations regarding id_1 and id_2, s reduces both types of idle times (id_1 and id_2) or even prevents them from occurring. Figure 4.2 illustrates an example with idle time id_2 or no idle time for $r_{j_1} < r_{j_2}$. □

Fig. 4.2: Illustration of Proposition 4.1.1 with maintenance operation



Proposition 4.1.2. *In an optimal schedule including two jobs j_1 and j_2 with $j_1, j_2 \in \{1, \ldots, n\}$, characterized by $w_{j_1} > w_{j_2}$ and $r_{j_1} \leq r_{j_2}$, it holds $st_{j_1} < st_{j_2}$ for the jobs' start times.*

Proof. We show the feasibility of this property by contradiction. Assuming an optimal schedule S in that job j_2 is scheduled before job j_1. Without loss of generality, we can further assume that no other job or maintenance operation is scheduled between these jobs. The start times are then given by $st_{j_2} = st$ and $st_{j_1} = st+p$, the completion times by $C_{j_2} = st+p$ and $C_{j_1} = st+p+p$, which results in $w_{j_2}(st+p)$ and $w_{j_1}(st+p+p)$ for w_jC_j. We can switch these jobs regarding their position without influencing completion times of any earlier or later scheduled jobs. By switching the positions of j_1 and j_2, we get the new schedule S' with $w_{j_1}(st+p)$ and $w_{j_2}(st+p+p)$. Comparing the sums of objective function values of both jobs and schedules, we get $w_{j_2}(st+p)+w_{j_1}(st+p+p) = w_{j_2}st+w_{j_2}p+w_{j_1}st+w_{j_1}p+w_{j_1}p$ for S and $w_{j_1}(st+p)+w_{j_2}(st+p+p) = w_{j_1}st+w_{j_1}p+w_{j_2}st+w_{j_2}p+w_{j_2}p$ for S'. Finally, we have to compare $w_{j_1}p$ and $w_{j_2}p$. Following the assumption of this property, it must hold that $w_{j_1}p > w_{j_2}p$. Then, the objective function value of S' is lower than the one of S what contradicts the statement that schedule S is an optimal solution.

When we consider a maintenance operation between jobs j_1 and j_2, the final result remains the same even though the length of the maintenance operation s is added to the terms of the start and completion times. The objective function values are $w_{j_2}(st + p) + w_{j_1}(st + p + s + p)$ and $w_{j_1}(st + p) + w_{j_2}(st + p + s + p)$ for schedules S and S' so that we have to compare $w_{j_1}(s + p)$ and $w_{j_2}(s + p)$. The objective function value of S' is obviously lower than the one of S, which again contradicts the statement that S is an optimal schedule. □

Proposition 4.1.3. *In an optimal schedule, a maintenance operation is scheduled after less than k jobs if and only if there is machine idle time (caused by r_j, see Proposition 4.1.1) between two of these k jobs before the positions of the maintenance operations are determined. In this case, the position of the required maintenance operation(s) has to be determined via a sequencing method (if the maintenance is scheduled in idle time, the considered idle time is not necessarily equal to or greater than s time units).*

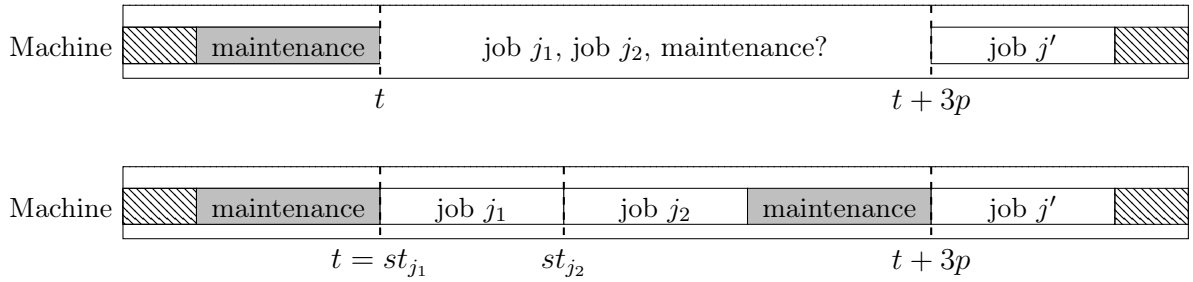
By assuming two jobs j_1 and j_2 with $j_1, j_2 \in \{1, \ldots, n\}$, we illustrate on the basis of two cases that the position of a maintenance operation can be determined for an optimal schedule only regardless of the other $n - 2$ jobs and maintenance operations, if no idle time occurs between these two jobs.

Proof. In the following, jobs j_1 and j_2, $w_{j_1} \geq w_{j_2}$, $k = 2$, and $p_j = s = p$ are given. Further, a maintenance operation is scheduled on the machine ending at time t and the machine is free to schedule both jobs and a maintenance operation from t to time $t + 3p$. Without loss of generality we assume that a job $j' \neq j_1, j_2$ with $r_{j'} = t + 3p$ starts at $r_{j'}$ on the machine. As a result, at least one maintenance operation must be scheduled (because of $k = 2$ and j') after j_1 or j_2 if j_1 and j_2 shall both be sequenced between t and $t + 3p$.

In case I, $r_{j_1} = t$ and $r_{j_2} \in (t, r_{j_1} + p]$ are given. Because of the given weights, it must hold $st_{j_1} < st_{j_2}$ (see Proposition 4.1.2) for the start times of the jobs in an optimal schedule. In addition, it is trivial to show that the sequence $(j_1, j_2, \text{maintenance})$ leads to an optimal schedule for the interval $[t, t + 3p]$ with start times $st_{j_1} = t$ and $st_{j_2} = t + p$, and $t + 2p$ as the start time of the maintenance operation. By scheduling the maintenance in $t + 2p$, the machine is totally restored at time $3t$ and can potentially perform k jobs including j' under consideration of the characteristics of j' and further jobs.

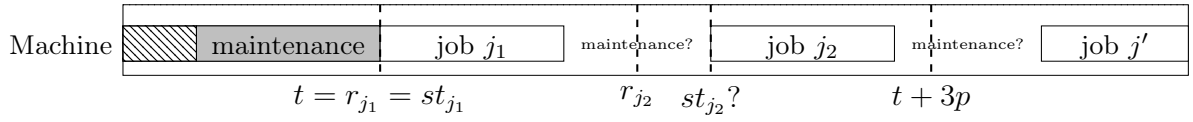
Besides, the objective function values of j_1 and j_2 are obviously optimal with $w_{j_1}(t + p) + w_{j_2}(t + 2p)$. Figure 4.3 displays this case.

Fig. 4.3: Illustration of Proposition 4.1.3, case I



For case II, we assume $st_{j_1} = r_{j_1} = t$ and $r_{j_1} + p < r_{j_2} \leq 3t - p$. In an optimal schedule, the machine is idle after completing j_1 and before starting j_2. In this case, the sequencing decisions regarding j_2 and the maintenance operation are not clear. If $r_{j_2} = st_{j_1} + p + s$, that is a maintenance operation can be scheduled between j_1 and j_2 without shifting j_2 and jobs or maintenance operations to the right following $t + 3p$, $(j_1, \text{maintenance}, j_2)$ is a feasible sequence as part of an optimal solution. The completion time of j_2 and, as a result, subsequent jobs are not affected. Note that minimizing the number of maintenance operations is not part of the optimization. If $r_{j_2} < st_{j_1} + p + s$, the sequencing decision in not clear. Depending on the determined sequence, j_1, j_2, and the maintenance operation may consume more machine runtime than $3p$ so that j' cannot start at its ready time. To determine the positions of the maintenance operation(s) and the jobs, we require the algorithm formulated in Section 4.2, because the respective parameter values of j_1, j_2 and following jobs must be considered to find an optimal solution. Figure 4.4 illustrates this issue. □

Fig. 4.4: Illustration of Proposition 4.1.3, case II



4.1.3 Complexity of Single Machine Scheduling Problems with Position-Dependent Maintenance

After studying the characteristics of the scheduling problem $1|pd, r_j, p_j = p|\sum w_j C_j$, we focus on related single machine layouts in consideration of *pd* maintenance.

Theorem 4.1.1. *The problem of minimizing total completion time* $1|pd|\sum C_j$ *is solvable in* polynomial time *by sorting all jobs with shortest processing time (SPT) rule and scheduling maintenance operations as late as possible (after k jobs).*

We proof this theorem by contradiction. For the single machine layout with continuous machine availability, $1||\sum C_j$, Smith et al. (1956) formulate a similiar proof applying contradiction to show the polynomial runtime for the appropriate problem.

Proof. We assume to have two jobs j_1 and j_2 with $j_1, j_2 \in \{1, \ldots, n\}$ and processing times $p_{j_1} < p_{j_2}$. An optimal schedule S is given in that j_2 is scheduled before job j_1. In addition, we can assume without loss of generality that no further job or maintenance operation is scheduled between j_2 and j_1. To perform both jobs, we need $p_{j_2} + p_{j_1}$ time units of idle time on the machine. Considering that such an interval starts at t so that start times $st_{j_2} = t$ and $st_{j_1} = t + p_{j_2}$, and completion times $C_{j_2} = t + p_{j_2}$ and $C_{j_1} = t + p_{j_2} + p_{j_1}$ result. The objective function value for these jobs is $2t + 2p_{j_2} + p_{j_1}$. Without influencing any completion times for previous or subsequent scheduled jobs or maintenance operations, we can switch the positions of j_1 and j_2. We refer to this new schedule as S'. In S', we have $C_{j_1} = t + p_{j_1}$ and $C_{j_2} = t + p_{j_1} + p_{j_2}$, which leads to an objective function value of $2t + 2p_{j_1} + p_{j_2}$. To evaluate S and S', we compare both objective function values, $2t + 2p_{j_2} + p_{j_1}$ and $2t + 2p_{j_1} + p_{j_2}$. We can truncate these terms so that we only have to compare p_{j_2} (S) and p_{j_1} (S'). Following $p_{j_1} < p_{j_2}$, which we assumed at the beginning, the objective function value of S' is obviously smaller than the one of S. This contradicts the statement that S is an optimal schedule. The resorting of jobs according to non-increasing p_j can be translated to the overall schedule of any instance of $1|pd|\sum C_j$. As a result, we can conclude that, in an optimal schedule of the problem $1|pd|\sum C_j$, the jobs are sorted in non-increasing order regarding p_j, i.e., we apply SPT rule.

If we consider a maintenance operation between j_1 and j_2, the result and conclusion remain the same. In this case, s as the length of the maintenance operation must be

considered when calculating the start and completion times that lead to the objective function values $2t + 2p_{j_2} + s + p_{j_1}$ for S, and $2t + 2p_{j_1} + s + p_{j_2}$ for S'. After simplifying the terms, only p_{j_1} and p_{j_2} remain as relevant parameters to compare, which are the same terms as in the previous remarks. As a result, SPT rule also leads to optimal schedules when maintenance operations are scheduled between two jobs.

Finally, we have already shown in Proposition 4.1.3 that maintenance operations are scheduled after k jobs in a machine layout where no idle time occurs in an optimal schedule.

□

Theorem 4.1.2. *The problem of minimizing total weighted completion time* $1|pd|\sum w_jC_j$ *is solvable in* polynomial time *by sorting all jobs with weighted shortest processing time (WSPT) rule and scheduling maintenance operations as late as possible (after k jobs).*

WSPT rule, or "Smith rule", presented by Smith et al. (1956), applies also to the case of *pd* maintenance operations. The proof of Theorem 4.1.1 can be modified by replacing p_j with $\frac{p_j}{w_j}$ (weight w_j per unit of processing time p_j).

Theorem 4.1.3. *The single machine problem* $1|pd|\sum T_j$ *of minimizing total tardiness with due date* d_j *and tardiness* $T_j = \max_{j\in\{1,\dots,n\}}\{C_j - d_j, 0\}$ *is* $\mathcal{NP}$-hard *in the ordinary sense.*

Further, the scheduling problems of minimizing total weighted tardiness $1|pd|\sum w_jT_j$*, minimizing total completion time in consideration of release dates* $1|pd, r_j|\sum C_j$*, and the problem of minimizing total weighted completion time with release dates* $1|pd, r_j|\sum w_jC_j$ *are* $\mathcal{NP}$-hard *in the strong sense.*

The mentioned problems of Theorem 4.1.3 are already $\mathcal{NP}$-hard in the original setting with continuous machine availability.

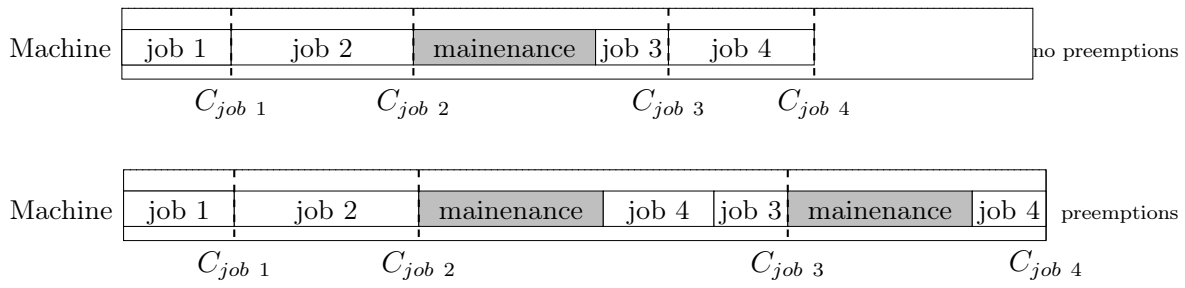
Du and Leung (1990) reduce even odd partition to $1||\sum T_j$ to prove that $1||\sum T_j$ is $\mathcal{NP}$-hard in the ordinary sense. This proof also holds for $1|pd|\sum T_j$ if we set the length of maintenance operations $s = 0$ for any given instance. For the problem $1||\sum w_jT_j$, Lawler (1977) present a reduction to 3-partition that is $\mathcal{NP}$-hard in the strong sense. By assuming $s = 0$ again, the same proof holds for $1|pd|\sum w_jT_j$. Lenstra et al. (1977) formulate the proof that the problem $1|r_j|\sum C_j$ is $\mathcal{NP}$-hard in the strong sense.

Their proof by reduction to 3-partition can be assigned to $1|pd, r_j|\sum C_j$ for $s = 0$. By assuming $w_j = 1$ for job j, these findings also hold for $1|pd, r_j|\sum w_j C_j$.

Theorem 4.1.4. *Minimizing total completion time whereas job preemption is allowed,* $1|pd, pmtn|\sum C_j$*, is solvable in* polynomial time *by sorting all jobs with SPT rule and scheduling maintenance operations as late as possible (after k jobs). Preemptions do not apply.*

It is easy to illustrate that preemptions do not improve solutions of the problem $1|pmtn|\sum C_j$ (see, e.g., Pinedo (2022)). If we consider *pd* maintenance, i.e., focusing on $1|pd, pmtn|\sum C_j$, preemptions can even increase the value of total completion time. The number of assigned maintenance operations potentially increase as a higher number of tasks (tasks correspond to jobs or parts of jobs) leads to a higher demand for maintenance operations due to preemption. These additional maintenance operations, which consume time on the machine, result in later job completion times compared to scheduling jobs without preemption. Machine idle times also do not reason the requirement of job preemption because idle times do not occur if no release dates are assumed or if all jobs are available at the beginning of the planning horizon. Figure 4.5 shows a small example with four jobs and $k = 2$ to demonstrate the potential negative impact of preemption for this layout.

Fig. 4.5: Example with four jobs and $k = 2$ *for* $1|pd, pmtn|\sum C_j$



Theorem 4.1.5. *The problem of minimizing total weighted completion time with preemptions* $1|pd, pmtn|\sum w_j C_j$ *is solvable in* polynomial time *by sorting all jobs with WSPT rule and scheduling maintenance operations as late as possible (after k jobs). Preemptions are not applied (see explanations for* $1|pd, pmtn|\sum C_j$*).*

We have already shown in Theorem 4.1.3 that $1|pd|\sum T_j$ is $\mathcal{NP}$-hard in the ordinary sense. If we allow job preemptions, the same applies as described for $1|pd, pmtn|\sum C_j$. Job preemptions do not lead to an improvement of the objective function value, so that the complexity results of $1|pd|\sum T_j$ also hold for $1|pd, pmtn|\sum T_j$:

Theorem 4.1.6. *The problem of minimizing total tardiness and allowing preemptions* $1|pd, pmtn|\sum T_j$ *is* $\mathcal{NP}$-hard *in the ordinary sense.*

Theorem 4.1.7. *The problem of minimizing total weighted completion time with release dates and preemptions*
$1|pd, r_j, pmtn|\sum w_jC_j$ *is* $\mathcal{NP}$-hard *in the strong sense.*

It is proved by Labetoulle et al. (1984) using 3-partition that $1|r_j, pmtn|\sum w_jC_j$ is $\mathcal{NP}$-hard in the strong sense. This proof also holds for the problem $1|pd, r_j, pmtn|\sum w_jC_j$. If we consider *pd* maintenance operations, we can, without loss of generality, set $s = 0$ for any given instance. As a result, $1|pd, r_j, pmtn|\sum w_jC_j$ is $\mathcal{NP}$-hard in the strong sense as well.

In Table 4.4, we summarize the findings of Section 4.1.3 dealing with the runtime complexity of single machine problems with *pd* maintenance.

Tab. 4.4: Complexity results presented in Section 4.1.3 including the respective theorem

objective \ layout	$\mathbf{1\|pd\|\gamma}$	$\mathbf{1\|pd, r_j\|\gamma}$	$\mathbf{1\|pd, pmtn\|\gamma}$	$\mathbf{1\|pd, r_j, pmtn\|\gamma}$
$\sum \mathbf{C_j}$	$\mathcal{P}$ (4.1.1)	$\mathcal{NP}$-hard* (4.1.3)	$\mathcal{P}$ (4.1.4)	
$\sum \mathbf{w_jC_j}$	$\mathcal{P}$ (4.1.2)	$\mathcal{NP}$-hard* (4.1.3)	$\mathcal{P}$ (4.1.5)	$\mathcal{NP}$-hard* (4.1.7)
$\sum \mathbf{T_j}$	$\mathcal{NP}$-hard (4.1.3)		$\mathcal{NP}$-hard (4.1.6)	
$\sum \mathbf{w_jT_j}$	$\mathcal{NP}$-hard (4.1.3)		$\mathcal{NP}$-hard (4.1.6)	

*in the strong sense

4.2 Minimizing Total Weighted Completion Time with Position-Dependent Maintenance, Release Dates, and Uniform Processing Times

As mentioned above, Baptiste (2000) presents an $\mathcal{O}(n^7)$ algorithm for $1|r_j, p_j = p|\sum w_j C_j$, which displays the studied problem without any type of machine availability constraint. We enhance the ideas of this approach to generate an exact algorithm with runtime $\mathcal{O}(n^{18})$ for the problem $1|pd, r_j, p_j = p|\sum w_j C_j$.

4.2.1 Notation

In addition to the notation introduced in Section 4.1.1 (see Table 4.3), we use the notation listed in Table 4.5. According to Proposition 4.1.1, every feasible schedule can be converted into a schedule that includes only idle times induced by release date r_j without increasing the objective function value. This means that, without loss of generality, every idle time ends at some job j starting at r_j. As a result, the considered schedule can be constrained such that set T of all feasible start and completion times of all jobs in a feasible schedule is defined by the number of jobs, maintenance operations and release dates. Without any maintenance operations and each $r_j = 0$, all start and completion times are multiples of p including zero as the earliest start time (a). Considering pd maintenance operations, they are linear combinations of p and s (b). Due to the release dates, the start and completion times are linear combinations that can be added to any r_j (c). Therefore, job $j' \in \{0, \ldots, n\}$ (a), maintenance operation $m \in \{0, \ldots, n-1\}$ (b), and release date r_j (c) are considered to define T as follows:

$$T := \{t : \exists r_j \forall j \in \{1, \ldots, n\}, \exists j' \in \{0, ..., n\}, \exists m \in \{0, ..., n-1\}, m \leq j', t = r_j + j' \cdot p + m \cdot s\}.$$

$f_j(t)$ denotes the weighted completion time with $f_j(t) = w_j \cdot C_j$ for each completion time $t = C_j \in T$ of job j with weight w_j. Since one job is considered to be scheduled in each iteration, we have a total of $n + 1$ *stages* in the dynamic program, including the initialization phase of the algorithm. The stages are indicated by $l \in \{0, \ldots, n\}$.

A *state* on stage l is denoted by

$$(\alpha, \beta, b, e, x)_l$$

with $\alpha, \beta \in \{0, \ldots, k\}$, $b, e \in T, b \leq e$, and $x \in \{0, 1\}$.

The value function of a state $(\alpha, \beta, b, e, x)_l$ is denoted by $F_l(\alpha, \beta, b, e, x)$. By abuse of notation, we define $\min \emptyset := \inf \emptyset = \infty$.

Tab. 4.5: Notation of the dynamic program for $1|pd, r_j, p_j = p| \sum w_j C_j$

l	Index for stage of dynamic program and index for selected job in stage l, $l \in \{0, \ldots, n\}$ with $l = 0$ as the initialization of the algorithm
T	Set of time points t based on r_j induced idle times, number of jobs and maintenance operations
t	Time point at which jobs possibly start and end, $t \in T$
b, e	Start and end of an interval in which jobs and maintenance operations are scheduled, $b, e \in T,\ b \leq e$
t_j	Starting time of job j, $t_j \in T$
C_j	Completion time of job j, $C_j \in T$
α	Number of scheduled jobs prior to the first maintenance in an interval, $\alpha \in \{0, \ldots, k\}$
β	Number of scheduled jobs after the last maintenance in an interval, $\beta \in \{0, \ldots, k\}$
u, v	Auxiliary variable that replaces α or β in stage $l-1$ (introduced in Section 4.2.3)
x	Binary variable; $x = 1$ if at least one maintenance operation is assigned in some interval $[b, e)$, $x \in \{0, 1\}$
f_j	Objective function value of job j
F_l	Value function (total weighted completion time) in stage l
Θ_l	Set of jobs scheduled in stage l
π_j	Position of job j in the schedule

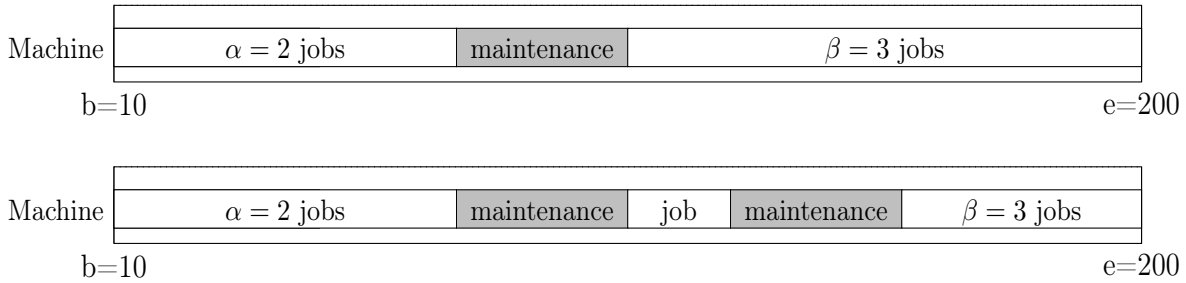
4.2.2 General Idea

A state $(\alpha, \beta, b, e, x)_l$ considers all jobs $j \in \{1, \ldots, l\}$ with $b - p \leq r_j < e$ to be optimally scheduled within time interval b and e in stage l so that

- there is at least one maintenance operation scheduled in the interval from b to e, if $x = 1$,
- there is no maintenance operation scheduled in the interval from b to e, if $x = 0$,
- α jobs appear in the subinterval of b and the start of the first maintenance operation within interval b and e (for $x = 1$),
- β jobs appear between the last maintenance operation of the interval and e (for $x = 1$),
- there are $\alpha = \beta$ jobs within the interval from b to e for $x = 0$.

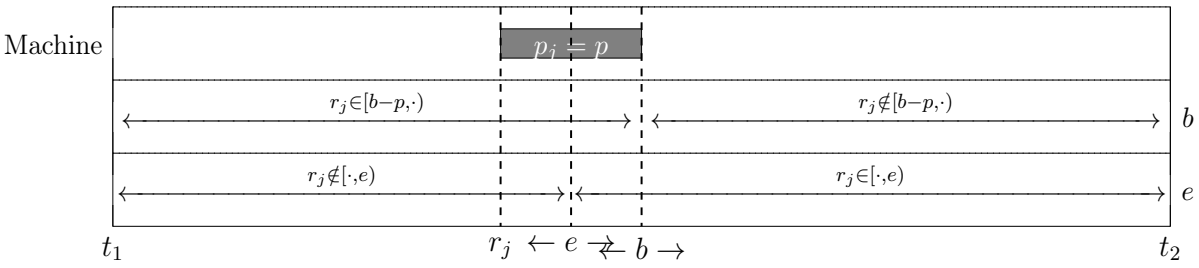
For example, $(2, 3, 10, 200, 1)_6$ indicates a state on stage 6 (i.e., a maximum 6 jobs might be scheduled) for time interval $[10, 200)$. In this state, two jobs can be scheduled prior to the first maintenance operation ($\alpha = 2$) and three jobs follow after the last maintenance ($\beta = 3$). Without any information about the states $l - 1$, this expression with $x = 1$ does not show the exact number of jobs possibly planned in this interval, but we know that the number of jobs is greater than or equal to $\alpha + \beta$ and less than or equal to l. Here, 5 or 6 jobs can be scheduled in $[10, 200)$ (see Gantt charts in Figure 4.6, where potential idle times are neglected). Similarly, $(1, 1, 10, 200, 0)_6$ indicates that only one job can be scheduled in $[10, 200)$ without any maintenance operation within the interval. State $(1, 2, 10, 200, 0)_6$ is not well-defined as $x = 0$ requires $\alpha = \beta$ jobs in total within the interval. Later, when states are connected in the graph of the dynamic program, we will see that such a state is never reached.

Fig. 4.6: Possible Gantt charts of state $(2, 3, 10, 200, 1)_6$



$F_l(\alpha, \beta, b, e, x)$ denotes the best value function of a partial schedule following the above mentioned constraints in state l. This partial schedule within interval b and e in state l is feasible only if $b - p \leq r_j < e$ holds for each job j in this schedule. Job j' with $r_{j'} \notin [b - p, e)$ is not part of such a schedule to obtain a left-shifted schedule. The influence of this constraint is visualized in Figure 4.7.

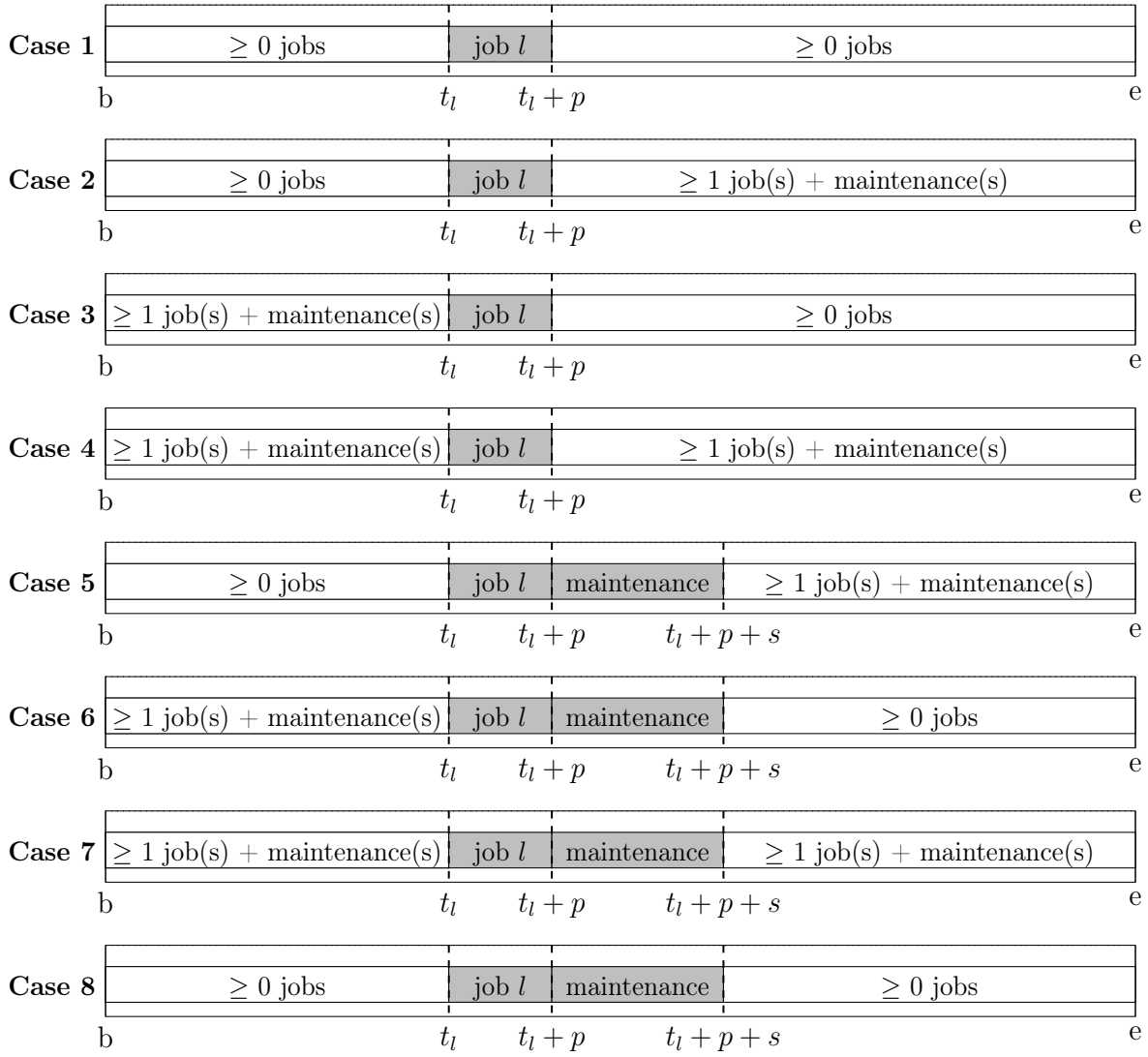
Fig. 4.7: Illustration of constraint $r_j \in [b - p, e)$



If there is no feasible state, we define $F_l(\alpha, \beta, b, e, x) = \infty$. To determine $F_l(\alpha, \beta, b, e, x)$, we need states $l - 1$ for all combinations of b and e with $b \leq e$. The first step is to schedule job l in state $(\alpha, \beta, b, e, x)_l$ at start time t_l within the specified interval. Job l is scheduled in stage l only if $r_l \in [b - p, e)$. If $r_l \notin [b - p, e)$, then job l is not scheduled and $F_l(\alpha, \beta, b, e, x) := F_{l-1}(\alpha, \beta, b, e, x)$. For $r_l \in [b - p, e)$, $f_l(t_l + p)$ is calculated with $t_l \in [\max\{b, r_l\}, e - p]$. Whenever job l can be assigned in interval $[b, e)$ at time t_l, we have to distinguish whether a maintenance operation is sequenced together with job l and whether maintenance operations are already scheduled in intervals $[b, t_l)$, and $[t_l + p, e)$ or $[t_l + p + s, e)$. In total, there are eight feasible cases, which are illustrated in Figure 4.8. Assuming that a given job l (and a maintenance operation) is scheduled at t_l, no maintenance operations may have yet been sequenced between b and e (cases 1 and 8). Otherwise, one or more maintenance operations may be already scheduled before t_l (cases 3 and 4), after completing job l (cases 2 and 5), or before and after job l (cases 4 and 7). In summary, cases 1 to 4 in Figure 4.8 depicts the feasible schedules when only job l is assigned in $[b, e)$, while cases 5 to 8 show analogously the cases when job l is scheduled together with a maintenance operation. The eight cases are required because k restricts the number of feasible solutions for scheduling jobs and showing the feasible combinations of the positions of jobs and maintenance operations. Note that these cases hold for each stage $l \in \{1, \ldots, n\}$. We assume that a maintenance operation can only be

assigned together with a job, so that at least one job must already be scheduled within an interval if at least one maintenance operation ($x = 1$) is sequenced in it (see cases 2, 4, 5, 6, and 7 in Figure 4.8).

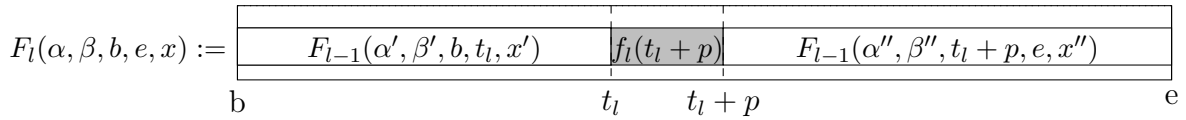
Fig. 4.8: Cases of scheduling job l and a maintenance operation (in gray) in stage l



Dynamic programs are characterized, inter alia, by the fact that in iteration l we only need the information of the previous iteration $l - 1$. Since we are dealing with intervals in our approach in which several jobs might be already scheduled by reaching stage l, we look at job $\varphi \in \{1, \ldots, l-1\}$ that is already scheduled within interval b and e up to stage $l - 1$. The following ideas also apply to schedules including maintenance operations.

Given state $l-1$, any job φ with $r_\varphi \in [b-p, e)$ is scheduled either before job l in interval $[b, t_l)$ or after job l in $[t_l+p, e)$ in $(\alpha, \beta, b, e, x)_l$. Constraint $r_\varphi \in [b-p, e)$ ensures that job φ is only assigned in exactly one of the two subintervals. If $r_\varphi \in [b-p, e)$, either $r_\varphi \in [b-p, t_l)$ and $r_\varphi \notin [t_l, e)$ applies or vice versa, $r_\varphi \notin [b-p, t_l)$ and $r_\varphi \in [t_l, e)$. The minimum value of $F_l(\alpha, \beta, b, e, x)$ is finally determined by minimizing the sum of $f_l(t_l+p)$, $F_{l-1}(\alpha', \beta', b, t_l, x')$, and $F_{l-1}(\alpha'', \beta'', t_l+p, e, x'')$ (see Figure 4.9) with $\alpha', \alpha'', \beta', \beta'', x', x''$ as values representing the number of jobs before and after maintenance operations in stage $l-1$ for the respective interval, but which may differ from α and β in stage l. Further, it must hold $x \geq \{x', x''\}$. A detailed description of the recursion is outlined in Section 4.2.3, where we introduce u and v to replace α and β when necessary.

Fig. 4.9: Composition of value function $F_l(\alpha, \beta, b, e, x)$

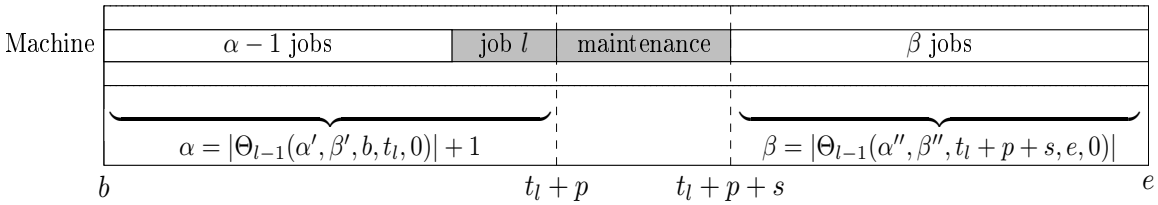


In stage l, state $(\alpha, \beta, b, e, x)_l$ is generated for every combination of b and e with $b \leq e$. For $(\alpha, \beta, \min\{T\}, \max\{T\}, x)_l$ with T as the set of all time points t, we assume that exactly l jobs are assigned within $\min\{T\}$ and $\max\{T\}$ or it must hold $F_l(\alpha, \beta, \min\{T\}, \max\{T\}, x) = \infty$. Referring to set T, the earliest start of an interval is $min\{T\} = \min\{r_j\}$ and the latest end is $\max\{T\} > \max\{r_j\}$ with $j \in \{1, \ldots, n\}$. Hence, $r_j \in [\min\{T\} - p, \max\{T\})$ holds for any job j as well as $r_{j'} \in [\min\{T\} - p, \min\{T\})$ for $j' \in \{1, \ldots, l\}$. As a result, l jobs must be scheduled within interval $b = \min\{T\}$ and $e = \max\{T\}$ in state l.

To illustrate the allocation of assigned jobs in stage l, let $\Theta_l(\alpha, \beta, b, e, x)$ be the set of scheduled jobs in state $(\alpha, \beta, b, e, x)_l$ with $|\Theta_l(\alpha, \beta, b, e, x)|$ as the number of jobs in that state. In general, $|\Theta_l| := |\Theta_{l-1}| + 1$ for a given state in an interval if job l can be scheduled within $[b, e)$, i.e. if $r_l \in [b-p, e)$. As long as no maintenance is scheduled, so we have $x = 0$ and state $(\alpha, \beta, b, e, 0)_l$, α is equal to β and as a result, $|\Theta_l(\alpha, \beta, b, e, 0)|$ corresponds directly to $\alpha = \beta$. If a maintenance operation is scheduled in stage l along with job l ($x = 1$), and no maintenance operations are assigned in the corresponding states of the previous stage $l-1$ ($x' = x'' = 0$), set $\Theta_{l-1}(\alpha', \beta', b, t_l, 0)$ contains $\alpha - 1 = \alpha'$ jobs except job l, which is assigned at time t_l. $\beta = \alpha''$ jobs are scheduled at the interval $[t_l + p + s, e)$

and are therefore in set $\Theta_{l-1}(\alpha'', \beta'', t_l + p + s, e, 0)$. Figure 4.10 illustrates this case. If more than one maintenance is scheduled in state $(\alpha, \beta, b, e, 1)_l$, we only know with the given information that $|\Theta_l(\alpha, \beta, b, e, 1)| > \alpha + \beta$ jobs are scheduled within interval b and e. Detailed value ranges of α, β, and x for this and every (in-)feasible case are outlined in Section 4.2.3. Note that $\Theta_l(\alpha, \beta, b, e, x)$ is not explicitly determined in the course of the formulated dynamic program.

Fig. 4.10: Relation between Θ_l and $(\alpha, \beta, b, e, 1)_l$



4.2.3 The Recursion

In stage $l = 0$, we initialize our recursion with equations (4.1) and (4.2). Equation (4.2) contains only states that are not well-defined (e.g. $\alpha = 1$ indicates a job before the first maintenance, although there is no job to schedule in stage 0). Thus, these states have infinitely high costs. All well-defined states in Equation (4.1) receive the value 0, because no job is available to schedule in $l = 0$.

$$F_0(0, 0, b, e, 0) \;=\; 0 \qquad \forall\, b, e \in T, b \leq e \tag{4.1}$$

$$F_0(\alpha, \beta, b, e, x) \;=\; \infty \qquad \forall\, b, e \in T, b \leq e, \alpha, \beta \in \{1, \ldots, k\}, x \in \{0, 1\} \tag{4.2}$$

Value functions F_l for stage $l > 0$ are given in equations (4.3) to (4.13). If $b - p \leq r_l < e$, then job l is scheduled at time t within $[b, e)$ and F_l is the sum of the weighted completion time f_l of job l and the value functions F_{l-1} of the remaining interval. Otherwise, $F_l := F_{l-1}$, displayed in the lower terms of equations (4.4) and (4.5). Based on the recursion in Baptiste (2000), Equation (4.4) contains the objective function for $x = 0$.

We introduce auxiliary variables u and v, which replace α and/or β in state $l-1$ where required and are defined in the respective equation. Not well-defined states for $x = 0$ and $\alpha \neq \beta$ are formulated in Equation (4.3).

$$F_l(\alpha, \beta, b, e, 0) \;=\; \infty \qquad \forall\, b, e \in T, b \leq e, \alpha, \beta \in \{1, \ldots, k\}, \alpha \neq \beta \tag{4.3}$$

$$F_l(\alpha, \alpha, b, e, 0) = \begin{cases} \min\limits_{\substack{t_l \in T,\\ \max\{b, r_l\} \leq t_l \leq e-p,}} \; \min\limits_{u \in \{0, \ldots, \alpha-1\}} \Big(F_{l-1}(u, u, b, t_l, 0) \\ \qquad + f_l(t_l + p) \\ \qquad + F_{l-1}(\alpha - u - 1, \alpha - u - 1, t_l + p, e, 0) \Big), & \text{if } b - p \leq r_j < e \\ F_{l-1}(\alpha, \alpha, b, e, 0), & \text{else} \end{cases} \tag{4.4}$$
$$\forall\, b, e \in T, b \leq e, \alpha \in \{1, \ldots, k\}$$

$$F_l(\alpha, \beta, b, e, 1) = \begin{cases} \min\limits_{i \in \{1, \ldots, 7\}} \Big(F_l^i(\alpha, \beta, b, e, 1) \Big), & \text{if } b - p \leq r_j < e \\ F_{l-1}(\alpha, \beta, b, e, 1), & \text{else} \end{cases} \tag{4.5}$$
$$\forall\, b, e \in T, b \leq e, \alpha, \beta \in \{1, \ldots, k\}$$

In Equation (4.5), we use value function $F_l^i(\alpha, \beta, b, e, 1)$, which describes the different cases i of how the current state including at least one maintenance operation can be composed of previous states (see Section 4.2.2 and Figure 4.8). We distinguish between cases where job l is scheduled together with a maintenance operation or not, and whether and where a maintenance operation is already scheduled within the interval. Table 4.6 shows the assignment of the seven cases to equations (4.6) to (4.12) described below.

Tab. 4.6: Cases for value function $F_l^i(\alpha, \beta, b, e, 1)$

interval $[\mathbf{b}, \mathbf{t_l})$	**interval** $[\mathbf{t_l} + \mathbf{p}(+\mathbf{s}), \mathbf{e})$	**job l**	**i**	**Equation**
-	maintenance	-	1	(4.6)
maintenance	-	-	2	(4.7)
maintenance	maintenance	-	3	(4.8)
-	maintenance	maintenance	4	(4.9)
maintenance	-	maintenance	5	(4.10)
maintenance	maintenance	maintenance	6	(4.11)
-	-	maintenance	7	(4.12)

Equation (4.6) with $i = 1$ describes the case where no further maintenance is added in stage l and the newly scheduled job l is located before the first maintenance in interval $[b, e)$.

$$
\begin{aligned}
F_l^1(\alpha, \beta, b, e, 1) \;:=\; & \min_{u \in \{0,\ldots,\alpha-1\}} \; \min_{\substack{t_l \in T, \\ \max\{b, r_l\} \le t_l \le e-p}} \Big(F_{l-1}(u, u, b, t_l, 0) \\
& + f_l(t_l + p) + F_{l-1}(\alpha - u - 1, \beta, t_l + p, e, 1) \Big) \\
& \forall b, e \in T, b \le e, \alpha, \beta \in \{1, \ldots, k\}
\end{aligned} \tag{4.6}
$$

In Equation (4.7), job l is scheduled again without a maintenance in $i = 2$, and the maintenance operation assigned at the rear of state $l - 1$ is scheduled before job l within interval $[b, t_l)$.

$$
\begin{aligned}
F_l^2(\alpha, \beta, b, e, 1) \;:=\; & \min_{v \in \{0,\ldots,\beta-1\}} \; \min_{\substack{t_l \in T, \\ \max\{b, r_l\} \le t_l \le e-p}} \Big(F_{l-1}(\alpha, \beta - v - 1, b, t_l, 1) \\
& + f_l(t_l + p) + F_{l-1}(v, v, t_l + p, e, 0) \Big) \\
& \forall\; b, e \in T, b \le e, \alpha, \beta \in \{1, \ldots, k\}
\end{aligned} \tag{4.7}
$$

In Equation (4.8) with $i = 3$, job l is scheduled within interval $[b, e)$ with at least one maintenance located before and after job l, so that at least one job must already be scheduled within $[b, t_l)$ and $[t_l + p, e)$.

$$
\begin{aligned}
F_l^3(\alpha,\beta,b,e,1) \;:=\; & \min_{u,v\in\{0,\ldots,k-1\},u+v\le k-1} \; \min_{\substack{t_l\in T,\\ \max\{b,r_l\}\le t_l\le e-p}} \Big(F_{l-1}(\alpha,u,b,t_l,1) \\
& + f_l(t_l+p) + F_{l-1}(v,\beta,t_l+p,e,1)\Big) \\
& \forall\; b,e\in T, b\le e, \alpha,\beta\in\{1,\ldots,k\}
\end{aligned}
\tag{4.8}
$$

Cases $i=\{4,5,6\}$ in equations (4.9) to (4.11) display the situation in which a maintenance operation of length s is scheduled with job l in stage l. As mentioned above, the maintenance operation is always scheduled immediately after job l.

In case $i=4$ in Equation (4.9), job l and the new maintenance are scheduled before the first previously scheduled maintenance operation so that at least one job must already be assigned within $[t_l+p+s,e)$.

$$
\begin{aligned}
F_l^4(\alpha,\beta,b,e,1) \;:=\; & \min_{v\in\{0,\ldots,k-\alpha+1\}} \; \min_{\substack{t_l\in T,\\ \max\{b,r_l\}\le t_l\le e-p-s}} \Big(F_{l-1}(\alpha-1,\alpha-1,b,t_l,0) \\
& + f_l(t_l+p) + F_{l-1}(v,\beta,t_l+p+s,e,1)\Big) \\
& \forall\; b,e\in T, b\le e, \alpha,\beta\in\{1,\ldots,k\}
\end{aligned}
\tag{4.9}
$$

Corresponding to $i=4$ in Equation (4.10), job l and the new maintenance are scheduled after the last previous maintenance in interval b and e in case $i=5$.

$$
\begin{aligned}
F_l^5(\alpha,\beta,b,e,1) \;:=\; & \min_{u\in\{0,\ldots,k-\beta\}} \; \min_{\substack{t_l\in T,\\ \max\{b,r_l\}\le t_l\le e-p}} \Big(F_{l-1}(\alpha,u,b,t_l,1) \\
& + f_l(t_l+p) + F_{l-1}(\beta,\beta,t_l+p+s,e,0)\Big) \\
& \forall b,e\in T, b\le e, \alpha\in\{1,\ldots,k\}, \beta\in\{1,\ldots,k-1\}
\end{aligned}
\tag{4.10}
$$

In case $i=6$ in Equation (4.11), job l and the new maintenance are scheduled at t_l within $[b,e)$, with maintenance operations and at least one job per maintenance operation are located before t_l and after t_l+p+s.

$$
\begin{aligned}
F_l^6(\alpha,\beta,b,e,1) \;:=\; & \min_{u,v\in\{0,\ldots,k-1\},u+v\leq k} \; \min_{\substack{t_l\in T,\\ \max\{b,r_l\}\leq t_l\leq e-p-s}} \Big(F_{l-1}(\alpha,u,b,t_l,1) \\
& +f_l(t_l+p)+F_{l-1}(v,\beta,t_l+p+s,e,1)\Big) \\
& \forall\; b,e\in T, b\leq e, \alpha,\beta\in\{1,\ldots,k\}
\end{aligned}
\tag{4.11}
$$

Finally, $i=7$ in Equation (4.12) describes the case of scheduling job l and a maintenance operation in stage l with no previously scheduled maintenance operations within interval b and e.

$$
\begin{aligned}
F_l^7(\alpha,\beta,b,e,1) \;:=\; & \min_{\substack{t_l\in T,\\ \max\{b,r_l\}\leq t_l\leq e-p-s}} \Big(F_{l-1}(\alpha-1,\alpha-1,b,t_l,0) \\
& +f_l(t_l+p)+F_{l-1}(\beta,\beta,t_l+p+s,e,0)\Big) \\
& \forall\; b,e\in T, b\leq e, \alpha,\beta\in\{1,\ldots,k\}, \alpha+\beta-1\leq k
\end{aligned}
\tag{4.12}
$$

In stage n, job n and all jobs $\{1,\ldots,n-1\}$ within interval b and e are scheduled with $b=\min\{T\}$ and $e=\max\{T\}$ in consideration of the constraints for $x=0$ and $x=1$. Value function F^* in Equation (4.13) shows the optimal total weighted completion time in stage n for the corresponding state.

$$
\begin{aligned}
F^* \;:=\; \min_{\alpha,\beta\in\{1,\ldots,k\}} \; & \Big(F_n(\alpha,\alpha,\min\{T\},\max\{T\},0), \\
& \min_{i\in\{1,\ldots,7\}} F_n^i(\alpha,\beta,\min\{T\}),\max\{T\},1\Big)
\end{aligned}
\tag{4.13}
$$

4.2.4 Properties of the Dynamic Program

In the following, we focus on the rumtime complexity of the generated approach, illustrate why this method leads to optimal schedules, and finally consider the possibility of transferring this algorithm to other single machine layouts.

The runtime complexity of the formulated dynamic program implies the number of feasible states $(\alpha, \beta, b, e, x)_l$ and the runtime to find the shortest path in the graph of the algorithm. The number of states is bounded by $l \in \{0, \ldots, n\}$, $\alpha \in \{0, k+1\}$, $\beta \in \{0, \ldots, k\}$, and $b, e \in T$. Since $|T|$ is bounded by n^3 (n release dates r_j, $n+1$ possible time slots, maximal $n-1$ maintenance operations), and k is bounded by n, we have at most $\mathcal{O}(n^9)$ states. We need $\mathcal{O}(m^2)$ with $m = \mathcal{O}(n^9)$ to find the shortest path in the graph among these states, so that we have a total runtime of $\mathcal{O}(n^{18})$.

Next, we show why $F_l(\alpha, \beta, b, e, x)$ is the minimum value function of state $(\alpha, \beta, b, e, x)_l$ by applying the findings of Bellman (1957) on the structure of optimal terms in dynamic programs. For this purpose, we examine why a function value in stage l is an optimal solution for the respective state when job l is scheduled, and why this is also true for function values of stage $l-1$. In the previous section, we have defined

$$F_l(\alpha, \beta, b, e, x) = \min_{\substack{t_l \in T, \\ \max\{b, r_l\} \leq t_l \leq e-p}} \{F_{l-1}(\alpha', \beta', b, t_l, x') + F_{l-1}(\alpha'', \beta'', t_l + p, e, x'') + f_l(t_l + p)\}$$

if $r_l \in [b-p, e)$. Else, F_l is equal to F_{l-1}. $\alpha, \beta, \alpha', \beta', \alpha'', \beta'', x, x'$ and x'' are defined according to the equations in Section 4.2.3. The following explanations apply equally to $x = 0$ and $x = 1$. If job l and a maintenance operation are assigned in state l, the length of a maintenance operation s must be considered in the following descriptions of subintervals.

We assume that $r_l \in [b-p, e)$ and a point in time $t_l \in T$ are given with $t_l \in [\max\{b, r_l\}, e-p]$ to schedule job l in state l. We have feasible schedules Π_1 and Π_2 with value functions $F_{l-1}(\alpha', \beta', b, t_l, x')$ and $F_{l-1}(\alpha'', \beta'', t_l + p, e, x'')$. As a result, job l, jobs $\Theta_{l-1}(\alpha', \beta', b, t_l, x')$ and $\Theta_{l-1}(\alpha'', \beta'', t_l + p, e, x'')$ of Π_1 and Π_2 are scheduled in state l. Since state $(\alpha, \beta, b, e, x)_l$ is one with a feasible schedule, it holds for every job j in state l, and consequently also for every job in Π_1 and Π_2, that every job has a feasible start time, i.e. $r_j \leq C_j - p$. If a feasible schedule contains only jobs j with $r_j \in [b-p, e)$, this

also holds for jobs in subintervals of the schedule in their appropriate intervals. Hence, the same holds for Π_1 and Π_2 if state l contains a feasible schedule. In state l, we search for a feasible schedule with the minimum value function F_l for state $(\alpha, \beta, b, e, x)_l$. If F_l is a minimum value function for all scheduled jobs in interval $[b, e]$, then any value function F_{l-1} must also be a minimum function for state $l-1$ (see the Bellman equation from Bellman (1957)). For state $(\alpha, \beta, b, e, x)_n$, $b = \min\{T\}$ and $e = \max\{T\}$ are given, so that $r_j \in [b-p, e)$ must hold for every job $j \in \{1, \ldots, n\}$. As a result, all n jobs are scheduled in the final schedule, which is a feasible schedule. According to the previous descriptions, F_n describes the minimum value function for state n.

Apart from this, Baptiste (2000) illustrates that a dynamic programming algorithm solving $1|r_j, p_j = p|\sum w_j C_j$ can also be applied to minimize total tardiness $\sum T_j$. We use this fact to formulate the following theorem.

Theorem 4.2.1. *The problem* $1|pd, r_j, p_j = p|\sum T_j$ *is solvable in* polynomial time *by using the dynamic program formulated in Section 4.2.3. Further, the problems of minimizing makespan* $1|pd, r_j, p_j = p|C_{max}$*, total completion time* $1|pd, r_j, p_j = p|\sum C_j$*, and maximum lateness* $1|pd, r_j, p_j = p|L_{max}$ *are solvable in* polynomial time.

Baptiste (2000) shows that the objective functions total weighted completion time $\sum w_j C_j$ and total tardiness $\sum T_j$ are cases of sum-based cost functions that follow the same principles regarding their optimization for $p_j = p$. Both objectives are characterized by function f_j that is non-decreasing with time points t_1 and t_2 with $t_2 > t_1$ and $f_j(t_1) \leq f_j(t_2)$. In addition, they are monotone with $(f_{j_1} - f_{j_2})t_1 \leq (f_{j_1} - f_{j_2})t_2$ or $(f_{j_1} - f_{j_2})t_1 \geq (f_{j_1} - f_{j_2})t_2$ for any pair of jobs j_1 and j_2. As a result, without loss of generality, it holds equality considering the runtime complexity of the objective functions $\sum w_j C_j$ and $\sum T_j$. These characteristics of $\sum w_j C_j$ and $\sum T_j$ remain applicable for the case $p_j = p$ subject to *pd* maintenance so we can conclude that $1|pd, r_j, p_j = p|\sum T_j$ is solvable in polynomial time by applying the dynamic program formulated in Section 4.2.3. Using the findings of the complexity hierarchy of objective functions (see, e.g., Pinedo (2022)), it can be additionally concluded, without loss of generality, that the optimization of total completion time $\sum C_j$, maximum lateness L_{max}, and makespan C_{max} are polynomially solvable for the single machine layout with release dates, uniform processing

times, and *pd* maintenance, since it holds for the problems $1|pd, r_j, p_j = p|\sum w_j C_j$ and $1|pd, r_j, p_j = p|\sum T_j$.

4.2.5 Numerical Example

We introduce a small instance with $n = 3$ jobs, processing time $p_j = p = 1$, release dates r_j, and weights w_j (see Table 4.7) to demonstrate the approach of the dynamic program to determining the minimum total weighted completion time. Maintenance operations take $s = 1$ time units on the machine and must be performed after at most $k = 2$ jobs.

Tab. 4.7: Example instance for $1|pd, r_j, p_j = p|\sum w_j C_j$

job j	**1**	**2**	**3**
$\mathbf{r_j}$	2	0	1
$\mathbf{w_j}$	5	2	1
$p = 1,\ k = 2,\ s = 1$			

First, set $T = \{0, 1, 2, 3, 4, 5, 6, 7\}$ is determined including all r_j related start and completion times t of all n jobs (see explanations in Section 4.2.1). The jobs are already sorted in non-increasing order according to w_j, so we obtain job sequence $(1, 2, 3)$ to select them in stages 1 to n.

In the first step of the dynamic program, F_l is generated for each state in stage $l = 0$ with $\alpha, \beta, x = 0$ and $b, e \in T$ with $b \leq e$ as shown exemplary in Table 4.8. The value function F_0 of each regular state is equal to zero.

Tab. 4.8: Dynamic program: value functions in stage $l = 0$

$F_0(\alpha, \beta, b, e, x)$		
$F_0(0, 0, 0, 0, 0)$	$=$	0
$F_0(0, 0, 0, 1, 0)$	$=$	0
$F_0(0, 0, 0, 2, 0)$	$=$	0
$\dots$		
$F_0(0, 0, 6, 7, 0)$	$=$	0
$F_0(0, 0, 7, 7, 0)$	$=$	0

The next step is to schedule job 1 in stage $l = 1$ in any state at time t_1 with $t_1 \in [\max\{b, r_1\}, e - p]$ if $r_1 \in [b - p, e)$. If $r_l \notin [b - p, e)$, $F_l := F_{l-1}$. Table 4.9 shows sample data for function values with the combinations of α, β, b, e, and x. For instance, r_1 is not within $[-1, 0)$ in interval $[0, 2)$ so job 1 is not scheduled within this interval and the corresponding state equals the state of stage 0, i.e., $F_1(0, 0, 0, 2, 0) := F_0(0, 0, 0, 2, 0)$. In state $(1, 1, 3, 3, 0)_1$, $r_1 \in [b-p, e)$ and job 1 is scheduled at $t_1 = 3$. However, $F_1(1, 1, 3, 3, 0)$ is equal to infinity because state $(0, 0, 4, 3, 0)_0$ violates the restriction that $b \leq e$ must hold for every combination of b and e in every state $(\alpha, \beta, b, e, x)_l$. Following this scheme, each function value of stage 1 is determined.

Tab. 4.9: Dynamic program: value functions in stage $l = 1$

$r_1 = 2,\ t_1 \in \{2, 3, 4, 5, 6\}$					
$r_1 \notin [-1, 0)$	$F_1(0, 0, 0, 0, 0)$	:=	$F_0(0, 0, 0, 0, 0)$	=	0
$r_1 \notin [-1, 1)$	$F_1(0, 0, 0, 1, 0)$	:=	$F_0(0, 0, 0, 1, 0)$	=	0
$r_1 \notin [-1, 2)$	$F_1(0, 0, 0, 2, 0)$	:=	$F_0(0, 0, 0, 2, 0)$	=	0
$r_1 \in [-1, 3)$	$F_1(1, 1, 0, 3, 0)$	:=	$F_0(0, 0, 0, 2, 0) + F_0(0, 0, 3, 3, 0) + f_1(2)$	=	15
$r_1 \in [-1, 4)$	$F_1(1, 1, 0, 4, 0)$	:=	$F_0(0, 0, 0, 2, 0) + F_0(0, 0, 3, 4, 0) + f_1(2)$	=	15
...					
$r_1 \notin [-1, 2)$	$F_1(0, 0, 1, 2, 0)$	:=	$F_0(0, 0, 1, 2, 0)$	=	0
$r_1 \in [-1, 3)$	$F_1(1, 1, 1, 3, 0)$	:=	$F_0(0, 0, 1, 1, 0) + F_0(0, 0, 3, 3, 0) + f_1(1)$	=	15
...					
$r_1 \in [2, 3)$	$F_1(1, 1, 3, 3, 0)$	:=	$F_0(0, 0, 3, 3, 0) + F_0(0, 0, 4, 3, 0) + f_1(3)$	=	∞
$r_1 \in [2, 4)$	$F_1(0, 0, 3, 4, 0)$	:=	$F_0(0, 0, 3, 3, 0) + F_0(0, 0, 4, 4, 0) + f_1(3)$	=	20
...					
$r_1 \notin [4, 6)$	$F_1(0, 0, 7, 7, 0)$	:=	$F_0(0, 0, 7, 7, 0)$	=	0

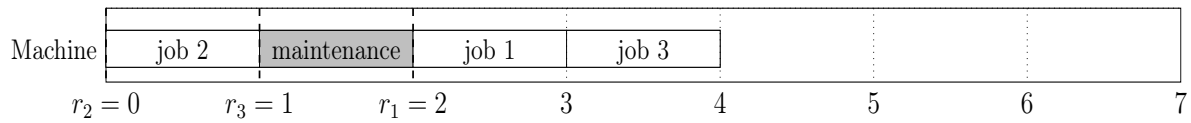
After completing the generation of the function values in stage 1, job 2 is scheduled in stage $l = 2$ at time t_2 with $t_2 \in [\max\{b, r_2\}, e - p]$ if $r_2 \in [b - p, e)$ (see sample data in Table 4.10). The scheme described for stage 1 remains the same for stage 2 and is applied to determine the function values F_2.

Tab. 4.10: Dynamic program: value functions in stage $l = 2$

$r_2 = 0,\ t_2 \in \{0,1,2,3,4,5,6\}$					
$r_2 \notin [-1,0)$	$F_2(0,0,0,0,0)$	:=	$F_1(0,0,0,0,0)$	=	0
...					
$r_2 \in [-1,2)$	$F_2(1,1,0,2,0)$	:=	$F_1(0,0,0,0,0) + F_1(0,0,1,2,0) + f_2(0)$	=	2
$r_2 \in [-1,3)$	$F_2(2,2,0,3,0)$	:=	$F_1(0,0,0,0,0) + F_1(1,1,1,3,0) + f_2(0)$	=	17
$r_2 \in [-1,3)$	$F_2(1,1,0,3,1)$	:=	$F_1(0,0,0,0,0) + F_1(1,1,2,3,0) + f_2(0)$	=	17
...					
$r_2 \in [0,1)$	$F_2(1,1,1,1,0)$	:=	$F_1(0,0,1,1,0) + F_1(0,0,2,1,0) + f_2(1)$	=	∞
$r_2 \in [0,2)$	$F_2(1,1,1,2,0)$	:=	$F_1(0,0,1,1,0) + F_1(0,0,2,2,0) + f_2(1)$	=	4
$r_2 \in [0,3)$	$F_2(2,2,1,3,0)$	:=	$F_1(0,0,1,1,0) + F_1(1,1,2,3,0) + f_2(1)$	=	19
$r_2 \in [0,4)$	$F_2(2,2,1,4,0)$	:=	$F_1(0,0,1,1,0) + F_1(1,1,2,4,0) + f_2(1)$	=	19
$r_2 \in [0,4)$	$F_2(1,1,1,4,1)$	:=	$F_1(0,0,1,1,0) + F_1(1,1,3,4,0) + f_2(1)$	=	24
...					
$r_2 \notin [1,2)$	$F_2(0,0,2,2,0)$	:=	$F_1(0,0,2,2,0)$	=	0
$r_2 \notin [1,3)$	$F_2(1,1,2,3,0)$	:=	$F_1(1,1,2,3,0)$	=	15
...					
$r_2 \notin [6,7)$	$F_2(0,0,7,7,0)$	:=	$F_1(0,0,7,7,0)$	=	0

In stage $l = n = 3$, which is the last stage of the dynamic program for this instance, only the maximum interval with $b = \min\{T\} = 0$ and $e = \max\{T\} = 7$ is considered. For each feasible combination of α, β, and x, states $(\alpha, \beta, 0, 7, x)_3$ with their value functions F_3 are generated (see again exemplary data in Table 4.11). As a result, total weighted completion time F^* in stage $n = 3$ is the minimum of the values of $F_3(\alpha, \beta, 0, 7, x)$, here $F^* = 21$. The associated Gantt chart for the final schedule is shown in Figure 4.11.

Fig. 4.11: Gantt chart with final schedule for example instance



Tab. 4.11: Dynamic program: value functions in final stage $l = 3$

$r_3 = 1,\ t_3 \in \{1,2,3,4,5,6\}$
$r_3 \in [-1,7),\ F_3(2,1,0,7,1) = \min\{$
$F_2(0,0,0,1,0) + F_2(1,1,2,7,1) + f_3(1) = \infty,$
$F_2(1,1,0,1,0) + F_2(1,1,3,7,0) + f_3(1) = 24,$
$\dots$
$F_2(2,2,0,3,0) + F_2(0,0,5,7,0) + f_3(4) = 22,$
$F_2(2,2,0,4,0) + F_2(0,0,6,7,0) + f_3(5) = 23,$
$F_2(2,2,0,5,0) + F_2(0,0,7,7,0) + f_3(6) = 24$
$\} = 22$
$r_3 \in [-1,7),\ F_3(1,2,0,7,1) = \min\{$
$F_2(0,0,0,1,0) + F_2(2,2,3,7,0) + f_3(1) = \infty,$
$\dots$
$F_2(1,1,0,3,1) + F_2(0,0,4,7,1) + f_3(3) = 21,$
$F_2(1,1,0,4,1) + F_2(0,0,5,7,1) + f_3(4) = 22,$
$\dots$
$\} = 21$
$r_3 \in [-1,7),\ F_3(1,1,0,7,1) = \min\{$
$F_2(0,0,0,1,0) + F_2(1,1,3,7,1) + f_3(1) = \infty,$
$\dots$
$F_2(1,1,0,4,1) + F_2(0,0,6,7,0) + f_3(5) = 23,$
$F_2(1,1,0,4,1) + F_2(0,0,7,7,0) + f_3(6) = 24$
$\} = 23$
$\Rightarrow\ \mathbf{F}^* := \min\{\mathbf{22},\mathbf{21},\mathbf{23}\} = \mathbf{21}$

4.2.6 Conclusion

In this section, we study single machine scheduling problems with position-dependent (*pd*) maintenance operations, as they can be found in any real-world case, where the number of jobs primarily influences the wear and tear of machines rather than of time-dependent factors such as job processing times or the machine's runtime. We focus on single machine layouts with release dates and uniform processing times for any number of given jobs to minimize total weighted completion time, expressed by $1|pd, r_j, p_j = p|\sum w_j C_j$ using Graham's three field notation. A dynamic program is formulated to solve this problem in polynomial time with runtime $\mathcal{O}(n^{18})$. This dynamic program can also be applied to minimize total tardiness, i.e., $1|pd, r_j, p_j = p|\sum T_j$. In addition, related single machine scheduling problems with *pd* maintenance are examined regarding their

runtime complexity. Table 4.12 shows these results together with the research conducted in particular by Drozdowski et al. (2017).

Tab. 4.12: Complexity results for single machine layouts including position-dependent maintenance (pd)

objective \ layout	$\mathbf{1\|pd\|\gamma}$	$\mathbf{1\|pd, r_j\|\gamma}$	$\mathbf{1\|pd, pmtn\|\gamma}$	$\mathbf{1\|pd, r_j, pmtn\|\gamma}$	$\mathbf{1\|pd, r_j, p_j = p\|\gamma}$
$\mathbf{C_{max}}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}^x$
$\mathbf{L_{max}}$	$\mathcal{P}$	$\mathcal{NP}$-hard*	$\mathcal{P}$	$\mathcal{NP}$-hard*	$\mathcal{P}^x$
$\sum \mathbf{C_j}$	$\mathcal{P}^x$	$\mathcal{NP}$-*hard*^{*x}	$\mathcal{P}^x$	$\mathcal{NP}$-hard	$\mathcal{P}^x$
$\sum \mathbf{w_j C_j}$	$\mathcal{P}^x$	$\mathcal{NP}$-*hard*^{*x}	$\mathcal{P}^x$	$\mathcal{NP}$-*hard*^{*x}	$\mathcal{P}^x$
$\sum \mathbf{T_j}$	$\mathcal{NP}$-*hard*^{x}	$\mathcal{NP}$-hard	$\mathcal{NP}$-*hard*^{x}	$\mathcal{NP}$-hard	$\mathcal{P}^x$
$\sum \mathbf{w_j T_j}$	$\mathcal{NP}$-*hard*^{x}	$\mathcal{NP}$-hard	$\mathcal{NP}$-*hard*^{x}	$\mathcal{NP}$-hard	

*in the strong sense

^{x} *studied in this work*

Although numerous single machine layouts with *pd* maintenance are examined, there are still some blind spots. One of these layouts that is still open regarding its runtime is $1|pd, r_j, p_j = p|\sum w_j T_j$ (i.e., minimizing the total weighted tardiness for the layout studied in this work). In fact, the runtime complexity of the problem, including continuous machine availability, is also still open, so this layout is definitely well worth considering. In addition, layouts that deal with job preemption, release dates, and $p_j = p$ may be especially interesting in terms of their complexity. For example, we know that the problem $1|r_j, pmtn, p_j = p|\sum T_j$, the same layout but without any type of machine availability constraint, and as a consequence minimizing maximum lateness $1|r_j, pmtn, p_j = p|L_{max}$ are solvable in polynomial time (see Tian et al. (2006) for minimizing $\sum T_j$). However, their proof cannot be translated to the case of *pd* maintenance. In their work dealing with $1|r_j, pmtn, p_j = p|\sum T_j$, Tian et al. (2006) construct a non-delay schedule that uses as many job preemptions as necessary to avoid idle time in the schedule. If this method were applied to our case, the number of maintenance operations would potentially increase significantly because each task as part of a job counts equally for machine deterioration and for k accordingly. Furthermore, Drozdowski et al. (2017) show via 3-partition that solving $1|pd, r_j, pmtn|L_{max}$ is $\mathcal{NP}$-hard in the strong sense. This proof does not hold for $p_j = p$ either, but could be a basis for dealing with the layout including uniform processing times.

Moreover, efficient heuristics should be generated for the problems with high runtime and for the $\mathcal{NP}$-hard problems in order to enable reliable high-quality solutions for real-world applications. Apart from single machine layouts, the whole field of shop floors such as flow shops or job shops including *pd* maintenance scheduling has not yet been studied.

4.3 Minimizing the Number of Tardy Jobs with Position-Dependent Maintenance, Release Dates, and Uniform Processing Times

In this section, which is from Hipp (2024), we consider the single machine scheduling problem of minimizing the number of late jobs, taking into account release dates, due dates, and uniform processing times of jobs. In addition, limited machine availability in the form of position-dependent maintenance operations is assumed. The characteristics of this problem are analyzed and an algorithm for solving the above single machine problem in polynomial time is formulated.

The remainder of this section is organized as follows. In Section 4.3.1, we describe the studied problem, before the exact solution approach is presented in Section 4.3.2. After demonstrating the dynamic program with an example instance, the section is summarized in Section 4.3.4 including an outlook.

4.3.1 Problem Description

In this study, we are given a set of $n \in \mathbb{N}$ jobs to schedule on a single machine. Each job $j \in \{1, \ldots, n\}$ is defined by its processing time p_j, which is identical for the whole set of jobs, so we have $p_j = p$ where p is a positive integer. Besides, each job has a release date $r_j \in \mathbb{N}_0$ and due date $d_j \in \mathbb{N}$. Without loss of generality, we assume $d_j \leq d_{j+1}$ for all $j \in \{1, \ldots, n-1\}$. To characterize maintenance operations, the positive integers $k \in \mathbb{N}$ and $s \in \mathbb{N}$ are given, where k defines the maximum number of jobs after which idle time must be scheduled on the machine to complete a maintenance operation of s time units. The problem is to schedule the n jobs on the machine by determining the completion time C_j of each job with $C_j \geq r_j + p$. A feasible schedule is found if no two jobs j and j' overlap

with $j \neq j'$ and $j, j' \in \{1, \ldots, n\}$ ($|C_j - C_{j'}| \geq p$) and no job is scheduled before its release date r_j ($C_j - p \geq r_j$). After scheduling at most k jobs, idle time on the machine must be blocked to process a maintenance operation of s time units. Among all feasible schedules, we look for one that minimizes the number of late jobs $\sum_{j=1}^{n} U_j$. U_j of job j is defined as a binary variable that is $= 1$ if $C_j - d_j > 0$, otherwise $= 0$. Using Graham's three-field notation Graham et al. (1979), this problem can be described as $1|pd, p_j = p, r_j| \sum U_j$ (or $1|\left([0, \overline{s}], \omega, (n, f), p_j\right), p_j = p, r_j| \sum U_j$ according to Section 2.2).

4.3.2 Exact Solution Algorithm

The algorithm of this section is based on the dynamic program of Baptiste (1999), improved by Chrobak et al. (2004), which solves this layout without *pd* maintenance. Instead of minimizing the number of late jobs, we maximize the number of jobs scheduled on time in an interval and then determine the number of late jobs in a second step. First, we generate set T of all possible times points at which jobs can start or end. By assuming a left-shifted schedule, idle times on the machine occur only when induced by release date r_j. That is, T depends on r_j, the number of processed jobs $j' \in \{0, \ldots, n\}$, and maintenance operations $m \in \{0, \ldots, n-1\}$, so we get

$$T := \{t : \exists r_j, \exists j' \in \{0, \ldots, n\}, \exists m \in \{0, \ldots, n-1\}, m \leq j', t = r_j + j' \cdot p + m \cdot s\}.$$

In addition to the already mentioned parameters to characterize jobs and maintenance operations, α (number of jobs before the first maintenance, $\alpha \leq k$), β (number of jobs after the last maintenance, $\beta \leq k$), and x (binary variable $= 1$ if at least one maintenance operation is scheduled within an interval) are introduced to identify the current state of scheduled maintenance operations and their position in the sequence of a considered interval.

The algorithm contains n stages in which it is considered whether a job can be scheduled on time. $C(l, b, u, \alpha, \beta, x)$ denotes the minimum end of an interval in stage $l \in \{1, \ldots, n\}$ starting at time $b \in T$ to schedule $u \leq l$ jobs on time. It must hold $b - p \leq r_j$ for each job $j \in \{1, \ldots, u\}$. Following applies to the properties of $C(l, b, u, \alpha, \beta, x)$ and the corresponding interval:

if $x = 0$,
- no maintenance operation is scheduled in the interval from times b to $C(l, b, u, \alpha, \beta, 0)$
- $\alpha = \beta = u$ jobs are scheduled within this interval

if $x = 1$,
- at least one maintenance operation is scheduled within b and $C(l, b, u, \alpha, \beta, 1)$
- α jobs are scheduled in the subinterval of b and the start of the first maintenance operation and β jobs between the last maintenance of the interval and $C(l, b, u, \alpha, \beta, 1)$

We seek an interval starting at b and ending at $C(l, b, u, \alpha, \beta, x)$ to schedule u jobs on time. In stage l, job l (and a maintenance operation right behind job l) is checked to be scheduled in any possible interval starting at b. Jobs in an interval that have been scheduled in previous stages are described via v and w. $v \in \{0, \ldots, u-1\}$ indicates the number of on-time scheduled jobs between b and job l, and $w = u - v - 1$ is defined as the number of scheduled jobs between job l and the end of the interval. If $r_l < b - p$, job l is not considered to be scheduled and thus $C(l, b, u, \alpha, \beta, x) := C(l-1, b, u, \alpha, \beta, x)$. Else, start time t_l of job l is determined by $t_l = max\{r_l, C(l-1, b, v, \alpha', \beta', x')\}$. $C(l-1, b, v, \alpha', \beta', x')$ is the completion time for scheduling v jobs between b and job l. If job l can be scheduled on time ($t_l + p \leq d_l$), then the end of the interval starting at b is defined by $C(l, b, u, \alpha, \beta, x) := min\{C(l-1, b, u, \alpha, \beta, x), C(l-1, t_l+p, w, \alpha'', \beta'', x'')\}$ (the specifics of α, β, and x are given in the next paragraph). This means that the function value is the minimum of the function value in the previous stage $l-1$ and the end of the interval after scheduling the remaining w jobs following job l. In stage $l = n$, only intervals starting at $b = min\{T\}$ are considered. Among all completion times less than infinity, summarized in set C^*, we seek the maximum number of on-time scheduled jobs u. Finally, the number of late jobs $\sum_{j=1}^{n} U_j$ is determined by $\sum_{j=1}^{n} U_j := n - \max_{C^*}\{u\}$.

Before starting the procedure in stage $l = 1$, every feasible start and completion time of all jobs is generated with Equation (4.14). Equation (4.15) contains only not well-defined states for $x = 0$ with $u \neq \alpha$ or $\alpha \neq \beta$ jobs in stage l.

$$C(l, b, 0, 0, 0, 0) = b \quad \forall\, l \in \{0, \ldots, n\}, b \in T \tag{4.14}$$

$$C(l,b,u,\alpha,\beta,0) = \infty \tag{4.15}$$
$$\forall\ u \neq \alpha \vee \alpha \neq \beta, l \in \{1,\ldots,n\}, b \in T, u \in \{1,\ldots,l\}, \alpha,\beta \in \{1,\ldots,k\}$$

If $r_l < b - p$ in stage l, Equation (4.16) applies and the function value of stage $l-1$ corresponds to the one of stage l.

$$C(l,b,u,\alpha,\beta,x) := C(l-1,b,u,\alpha,\beta,x) \tag{4.16}$$
$$\forall\ r_l < b, l \in \{1,\ldots,n\}, b \in T, u \in \{1,\ldots,l\}, \alpha,\beta \in \{1,\ldots,k\}, x \in \{0,1\}$$

Else ($r_l \geq b-p$), the function values for stage l are given in equations (4.17) and (4.18). Job l is tested to be scheduled within b and $C(l,b,u,\alpha,\beta,x)$ in stage l with $u = v+w+1$ on-time assigned jobs. Equation (4.17) contains the function value when no maintenance operation is scheduled within the interval ($x = 0$), and Equation (4.18) treats the respective case for $x = 1$. In Equation (4.18), the feasible cases of scheduling job l with or without a maintenance operation as well as the position of maintenance operations in stage $l-1$ are referred to by case $i \in \{1,\ldots,7\}$. Table 4.13 outlines these cases, formulated in equations (4.21) to (4.27).

$$C(l,b,u,\alpha,\beta,0) := \min\big\{C(l-1,b,u,\alpha,\beta,0), C(l-1,t_l+p,w,w,w,0)\big\} \tag{4.17}$$
$$\forall\ r_l \geq b, t_l = \max_{v\in\{0,\ldots,u-1\}}\{r_l, C(l-1,b,v,v,v,0)\}, t_l + p \leq d_l,$$
$$l \in \{1,\ldots,n\}, b \in T, u \in \{1,\ldots,l\}, w = u - v - 1, \alpha,\beta \in \{1,\ldots,k\}$$

$$C(l,b,u,\alpha,\beta,1) := \min\big\{C(l-1,b,u,\alpha,\beta,1), \min_{i\in\{1,\ldots,7\}}\{C^i(l,b,u,\alpha,\beta,1)\}\big\} \tag{4.18}$$
$$\forall\ r_l \geq b, l \in \{1,\ldots,n\}, b \in T, u \in \{1,\ldots,l\}, \alpha,\beta \in \{1,\ldots,k\}$$

In stage n, C^* is defined in Equation (4.19) as the set of function values less than infinite of intervals starting at $b = \min\{T\}$, i.e., $C(n,\min\{T\},u,\alpha,\beta,x) < \infty$.

From this set, the maximum number of jobs scheduled at a given time is determined by $\max_{C^*}\{u\}$. Using Equation (4.20) we get the overall objective value – the number of late jobs.

$$C^* = \left\{C(n, \min\{T\}, u, \alpha, \beta, 0), \min_{i\in\{1,\dots,7\}}\{C^i(n, \min\{T\}, u, \alpha, \beta, 1)\}\right\} \tag{4.19}$$
$$\forall\ C(n, \min\{T\}, u, \alpha, \beta, x) < \infty, x \in \{0,1\}, u \in \{1,\dots,n\}, \alpha, \beta \in \{1,\dots,k\}$$

$$\sum_{j=1}^{n} U_j = n - \max_{C^*}\{u\} \tag{4.20}$$

Tab. 4.13: Cases for function value $C^i(l, b, u, \alpha, \beta, 1)$

interval $[\mathbf{b}, \mathbf{t_l}]$	**interval $[\mathbf{t_l}(+\mathbf{s}), \mathbf{C}]$**	**job l**	**i**	**Equation**
-	-	maintenance	1	(4.21)
maintenance	-	-	2	(4.22)
-	maintenance	-	3	(4.23)
maintenance	maintenance	-	4	(4.24)
maintenance	-	maintenance	5	(4.25)
-	maintenance	maintenance	6	(4.26)
maintenance	maintenance	maintenance	7	(4.27)

Cases $i \in \{1,\dots,7\}$ that apply in Equation (4.18) for $x = 1$ are formulated in equations (4.21) to (4.27) to show feasible options for assigning job l in stage l with at least one maintenance operation in the corresponding interval before, after, or scheduled together with job l. Following definitions hold for all seven cases if not redefined in the respective equation: $l \in \{1,\dots,n\}, b \in T, u \in \{1,\dots,l\}, \alpha, \beta \in \{1,\dots,k\}$. Table 4.13 gives an overview of where maintenance operations are assigned in each case. α' and β' are introduced as auxiliary variables to replace α or β when necessary.

$$C^1(l, b, u, \alpha, \beta, 1) := C(l-1, t_l + p + s, \beta, \beta, \beta, 0) \tag{4.21}$$
$$\forall\ t_l = \max\{r_l, C(l-1, b, \alpha-1, \alpha-1, \alpha-1, 0)\}, t_l + p \le d_l, \alpha + \beta = u$$

$$C^2(l,b,u,\alpha,\beta,1) := C(l-1,t_l+p,w,w,w,0) \tag{4.22}$$
$$\forall\ t_l = \max_{v\in\{0,\ldots,u-1\}}\{r_l, C(l-1,b,v,\alpha,\beta-w-1,1)\},$$
$$t_l+p\le d_l, w=u-v-1, w<\beta, \alpha+\beta\le u$$

$$C^3(l,b,u,\alpha,\beta,1) := C(l-1,t_l+p,w,\alpha-v-1,\beta,1) \tag{4.23}$$
$$\forall\ t_l = \max_{v\in\{0,\ldots,\alpha-1\}}\{r_l, C(l-1,b,v,v,v,0)\},$$
$$t_l+p\le d_l, w=u-v-1, \alpha+\beta\le u$$

$$C^4(l,b,u,\alpha,\beta,1) := C(l-1,t_l+p,w,\alpha',\beta,1) \tag{4.24}$$
$$\forall\ t_l = \max_{v\in\{0,\ldots,u-2\}}\{r_l, C(l-1,b,v,\alpha,\beta',1)\}, t_l+p\le d_l,$$
$$w=u-v-1, \alpha',\beta'\in\{0,\ldots,k-1\}, \alpha'+\beta'<k, \alpha+\beta\le u$$

$$C^5(l,b,u,\alpha,\beta,1) := C(l-1,t_l+p+s,\beta,\beta,\beta,0) \tag{4.25}$$
$$\forall\ t_l = \max_{v\in\{0,\ldots,u-1\}}\{r_l, C(l-1,b,v,\alpha,\beta',1)\}, t_l+p\le d_l,$$
$$\beta'\in\{0,\ldots,k-\beta\}, \beta=u-v-1, \alpha+\beta\le u$$

$$C^6(l,b,u,\alpha,\beta,1) := C(l-1,t_l+p+s,w,\alpha',\beta,1) \tag{4.26}$$
$$\forall\ t_l = \max\{r_l, C(l-1,b,\alpha-1,\alpha-1,\alpha-1,0)\}, t_l+p\le d_l,$$
$$w=u-\alpha-2, \alpha'\in\{1,\ldots,k-\alpha+1\}, \alpha+\beta\le u$$

$$C^7(l,b,u,\alpha,\beta,1) := C(l-1,t_l+p+s,w,\alpha',\beta,1) \tag{4.27}$$
$$\forall\ t_l = \max_{v\in\{1,\ldots,u-1\}}\{r_l, C(l-1,b,v,\alpha,\beta',1)\}, t_l+p\le d_l,$$
$$w=u-v-1, w\ge 1, \alpha',\beta'\in\{0,\ldots,k-1\}, \alpha'+\beta'\le k, \alpha+\beta\le u$$

The runtime complexity of function value $C(l, b, u, \alpha, \beta, x)$ is $\mathcal{O}(n^7)$ with l, u, α, and β each bounded by n. b is defined by the number of feasible starting times $|T|$, which is bounded by n^3 due to release date r_j, $n+1$ possible time slots, and maximal $n-1$ maintenance operations. Chrobak et al. (2004) state that each function value of the original algorithm is computed in linear time. Since this also applies to our approach, the entire problem is solvable in $\mathcal{O}(n^8)$.

4.3.3 Numerical Example

To illustrate the approach of the dynamic program, we solve a small instance with $n = 2$ jobs, processing times $p_j = p = 2$, release dates r_j, and due dates d_j, summarized in Table 4.14. Maintenance operations take $s = 2$ time units on the machine and must be performed after $k = 1$ job.

Tab. 4.14: Example instance for $1|pd, r_j, p_j = p| \sum U_j$

job j	**1**	**2**
$\mathbf{r_j}$	0	2
$\mathbf{d_j}$	2	5
$p = 2,\ k = 1,\ s = 2$		

Before starting the algorithm, set $T = \{0, 2, 4, 6, 8\}$ is determined including the feasible start and completion times of jobs. In the initialization, displayed in Table 4.15, these time points are generated to be called in the stages with $C(l, b, 0, 0, 0, 0)$. In the dynamic program, we have $n = 2$ stages, $b \in T$, α and β are bounded by $k = 1$, and $u \leq 2$. Since the jobs are already sorted in non-decreasing order regarding d_j, job sequence $(1, 2)$ holds to select one jobs each in stages 1 and 2.

Tab. 4.15: Dynamic program: value functions in initialization

$C(l,b,0,0,0,0)$	$=$	b	$C(l,b,0,0,0,0)$	$=$	b
$C(0,0,0,0,0,0)$	$=$	0	$C(1,6,0,0,0,0)$	$=$	6
$C(0,2,0,0,0,0)$	$=$	2	$C(1,8,0,0,0,0)$	$=$	8
$C(0,4,0,0,0,0)$	$=$	4	$C(2,0,0,0,0,0)$	$=$	0
$C(0,6,0,0,0,0)$	$=$	6	$C(2,2,0,0,0,0)$	$=$	2
$C(0,8,0,0,0,0)$	$=$	8	$C(2,4,0,0,0,0)$	$=$	4
$C(1,0,0,0,0,0)$	$=$	0	$C(2,6,0,0,0,0)$	$=$	6
$C(1,2,0,0,0,0)$	$=$	2	$C(2,8,0,0,0,0)$	$=$	8
$C(1,4,0,0,0,0)$	$=$	4			

Starting in stage 1, job 1 is considered to be scheduled on time in intervals starting at b if $r_1 \geq b - p$ and $t_1 + p \leq d_1$ with t_1 as the start time of job 1. For $r_1 < b - p$, it holds $C(l,b,u,\alpha,\beta,x) := C(l-1,b,u,\alpha,\beta,x)$. For not well-defined cases, $C(l,b,u,\alpha,\beta,x) = \infty$. Table 4.16 shows the results for $C(1,b,u,\alpha,\beta,x)$. For instance, job 1 can be scheduled on time by considering an interval starting at $b = 0$. The minimum end of this interval is $C = 2$, or $C = 4$ if scheduling a maintenance operation along with job 1. For intervals starting at $b > 0$, we get $C(l,b,u,\alpha,\beta,x) = \infty$ because job 1 cannot be completed on time, i.e., $t_1 + p > d_1$.

After generating the function values for stage 1, we move on to stage 2, which in this instance is the last stage of the algorithm. Job 2 to is now considered to be scheduled on time following the same rules as in stage 1, but with $b = \min\{T\}$ as the only feasible start of the interval. In Table 4.17, feasible combinations of u, α, β, and x are shown. Determining $\max\{u\}$ from the set of function values $\neq \infty$, job 1 is the job to be scheduled on time. Two function values lead to an optimal solution with $u = 1$. In $C(2,0,1,1,0,1)$, a maintenance operation scheduled after job 1 is taken into account. Note that the completion time of the maintenance operation must be considered when determining the function value, but not whether the corresponding job can be completed on time. Finally, Equation (4.20) is applied to determine the number of late jobs, i.e. the number of on-time jobs is deducted from n: $2 - 1 = 1$ job is late. Figure 4.12 illustrates the Gantt chart of the optimal solution.

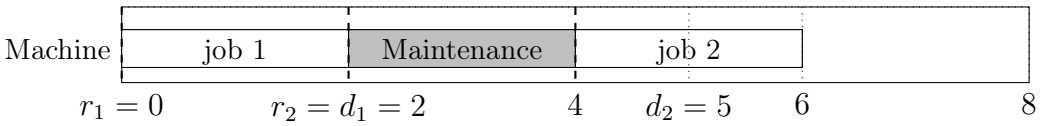
Tab. 4.16: Dynamic program solving $1|pd, p_j = p, r_j|\sum U_j$: value functions in stage 1

$C(l, b, u, \alpha, \beta, x)$		
$C(1, 0, 1, 1, 1, 0)$	$=$	2
$C(1, 0, 1, 1, 0, 1)$	$=$	4
$C(1, 2, 1, 1, 1, 0)$	$=$	∞
$C(1, 2, 1, 1, 0, 1)$	$=$	∞
$C(1, 4, 1, 1, 1, 0)$	$=$	∞
$C(1, 4, 1, 1, 0, 1)$	$=$	∞
$C(1, 6, 1, 1, 1, 0)$	$=$	∞
$C(1, 6, 1, 1, 0, 1)$	$=$	∞
$C(1, 8, 1, 1, 1, 0)$	$=$	∞
$C(1, 8, 1, 1, 0, 1)$	$=$	∞

Tab. 4.17: Dynamic program solving $1|pd, p_j = p, r_j|\sum U_j$: value functions in stage 2

$C(l, b, u, \alpha, \beta, x)$		
$C(2, 0, \mathbf{1}, 1, 1, 0)$	$=$	2
$C(2, 0, \mathbf{1}, 1, 0, 1)$	$=$	4
$C(2, 0, 2, 1, 1, 1)$	$=$	∞

Fig. 4.12: Gantt chart with final schedule of example instance for minimizing $\sum U_j$



4.3.4 Conclusion

In this section, we consider the single machine scheduling problem of minimizing the number of late jobs with uniform processing times, release dates, due dates, and *pd* maintenance, i.e., $1|pd, p_j = p, r_j|\sum U_j$. As a factor of machine wear and tear other than time, *pd* maintenance is presented as a relevant characteristic in real-world cases as well as in research to cause machine deterioration. To the best of our knowledge, this type of deterioration has not been extensively studied, and the runtime complexity of several typical scheduling layouts is still open. Therefore, we present an exact algorithm for solving the single machine layout $1|pd, p_j = p, r_j|\sum U_j$ in $\mathcal{O}(n^8)$.

In future work, extensive studies dealing with single machine and other machine scheduling layouts including *pd* maintenance should be carried out to obtain more insight into the influence of *pd* maintenance on scheduling problems and to generate suitable algorithms for real-world cases.

Chapter 5

Single Machine Scheduling Problems with Fixed Positioned Jobs

In this chapter, referred to the work of Hipp and Jaehn (2024a), we deal with the classical scheduling problem of assigning a set of jobs on a single machine to achieve due date or completion time related objectives. We assume a job set that includes jobs with specific position restrictions in terms of sequencing, called special jobs. This means that special jobs have to be scheduled at fixed positions in the final sequence; time restrictions are not given. Besides, the special jobs have the same characteristics as the remaining ones of the job set (e.g., ready times, etc.). Under consideration of these special jobs, we study the runtime complexity of problems with uniform due dates or processing times for the classical scheduling objective functions. Inter alia, we show the $\mathcal{NP}$-hardness of the single machine scheduling problem with uniform due date d to minimize total weighted late work, and present a pseudo-polynomial algorithm with runtime $\mathcal{O}(n^2d)$ to solve this problem optimally.

5.1 Introduction

In traditional machine scheduling, one of the main challenges is to determine job sequences that meet defined requirements in consideration of given constraints and objective functions. Jobs might be restricted by characteristics such as release dates or precedence constraints, but in general, job positions in the final sequence can be determined with

a high degree of freedom. In this chapter, we break this rule by assuming a subset of jobs that must be sequenced to predetermined positions in the final sequence. We call these jobs *special jobs* in contrast to regular jobs that do not have this restrictive position constraint. Applying the three-field notation of Graham et al. (1979), we introduce this constraint with *fp* for *fixed position* in the β-field. To the best of our knowledge, Calinescu et al. (2017) and Jaehn (2024) are the first who deal with this type of job constraint (see also chapters 3 and 4).

The motivation for studying models with fp jobs is of theoretical and practical nature. Focusing on machine scheduling with availability constraints, jobs with fixed positions can be seen as a variant that can be embedded in non-time related maintenance planning. In contrast to the majority of studies dealing with time-dependent machine availability constraints (see, e.g., current surveys of Gawiejnowicz (2020) and Geurtsen et al. (2023)), models that include non-time related factors to constrain machine availability depend, inter alia, on job characteristics in addition to processing times. To the best of our knowledge, the only survey investigating the entire research field of state-dependent machine availability in scheduling is provided by Briskorn et al. (2024). For instance, machines' wear and tear might be caused by the number of jobs regardless of their processing times, studied by Drozdowski et al. (2017), Hipp (2024), and Hipp and Jaehn (2024b). A common feature of these approaches is that the characteristics that influence the machine state and maintenance scheduling are not directly included in the objective functions. Our approach considering fp jobs can be interpreted as a variant of position-dependent maintenance, but with predefined positions, and as maintenance operations that have the same characteristics as regular jobs. In practice, service workers and their scheduling or the delivery and usage of operating resources that, e.g., have due dates to be scheduled in work schedules, can be seen as typical examples that represent this type of maintenance operations. Similar schemes can be found in logistic environments with non-negative delivery constraints presented by Briskorn et al. (2010) and Calinescu et al. (2017). The positions of special jobs, here trucks ensuring that enough items are available for subsequent jobs or that items are picked so that the warehouse capacity limit is adhered to, are mandatory to ensure the functionality of such systems.

Another similarity can be drawn to position-dependent processing times, e.g., studied by Mosheiov (2008), Mor and Mosheiov (2018), or Mosheiov et al. (2020). Here, jobs' processing times depend on the position at which they are scheduled. Assuming that the typical goal of a scheduler to minimize processing times to achieve the given objective, fp jobs can be seen as a variant of jobs with position-dependent processing times, which meet their regular processing time at the desired position and take very large values otherwise.

In this chapter, we deal with fp jobs as a subset of given job sets included in single machine scheduling environments and investigate the runtime complexity of algorithms for solving the corresponding problems. Since, to the best of our knowledge, research dealing with this job restriction is still in its infancy, we limit our studies to models with uniform processing times or uniform due dates, but address the commonly tackled range of objective functions in scheduling (see, e.g., Pinedo (2022)).

The reminder of this chapter is organized as followed. In Section 5.2, we present the problem description that holds for the subsequent problems, which are investigated regarding their runtime complexity. In Section 5.3, we concentrate on single machine problems with fp jobs, uniform processing times, and different objective functions, and discuss the runtime complexity of solving them. Section 5.4 follows the same scheme of dealing with single machine scheduling problems including fp jobs, the same set of objective functions, but with uniform due dates instead of processing times. For the single machine problem with fp jobs and uniform due dates minimizing total weighted late work, which we show to be $\mathcal{NP}$-hard, a pseudo-polynomial algorithm is presented to solve this problem to optimality. The approach is inspired by dynamic programs for solving the Knapsack Problem. After illustrating this algorithm with a small example instance, a brief conclusion is given in Section 5.5.

5.2 Problem Description

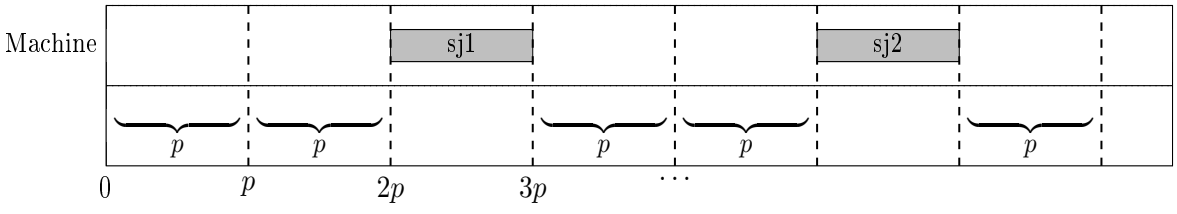
We study single machine problems scheduling jobs of set J with $j \in J = \{1, \ldots, n\}$ and processing time $p_j \in \mathbb{N}$. In addition, we consider special jobs with fixed positions in the sequence. A subset $Q \subsetneq J$ of special jobs is given. For each special job $j \in Q$, position $\sigma(j)$ is given, which defines the position of special job j in the final job sequence.

We assume without loss of generality that the special jobs are indexed in increasing order based on their position in the sequence, i.e. $j < j'$ for $(j, j') \in Q$ and $\sigma(j) < \sigma(j')$. Due to the fixed positions of the jobs in Q, only the positions of the remaining jobs have to be determined. These jobs in subset $J \backslash Q$ are called regular jobs. Regular and special jobs have the same characteristics with the same influence on the considered objective function. For instance, if weights, ready times, or due dates are given, they hold for both special and regular jobs.

5.3 Single Machine Problems with Fixed Positioned Jobs and Uniform Processing Times

Given uniform processing times $p_j = p$ with $p \in \mathbb{N}$ on a single machine and availability of each job $j \in J$ at the beginning of the planning horizon, each optimal schedule of the following problems is characterized by the same pattern for the considered objective functions. Since the machine can only run one job at a time, the set of start and completion times for scheduling regular jobs is known a priori. The start and completion time of each special job is known because of its fixed position in the final sequence. Regular jobs can be checked to be sequenced on the free slots of the schedule to determine an optimal solution for the respective objective function. This means that the problem $1|fp, p_j = p|\gamma$ with a completion time or due date related objective function as γ-entry, where no direct job dependencies need be considered, can be solved to optimality applying the assignment problem. Originally studied by Kuhn (1955) and Munkres (1957), the assignment problem is solvable in $\mathcal{O}(n^3)$. Figure 5.1 illustrates the general structure of this class of problems based on a small example.

Fig. 5.1: Structure of an optimal schedule for $1|fp, p_j = p|\gamma$ *with* $\sigma(sj1) = 3, \sigma(sj2) = 5$



In the following, we look at the runtime of approaches solving typical completion time and due date related objective functions, keeping in mind that the assignment problem can be seen as a "worst case" approach to determining an optimal solution for $1|fp, p_j = p|\gamma$.

5.3.1 Completion Time Related Objective Functions

Theorem 5.3.1. *Minimizing makespan in a single machine layout including special jobs with fixed positions and uniform processing times of all jobs,* $1|fp, p_j = p|C_{max}$*, is solvable to optimality in* $\mathcal{O}(n)$*.*

Proof. For any completion time based scheduling problem, we can assume a left-shifted schedule for an optimal solution (see, e.g., Pinedo (2022)). Thus, there are no idle times and the effect of each regular job on the makespan C_{max} is equal because uniform processing times of all n jobs are given. The completion times of the special jobs can be determined in advance (see Figure 5.1). As a result, the regular jobs can be inserted arbitrarily and the makespan can be easily determined with $C_{max} = n \cdot p$. □

Theorem 5.3.2. *Minimizing total completion time as the sum of completion times in a single machine layout with special jobs and* $p_j = p$*, i.e.* $1|fp, p_j = p|\sum C_j$*, is solvable to optimality in* $\mathcal{O}(n)$*.*

It holds the same as for Theorem 5.3.1. Uniform processing times for all jobs lead to optimal schedules regardless of the specific position of each job until the schedule is left-shifted.

Theorem 5.3.3. *Minimizing total weighted completion time in the single machine problem* $1|fp, p_j = p|\sum w_j C_j$ *is solvable to optimality in* $\mathcal{O}\big(n \log(n)\big)$ *by sorting all regular jobs in non-increasing order regarding their weight* w_j*.*

For $p_j = p$, the processing times do not have the characteristically significant effect on the total (weighted) completion time, as already mentioned in the course of Theorem 5.3.2. Moreover, it is easy to show via contradiction that sorting all regular jobs by non-increasing weights w_j leads to an optimal solution.

Proof. Let us assume for a contradiction, we have an optimal sequence Π with regular jobs i and k and $w_i < w_k$. We further assume that k starts $x \geq 1$ slots after i in the

schedule. As a result, we get start times $st_i = S$ and $st_k = S + xp$, and completion times $C_i = S + p$ and $C_k = S + (x + 1)p$, respectively. The objective function values for both jobs are given by $w_i(S + p)$ and $w_k\,(S + (x + 1)p)$. Now we can switch both jobs (because i and k are regular jobs, no position restrictions apply) without affecting any other completion times of earlier or later scheduled jobs, resulting in $w_kC_k = w_k(S + p)$ and $w_iC_i = w_i\,(S + (x + 1)p)$ and solution Π'. Comparing the sums of objective function values for both schedules, we get $w_i(S+p)+w_k(S+xp+p) = w_iS+w_ip+w_kS+w_kxp+w_kp$ for Π and $w_k(S + p) + w_i(S + xp + p) = w_kS + w_kp + w_iS + w_ixp + w_ip$ for Π', so finally we have to compare w_kxp with w_ixp. Because x and p are constants and $w_i < w_k$ must hold according to the formulated assumption, the objective function value of Π' is lower that of Π. This contradicts the statement that Π is an optimal solution.

Sorting jobs according their weights can be done in $\mathcal{O}\big(n\log(n)\big)$, shown for instance by Pinedo (2022), for a variety of basic priority rules. □

5.3.2 Due Date Related Objective Functions

Theorem 5.3.4. *Minimizing maximum lateness $L_{max} = \max\limits_{j\in J}\{C_j - d_j\}$ of scheduled jobs in a single machine layout $1|fp, p_j = p|L_{max}$ is solvable to optimality in $\mathcal{O}\big(n\log(n)\big)$ by applying earliest due date (EDD) rule on the regular jobs.*

Proof. It can be shown by contradiction that the EDD rule leads to an optimal sequence in $\mathcal{O}\big(n\log(n)\big)$. Because of $p_j = p$, the processing times do not affect the decision making.

Let us assume regular jobs i and k, $d_i = S + (x + 1)p$, $d_k = S + xp$, $x \geq 1$ slots including regular or special jobs, and an optimal schedule Π with start times $st_i = S$ and $st_k = S + xp$. Completion times $C_i = S + p$ and $C_k = S + (x + 1)p$ lead to lateness $L_i = -xp$ and $L_k = p$, thus $L^{\Pi}_{max} = p$. Now we can switch the positions of i and k without influencing the objective function values of the remaining jobs. In this new schedule Π', $st_i = S + xp$, $st_k = S$, $C_i = S + (x + 1)p$, and $C_k = S + p$. As a result, $L_i = 0$, $L_k = -(x-1)p$, and $L^{\Pi'}_{max} = 0$. Obviously, $L^{\Pi'}_{max} < L^{\Pi}_{max}$, which contradicts the statement that Π is an optimal solution. All jobs can be reordered regarding this procedure to obtain an EDD sequence that minimizes L_{max}. □

Theorem 5.3.5. *Minimizing the number of tardy jobs $\sum U_j$ in the single machine problem $1|fp, p_j = p|\sum U_j$ has a runtime complexity of $\mathcal{O}(n^2)$ by applying a modified version of the algorithm from Moore (1968).*

The algorithm from Moore (1968) is proven to solve $1||\sum U_j$ in polynomial time. We can modify this approach to solve $1|fp, p_j = p|\sum U_j$ in polynomial time as well. It is given a job instance J with job $j \in J$, due date $d_j \in \mathbb{N}$, special jobs Q and regular jobs $RJ = J \backslash Q$, processing time $p_j = p$ ($p \in \mathbb{N}$), and fixed positions $\sigma(j')$ for each special job $j' \in Q$. U_j is equal to one if j (or parts of j) is tardy, else $U_j = 0$. Because of the uniform processing times, the start and completion times of special jobs are known in advance as well as their objective function values. As a result, only regular jobs are considered in the following to seek an optimal solution for $1|fp, p_j = p|\sum U_j$.

All regular jobs of set RJ are sorted and scheduled with EDD rule (according to Moore). If no job is tardy, the algorithm terminates. Else, the first tardy job of the sequence, let us say job $i \in RJ$, is shifted to the end of the schedule. We differ two subsets, J' and I, where I contains the set of tardy jobs that are shifted to the end of the schedule, i.e. $i \in I$, and $J' = RJ \backslash I$. The jobs originally sequenced after i are now left-shifted one position. This procedure of assigning tardy jobs from J' to I and shifting remaining jobs in J' is repeated until J' contains only jobs that are completed on time. Compared to Moore's original algorithm with its runtime complexity of $\mathcal{O}(n^2)$, our modified approach differs only in the selection rule of the job to be shifted to the end of the schedule. Thus, the runtime complexity for solving $1|fp, p_j = p|\sum U_j$ is also $\mathcal{O}(n^2)$.

Now we prove that this modified version of the algorithm of Moore leads to an optimal solution. For this, we use the findings presented in the works of Moore (1968) and Cheriyan et al. (2021).

Proof. Let us assume a schedule with all regular jobs sorted by the EDD rule and indexed accordingly. If no job is tardy, i.e. $\sum U_{j \in RJ} = 0$, we have an optimal solution. Let us assume instead, we have a first tardy job $i \in RJ$ in the EDD schedule Π. Then we know that $i \cdot p > d_i$, and that $d_i = \max_{j' \in \{1,\dots,i\}} \{d_{j'}\}$. If we select job i and schedule it at the end of the schedule, we do not worsen $\sum U_j$, but jobs 1 to $i-1$ are still ordered according to EDD rule and are on time. Jobs $i+1$ through n are shifted to the left so that their

completion times are improved by p units each. The specific jobs remain at their current positions. We now have schedule Π' with $\sum U_j^{\Pi'} \leq \sum U_j^{\Pi}$. This argument is repeated until all on-time regular jobs are sequenced according to EDD rule and the tardy regular jobs are at the end of the sequence. □

Finally, we illustrate this approach with a small example of five regular and two special jobs. Figure 5.2 shows the initial schedule after scheduling the regular jobs according to EDD rule. An optimal solution after generating the two subsets is shown in the Gantt chart in Figure 5.3.

Fig. 5.2: Applying EDD rule to an example instance of $1|fp, p_j = p|\sum U_j$ *(in bold: tardy jobs)*

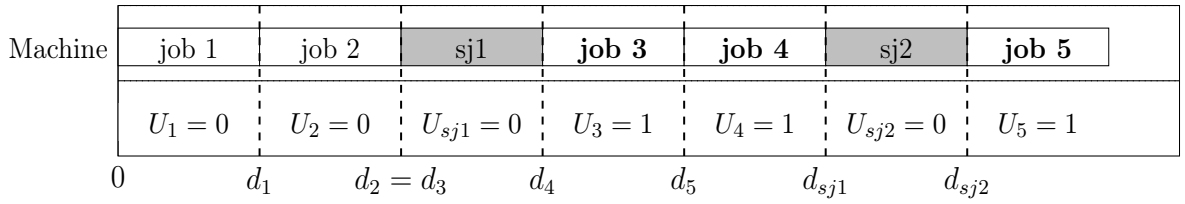
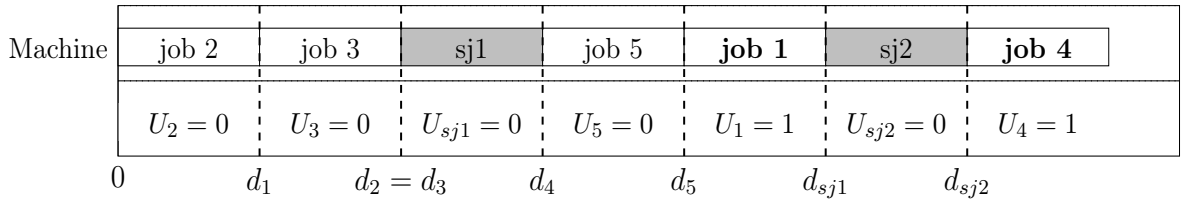


Fig. 5.3: Optimal schedule for the presented instance applying the modified algorithm of Moore for $1|fp, p_j = p|\sum U_j$ *(in bold: tardy jobs)*



Theorem 5.3.6. *Minimizing the weighted sum of tardy jobs for a single machine with fixed positions and uniform processing times,* $1|fp, p_j = p|\sum w_j U_j$*, is optimally solvable in* $\mathcal{O}(n^2)$ *by applying a modified version of Moore's algorithm.*

Again, the algorithm of Moore (1968) represents the basis for this approach. After sorting all regular jobs according to non-decreasing due dates d_j (EDD rule), we focus on subset J' including the first tardy job and all previous sequenced jobs. We select job $j \in J'$ with $\arg\min_{j \in J'}\{w_j\}$ and schedule it in I at the end of the sequence with I as the subset of tardy jobs. We repeat this approach until subset $RJ \backslash I$ contains only non-tardy jobs. Again, the runtime complexity does not differ from the original algorithm of Moore because only the selection rule is modified.

Based on the findings of Theorem 5.3.5 we show in the following by contradiction that the subset of tardy jobs contains the jobs with smallest w_j.

Proof. Let us assume, we have an optimal sequence Π with two regular jobs i and k at positions $\pi - x$ and π ($\pi \geq 2$) with start times $st_i = (\pi - x)p$ and $st_k = \pi p$, and $x \in [1, \pi)$ slots between i and k. In addition, $d_i = d_k = (\pi - x + 1)p$ and $w_i < w_k$ are given. In this schedule, $C_i = (\pi - x + 1)p$ and $C_k = (\pi + 1)p$, i.e. $U_i + U_k = w_k$. In a new schedule Π', we switch the positions of the two jobs. That is, $st_i = \pi p$, $st_k = (\pi - x)p$, $C_i = (\pi + 1)p$, and $C_k = (\pi - x + 1)p$. The objective function value of Π' is then $U_i + U_k = w_i$. At the beginning we defined $w_i < w_k$, i.e., the objective function value of Π' is lower than that of Π. This result contradicts the statement that Π is an optimal solution. □

Theorem 5.3.7. *Focusing on late work Y_j of job j, $1|fp, p_j = p|\sum Y_j$ is solvable to optimality in $\mathcal{O}\big(n\log(n)\big)$ by applying the algorithm of Potts and Van Wassenhove (1992) to the regular jobs.*

For job $j \in J$, Y_j is defined as the minimum of tardiness $T_j = \max_{j \in J}\{C_j - d_j, 0\}$ and p, i.e. $Y_j = \min_{j \in J}\{p, T_j\}$. Positions, start and completion times of all special jobs of subset Q are fixed (see Figure 5.1), so we focus only on regular jobs ($RJ = J \backslash Q$) in the following considerations. However, special jobs remain part of the schedule, so their processing times still have an impact on the regular jobs' completions times. As defined in the algorithm of Potts and Van Wassenhove, the regular jobs are first sorted according to EDD rule. Without loss of generality, we can reindex them according to the new order by leaving out the special jobs, so that we have $d_1 \leq d_2 \leq \ldots \leq d_{n-|Q|}$. Then we set $T_{max} = \max_{j \in RJ}\{T_j\}$ as the maximum tardiness of all regular jobs. We determine $u = \left\lceil \frac{T_{max}}{p} \right\rceil$ to define job sequences $\Pi_1 = (u, u+1, \ldots, n-|Q|, 1, \ldots, u-1)$ and $\Pi_2 = (u+1, \ldots, n-|Q|, 1, \ldots, u)$. These sequences do not include the special jobs.

Based on Π_1 or Π_2, we define π_j^f in Equation 5.1 as the last position of job $j \in J$ in the sequence, where $\pi_{rj \in RJ}$ is the position of a regular job in Π_1 and Π_2. Special jobs are indexed according to their positions, i.e. $s_j < sj'$ for $\sigma(sj) < \sigma(sj')$.

$$\pi_j^f = \begin{cases} \pi_{rj} + (sj-1), & if\ \pi_{rj} \in \varphi,\ \varphi = \big(\max\big\{0, \sigma(sj-1) - (sj-1)\big\}, \sigma(sj) - sj\big] \\ & \forall\ sj \in Q,\ rj \in RJ \\ \pi_{rj} + \max\limits_{sj \in Q}\big(sj\big), & else\ if\ \pi_{rj} \notin \varphi\ \forall\ sj \in Q,\ rj \in RJ \\ \sigma(j), & else\ if\ j \in Q \end{cases} \tag{5.1}$$

One of these final sequences Π_1^f and Π_2^f leads to the optimal schedule for $1|fp, p_j = p|\sum Y_j$. Since the sequencing of the regular jobs corresponds to the original algorithm of Potts and Van Wassenhove (1992), and inserting the special jobs does not increase the runtime, the original runtime of $\mathcal{O}\big(n\log(n)\big)$ still applies.

When we prove the statement of Theorem 5.3.7, it is given that special jobs and their objective function values are predetermined and are not part of the algorithm itself. Furthermore, based on Potts and Van Wassenhove (1992), we can postulate about the set of regular jobs that (partially) on-time scheduled jobs are sorted according to EDD rule, and that late jobs can be sequenced arbitrarily, since Y_j is bounded by p. As a result, we have to prove that one of the sequences mentioned in Theorem 5.3.7 leads to an optimal schedule.

Proof. A schedule is given where all regular jobs are sorted according to EDD rule and indexed accordingly. Special job $sj \in Q$ with $\sigma(sj)$ is a constant with start time $p(\sigma(sj)-1)$ and $Y_{sj} = \min\big\{p, \max\{p \cdot \sigma(sj) - d_{sj}, 0\}\big\}$ (see Figure 5.1). sj is not included in the reindexing, but its processing time must be considered in the completion times of subsequent regular jobs. Without loss of generality, we can assume to have a special job with $\sigma(sj) = 1$ and given Y_{sj}, and n regular jobs. The EDD sequence together with sj is then $(sj, 1, \ldots, n)$.

In the following, Y_j and T'_{max} refer to regular jobs. If $T'_{max} = 0$, all regular jobs are on time. Else, $T'_{max} > 0$. Remember that $u = \left\lceil \frac{T'_{max}}{p} \right\rceil$. We assume to receive an optimal schedule $(sj, j, \ldots, n, 1, \ldots, j-1)$ with $j = u$ or $j = u+1$. $u-1$ or u jobs are shifted to

the end of the schedule and $(u,\ldots,n)$ or $(u+1,\ldots,n)$ are EDD sequences. For $j=u+1$, jobs $(u+1,\ldots,n)$ are completed on time and jobs $(1,\ldots,u)$ are tardy with $\sum Y_j = pu$. A sequence with $j>u+1$ leads to $\sum Y_j > pu$. Then we do not have an optimal schedule, because compared to $j=u+1$ more jobs with small d_j are shifted to the end of the schedule.

Now we assume that $j<u$. Further, let $T'_{max}=p(k+q)-d_k$ for job $k>j$ with $d_k \geq d_j$, and $q=|Q'|$ with $sj \in Q'$ for $\sigma(sj)<\sigma(k)$. q displays the number of special jobs scheduled before k (here $q=1$)[1]. Then $u<k$ because $p(k+q)>T'_{max}\geq pu$ with $u=\left\lceil \frac{p(k+q)-d_k}{p} \right\rceil$. Without loss of generality, we can determine $j=u-1$ and receive job sequence $(sj,u-1,\ldots,k,\ldots,n,1,\ldots,u-2)$. The start time of k is obviously greater than or equal to $p(k-u+q+1)$ and not less than d_k because of $p(k-u+q+1)-d_k \geq 0$ as $T'_{max}-pu\geq 0$. Therefore, k is tardy and job sequence $(u-1,\ldots,k,\ldots,n)$ is not a sequence of non-tardy jobs.

As a result, only one of the schedules determined with $j=u$ or $j=u+1$ generates an optimal schedule. □

Theorem 5.3.8. *Minimizing total weighted late work in the problem* $1|fp,p_j=p|\sum w_jY_j$ *is solvable in* $\mathcal{O}(n^3)$.

Proof. We apply again the obvious findings illustrated in Figure 5.1. As already mentioned, we can restrict ourselves to the completion times for special jobs of Q and regular jobs of $RJ=J\backslash Q$ from set $\{0,p,2p,\ldots\}$, where the completion times of the special jobs are given a priori. For each job $j\in J$, it holds $C_j=\pi\cdot p$ with $\pi\in\{1,\ldots,n\}$ as the position of job j. Late work $Y_j(\pi)$ of job j at position π is $Y_j(\pi)=\min\limits_{j\in J}\{p,\max\{\pi p-d,0\}\}$ accordingly. Finally, we have to assign each job j to one of n positions in the schedule with costs of $w_jY_j(\pi)$. This problem corresponds exactly to the assignment problem solvable in $\mathcal{O}(n^3)$. □

Theorem 5.3.9. *Minimizing total tardiness* $T_j=\max\limits_{j\in J}\{C_j-d,0\}$ *in the single machine layout* $1|fp,p_j=p|\sum T_j$ *is solvable in* $\mathcal{O}\big(n\log(n)\big)$ *by applying EDD rule.*

The proof by contradiction presented for Theorem 5.3.4 also holds for $1|fp,p_j=p|\sum T_j$. As a result, EDD rule with its runtime of $\mathcal{O}\big(n\log(n)\big)$ leads to an optimal schedule for

1 Even though sj is not considered in the sorting procedure, its processing time influences T_j of the regular jobs.

$1|fp, p_j = p|\sum T_j$. Special jobs have no impact on the approach because their final position in the sequence, and thus their effect on the objective function, can be determined a priori.

Theorem 5.3.10. *Minimizing total weighted tardiness in the single machine scheduling problem* $1|fp, p_j = p|\sum w_j T_j$ *is solvable in* $\mathcal{O}(n^3)$.

The proof of Theorem 5.3.8 also holds for $1|fp, p_j = p|\sum w_j T_j$ by modifying the corresponding equations for $w_j T_j$. The approach to solving the assignment problem with its runtime of $\mathcal{O}(n^3)$ can be applied again.

Before looking at single machine scheduling problems with uniform due dates, the results for the examined layouts $1|fp, p_j = p|\gamma$ of this section are summarized in Table 5.1.

Tab. 5.1: Complexity results for single machine layouts including jobs with fixed positions (fp) and uniform processing times

	$\mathbf{1\|fp, p_j = p\|\gamma}$					
objective	$\mathbf{C_{max}}$	$\sum \mathbf{C_j}$	$\sum \mathbf{w_j C_j}$	$\mathbf{L_{max}}$	$\sum \mathbf{U_j}$	$\sum \mathbf{w_j U_j}$
runtime	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}\big(n\log(n)\big)$	$\mathcal{O}\big(n\log(n)\big)$	$\mathcal{O}\big(n^2\big)$	$\mathcal{O}\big(n^2\big)$

	$\mathbf{1\|fp, p_j = p\|\gamma}$					
objective	$\sum \mathbf{Y_j}$	$\sum \mathbf{w_j Y_j}$	$\sum \mathbf{T_j}$	$\sum \mathbf{w_j T_j}$		
runtime	$\mathcal{O}\big(n\log(n)\big)$	$\mathcal{O}(n^3)$	$\mathcal{O}\big(n\log(n)\big)$	$\mathcal{O}(n^3)$		

5.4 Single Machine Problems with Fixed Positioned Jobs and Uniform Due Dates

In this section, we study single machine scheduling problems with uniform due dates for regular and special jobs. In the following, we concentrate on due date-related objective functions because completion time-related objectives ($C_{max}, \sum C_j, \sum w_j C_j$) are generally not affected by due dates.

Theorem 5.4.1. *Minimizing maximum lateness in a single machine layout with uniform due dates and special jobs characterized by fixed positions,* $1|fp, d_j = d|L_{max}$, *is solvable in* $\mathcal{O}(n)$.

Proof. It is obvious that we can determine $C_{max} = \sum_{j \in J} p_j$ regardless of the job sequence because all jobs are available at the beginning of the planning horizon. Thus, $L_{max} = C_{max} - d$ holds for any job sequence with $d_j = d$. The sequencing can be done arbitrarily, taking into account the special jobs' characteristics. □

Theorem 5.4.2. *Minimizing the number of tardy jobs in the problem* $1|fp, d_j = d|\sum U_j$ *is optimally done in* $\mathcal{O}\big(n \log(n)\big)$ *by applying SPT priority rule to the regular jobs.*

We show via contradiction that the regular jobs of an optimal solution for the problem $1|fp, d_j = d|\sum U_j$ are always scheduled in non-decreasing order according to their processing times (SPT rule).

Proof. We assume to have an optimal solution with sequence Π and jobs i and k with $p_i > p_k$. i and k are regular jobs, $(i, k) \in J \backslash Q$, so that we can switch their position arbitrarily. We have start times $S_i = S$, $S_k = S + x + p_i$, and completion times $C_i = S + p_i$, $C_k = S + p_i + x + p_k$ where $x \in \mathbb{N}$ is an interval including a given number of jobs – regular and/or special jobs – between i and k. We set the uniform due date $d = S + p_k$. In this case, the objective function values are determined with $C_i - d = (S + p_i) - (S + p_k) > 0$ and $C_k - d = (S + p_i + x + p_k) - (S + p_k) = p_i + x$, i.e., $\sum U_j$ is given by $\sum_{j \in i,k} U_j^{\Pi} = 2$. We can switch the position of both jobs without influencing other jobs in the schedule and obtain sequence Π' with $C_k = S + p_k$ and $C_i = S + p_k + x + p_i$. As a result we get objective values $C_k - d = (S + p_k) - (S + p_k) = 0$ and $C_i - d = (S + p_k + x + p_i) - (S + p_k) = x + p_i$ and $\sum_{j \in i,k} U_j^{\Pi'} = 1$. Comparing both solutions we receive $\sum_{j \in i,k} U_j^{\Pi'} < \sum_{j \in i,k} U_j^{\Pi}$, which contradicts the initial statement that Π represents the sequence of an optimal solution.

These findings hold for any pair of regular jobs in $1|fp, d_j = d|\sum U_j$ (special jobs are fixed regarding their position, as is known) and lead to the application of the SPT rule to obtain an optimal solution. □

Theorem 5.4.3. *Minimizing total late work in the single machine layout* $1|fp, d_j = d|\sum Y_j$ *is solvable in* $\mathcal{O}(n)$.

For Theorem 5.4.3, we can use the findings for $1|fp, d_j = d|L_{max}$, illustrated in the explanations of Theorem 5.4.1. C_{max} is given regardless of the job sequencing. Since Y_j is bounded by p_j, we can determine $\sum Y_j$ with $\sum Y_j = max\{0, C_{max} - d\}$. Regular jobs can be sequenced arbitrarily considering the positions of special jobs.

Theorem 5.4.4. *If at least two special jobs are given, solving* $1|fp, d_j = d|\sum w_j Y_j$ *is* $\mathcal{NP}$*-hard in the ordinary sense.*

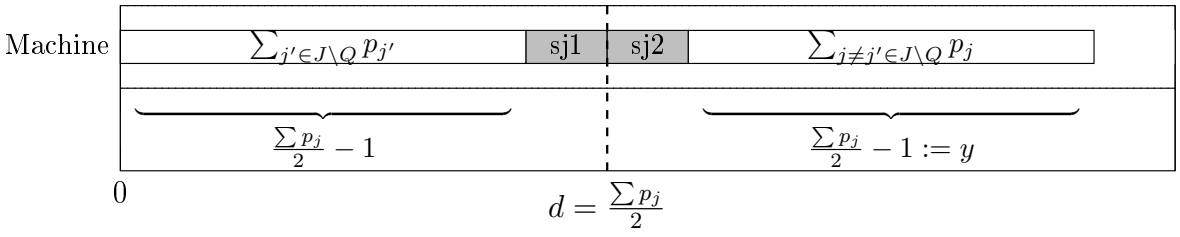
In the following, we show that the well-known partition problem (see for details Lenstra et al. (1977) and Garey and Johnson (1979)), which is known to be $\mathcal{NP}$-hard in the ordinary sense, can be transformed into an instance of $1|fp, d_j = d|\sum w_j Y_j$.

Proof. Given an instance with $2n$ jobs, where n is an arbitrary positive integer, i.e. $n \in \mathbb{N}$. Set J contains job j with processing time p_j and weight w_j. Uniform due date d is given by $d = \frac{\sum_{j\in J} p_j}{2}$. Two special jobs, $sj1$ and $sj2$, are given in subset $Q \subsetneq J$ with processing times $p_{sj1} = p_{sj2} = 1$, and weights $w_{sj1} = 2$ and $w_{sj2} = 0$, which have to be sequenced at positions $\sigma(sj1) = n$ and $\sigma(sj2) = n + 1$. For each regular job $j \in J\backslash Q$, $w_j = 1$ and $p_j \in \mathbb{N}$ are given.

Next, we show that the partition problem can be transformed into this instance of the problem $1|fp, d_j = d|\sum w_j Y_j$ by observing whether the given instance is a YES instance for the objective function value $y = \frac{\sum_{j\in J} p_j}{2} - 1$.

In the partition problem, $2n - 2$ elements (hereafter referred to as jobs) are given with the length $p_j \in \mathbb{N}$ of each job j. We call this subset J', which is tantamount to the set of regular jobs in the original problem. Together with the special jobs and their specifications, we have job set J. Because of the restrictions of the special jobs regarding their position in the sequence, $n - 1$ regular jobs, aggregated in subset J_1, must be scheduled before $sj1$ and the remaining $n - 1$ regular jobs in subset $J_2 = J'\backslash J_1$ must be scheduled after $sj2$. Considering $d = \frac{\sum_{j\in J} p_j}{2}$ and $y = \frac{\sum_{j\in J} p_j}{2} - 1$ (objective function value that is tested), we have a YES instance for the partition problem if and only if the sum of processing times of each set J_1 and J_2 is equal to $\frac{\sum_{j\in J} p_j}{2} - 1$ with $\sum_{j\in J_1} p_j = \sum_{j\in J_2} p_j$ and $\frac{\sum_{j\in J} p_j}{2} - 1 := y$. As a result, job subset J_1 and special job $sj1$ with $w_{sj1} = 2$ are completed on time with $C_{sj1} = d$. Job $sj2$ is completed tardy with $C_{sj2} = d + 1$ as well as all jobs of subset J_2, so we have $\sum_{j\in J} w_j Y_j = \frac{\sum_{j\in J} p_j}{2} - 1 = y$. Figure 5.4 illustrates the Gantt chart of the YES instance.

Fig. 5.4: Proof for Theorem 5.4.4: $1|fp, d_j = d|\sum w_j Y_j$



Next, we study the cases that lead to NO instances. Two cases are feasible, namely $\sum_{j \in J_1} p_j < \sum_{j \in J_2} p_j$ (i) and $\sum_{j \in J_1} p_j > \sum_{j \in J_2} p_j$ (ii).

Let us assume for case (i) that exactly one time unit is moved from subset J_1 to J_2 by exchanging one job from each subset. As a result, we have $\sum_{j \in J_1} p_j = \frac{\sum_{j \in J} p_j}{2} - 2$ and $\sum_{j \in J_2} p_j = \frac{\sum_{j \in J} p_j}{2}$. The number of jobs in both subsets must remain the same because of $sj1$ and $sj2$. In the new schedule, $C_{sj1} = \frac{\sum_{j \in J} p_j}{2} - 1$ and $C_{sj2} = \frac{\sum_{j \in J} p_j}{2}$, and $sj1$ as well as $sj2$ are consequently on time. All jobs in subset J_2 are tardy with $C_{j \in J_2} > \frac{\sum_{j \in J} p_j}{2}$. As illustrated in Figure 5.5, the new objective function value is now equal to $\frac{\sum_{j \in J} p_j}{2}$, i.e., $\sum_{j \in J} w_j Y_j > y$, which means we have a NO instance.

Fig. 5.5: Illustration of case (i): NO instance for $1|fp, d_j = d|\sum w_j Y_j$

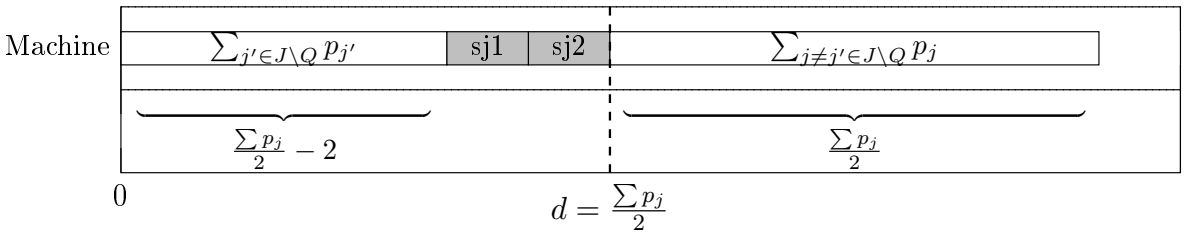
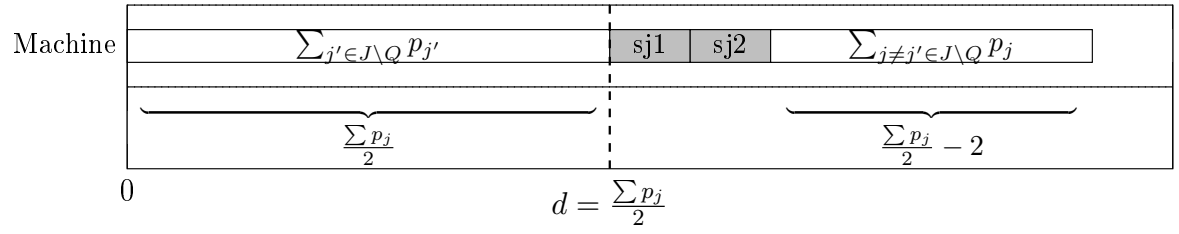


Fig. 5.6: Illustration of case (ii): NO instance for $1|fp, d_j = d|\sum w_j Y_j$



In case (ii), we switch again one job of each subset, but now with the goal of increasing J_1 by one time unit, i.e. $\sum_{j \in J_1} p_j = \frac{\sum_{j \in J} p_j}{2}$ and $\sum_{j \in J_2} p_j = \frac{\sum_{j \in J} p_j}{2} - 2$. As a result, $sj1$

and $sj2$ are tardy with $C_{sj1} = \frac{\sum_{j\in J} p_j}{2} + 1$ and $C_{sj_2} = \frac{\sum_{j\in J} p_j}{2} + 2$. The jobs of J_1 are on time with $C_{j\in J_1} \leq \frac{\sum_{j\in J} p_j}{2}$, all jobs of subset J_2 are tardy with $C_{j\in J_2} > \frac{\sum_{j\in J} p_j}{2} + 2$. The objective function value $\sum_{j\in J} w_j Y_j$ then consists of $w_{sj1}Y_{sj1} = 2$, $w_{sj2}Y_{sj2} = 0$, and $\sum w_{j\in J_2}Y_{j\in J_2} = \frac{\sum_{j\in J} p_j}{2} - 2$, i.e. $\sum_{j\in J} w_j Y_j = \frac{\sum_{j\in J} p_j}{2} > y$. Hence, we have again a NO instance. Figure 5.6 shows the schedule of this case. □

Theorem 5.4.5. *Minimizing total tardiness in the single machine scheduling problem* $1|fp, d_j = d|\sum T_j$ *is solvable to optimality in* $\mathcal{O}\big(n\log(n)\big)$ *by applying SPT rule.*

The proof follows the same argumentation as for Theorem 5.4.2. It is easy to show by contradiction that a schedule generated by a rule other than SPT does not lead to an optimal solution.

Before formulating an algorithm to solve $1|fp, d_j = d|\sum w_j Y_j$ to optimality, Table 5.2 shows the results of Section 5.4.

Tab. 5.2: Complexity results for single machine layouts inlcuding jobs with fixed positions (fp) and uniform due dates

	$\mathbf{1\vert fp, d_j = d\vert\gamma}$				
objective	$\mathbf{L_{max}}$	$\sum \mathbf{U_j}$	$\sum \mathbf{Y_j}$	$\sum \mathbf{w_j Y_j}$	$\sum \mathbf{T_j}$
runtime	$\mathcal{O}(n)$	$\mathcal{O}\big(n\log(n)\big)$	$\mathcal{O}(n)$	$\mathcal{NP}$-hard*	$\mathcal{O}\big(n\log(n)\big)$

*in the ordinary sense for ≥ 2 special jobs

5.4.1 Solution Approach for $1|fp, d_j = d|\sum w_j Y_j$

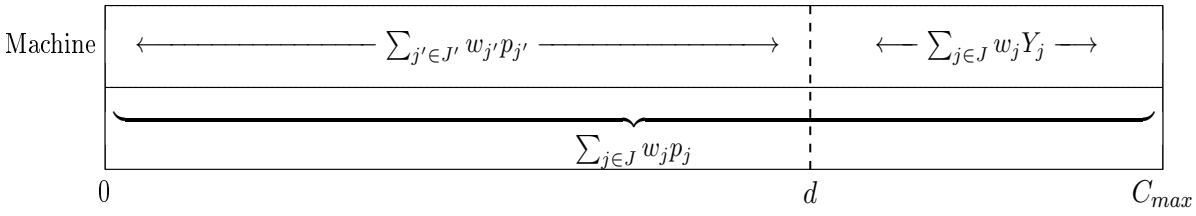
In Theorem 5.4.4, we study the problem $1|fp, d_j = d|\sum w_j Y_j$ and show that it is $\mathcal{NP}$-hard in the ordinary sense if at least two special jobs are given. In the following, we present a pseudo-polynomial algorithm that can solve an instance of this problem with an arbitrary number of special jobs in $\mathcal{O}(n^2 d)$ with d as the uniform due date for n jobs.

In addition to the problem description presented in Section 5.2, weight $w_j \in \mathbb{N}$ and the uniform due date $d_j = d$ with $d \in \mathbb{N}$ are given. The problem is to find a feasible schedule, i.e., all n jobs have to be scheduled considering the restrictions of the special jobs. Feasibility is given by any schedule, in which no two jobs j and j' of J with $j \neq j'$ overlap and each special job $sj \in Q$ is sequenced at its respective position $\sigma(sj)$. Among the feasible schedules, the goal is to minimize the weighted sum of late work $\sum_{j=1}^{n} w_j Y_j$ with $Y_{j\in J} = \min\big\{p, \max\{C_j - d, 0\}\big\}$.

Our solution approach to solving $1|fp, d_j = d|\sum w_j Y_j$ is inspired by dynamic programming algorithms dealing with Knapsack Problems (see, e.g., Kellerer et al. (2004)). The idea behind our algorithm is shown below.

We know that $w_j Y_j$ of any given job j cannot exceed $w_j p_j$, based on the definition of late work Y_j. Following this idea, $w_\varphi Y_\varphi$ of an completely tardy job φ is part of $\sum_{j \in J} w_j p_j$ of all jobs of set J. Moreover, the uniform due date d can be seen as a bound that splits job set J with $\sum_{j \in J} w_j p_j$ into a subset of on-time jobs $j' \in J'$ with $\sum w_{j'} p_{j'}$ $\left(\sum w_{j'} Y_{j'} = 0\right)$ and a subset of tardy jobs with $\varphi \in J''$ and $\sum w_\varphi Y_\varphi := \sum_{j \in J} w_j Y_j > 0$. Following the definitions for on-time and tardy jobs, it holds $\sum_{j \in J} w_j p_j = \sum_{j' \in J'} w_{j'} p_{j'} + \sum_{j \in J} w_j Y_j$. Since $\sum_{j \in J} w_j p_j$ is a constant, maximizing $\sum_{j' \in J'} w_{j'} p_{j'}$ leads to minimizing $\sum_{j \in J} w_j Y_j$, and vice versa. Figure 5.7 illustrates this relationship between $\sum w_j p_j$, $\sum w_{j'} p_{j'}$, and $\sum w_j Y_j$.

Fig. 5.7: Relation between $\sum w_j p_j$ and $\sum w_j Y_j$ for $1|fp, d_j = d|\sum w_j Y_j$



Using these findings, our problem can be transformed into a modified Knapsack Problem with the objective of maximizing $\sum w_{j'} p_{j'}$ of the job units scheduled on time, i.e., between points in time 0 and d, to minimize $\sum w_j Y_j$. Obviously, a job can only be on time or tardy, but single units of a job can be on time and other units can be tardy; in fact, such a job starts before d and is completed after d. Therefore, the original Knapsack Problem, where only complete items (here jobs) are allowed to be packed, is not applicable. The Fractional or Continuous Knapsack Problem (see, e.g., Korte and Vygen (2018)) cannot be applied either. In the Fractional Knapsack Problem (FKP), fractions of items can be packed to achieve the objective. All items are sorted in non-increasing order according to $\frac{w_j}{p_j}$ to assign them in that order until the given capacity is consumed regardless of splitting items (equivalent to the greedy algorithm). In our case, the characteristics of special jobs prevent this method from being applied, because their position restrictions lead to

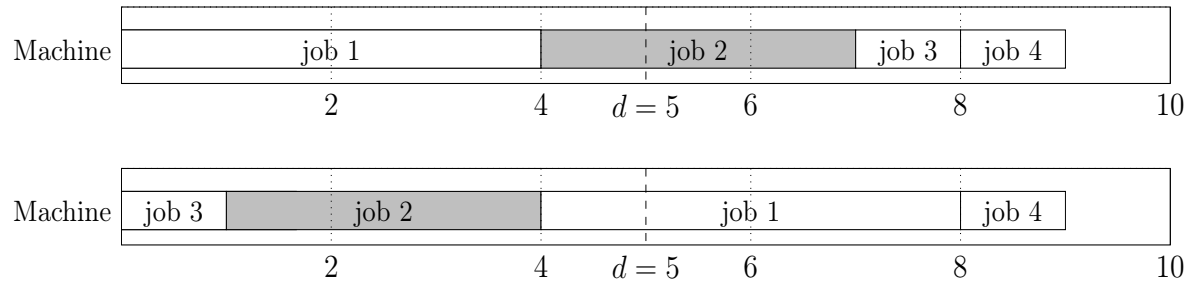
sequences that are not in line with the FKP approach. Table 5.3 and Figure 5.8 introduce a small example with four jobs, including one special job, to illustrate this issue.

Tab. 5.3: Example instance for $1|fp, d_j = d|\sum w_j Y_j$

job j	**1**	**2***	**3**	**4**
$\mathbf{p_j}$	4	3	1	1
$\mathbf{w_j}$	5	10	1	1
$d = 5$, $\sigma(2) = 2$				

*special job

Fig. 5.8: Final schedules for data of Table 5.3: Greedy approach (first Gantt) with $\sum w_j Y_j = 22$ *and the optimal schedule with* $\sum w_j Y_j = 16$ *(second Gantt)*



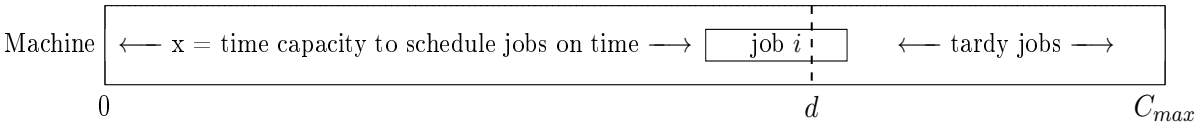
Therefore, we present a dynamic programming algorithm that modifies the decision steps of the classical approach for Knapsack Problems to solve our problem to optimality.

First, without loss of generality, we sort all regular jobs in non-increasing order regarding their weights. We assume that there exists an optimal schedule including at least one job j that can be scheduled on time, i.e., $C_j - d \leq 0$ (else, the algorithm is not required; $j = \underset{j \in J}{\operatorname{argmax}}(w_j)$ is assigned in the first position of the sequence, and the remaining jobs can be arbitrarily assigned to positions 2 to n). Furthermore, we postulate that in every optimal schedule a job j is completed at d, or that it starts before d but is completed tardy, i.e. $S_j < d \leq C_j$. For one job per stage of the dynamic program, two cases are distinguished as long as the workload of the machine is lower than d: Assign the job to the machine or not. (x, y, i) indicates the respective state in the dynamic program. The remaining time capacity on the machine to assign (units of) jobs on time is displayed by x and is initialized as well as bounded by d. y, bounded by n, declares the number of jobs that are scheduled on time. Job $i \in J \backslash Q$ is selected as the last of the regular jobs which to be scheduled to start before the deadline after testing the set of $J \backslash Q \backslash \{i\}$

regular jobs. This means that i can be thought of as a backup job required to ensure that free capacity of the machine is used before d, after all other regular jobs have been taken into account. As a result, the exact start time or position of i cannot be determined at the beginning of the algorithm. We restrict ourselves to regular jobs at this point. Special jobs are not considered for i because their positions are given a priori and the their scheduling cannot be controlled directly. We show later that special jobs are only scheduled together with regular jobs.

The idea of state (x, y, i) is illustrated in Figure 5.9. $|J\backslash Q| + 1$ stages are defined with stage 0 being the initialization and stages 1 to $|J\backslash Q|$ deciding for $|J\backslash Q|$ regular jobs whether they are checked to be scheduled on time (actually, we have to make $|J\backslash Q| < n$ decisions due to the characteristics of special jobs – if $|J\backslash Q| = n$, no special jobs are given). As a result, it holds $(x, y, i)_l$ as a state in stage $l \in \{0, \ldots, |J\backslash Q|\}$. The notation of the algorithm is given in Table 5.4.

Fig. 5.9: General idea of the algorithm to solve $1|fp, d_j = d|\sum w_j Y_j$



Tab. 5.4: Notation for the dynamic program solving $1|fp, d_j = d|\sum w_j Y_j$

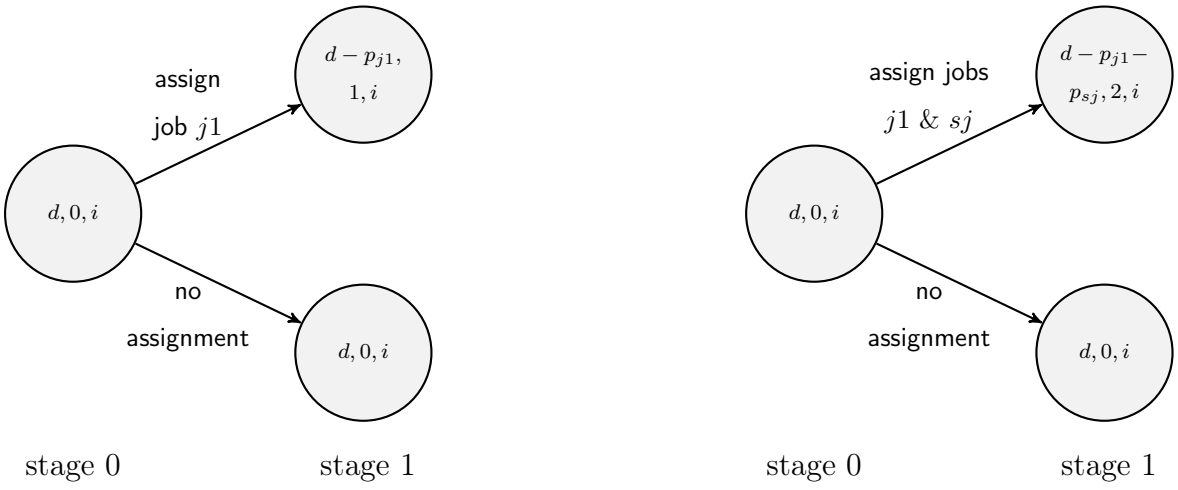
n	Number of jobs, $n \in \mathbb{N}$
J	Job set including n jobs
j	Job index, $j \in J$
d	Uniform due date for each job j, $d \in \mathbb{N}$
p_j	Processing time of job j, $p_j \in \mathbb{N}$
w_j	Weight of job j, $w_j \in \mathbb{N}$
C_j	Completion time of job j
Q	Subset of special jobs, $Q \subsetneq J$
$\sigma(j)$	Fix position of special job j, $\sigma(j) \in \{1, \ldots, n\}$
l	Stage of the dynamic program, $l \in \{0, \ldots, n\}$
i	Regular job that is considered last to be scheduled, $i \in J \backslash Q$
x	Time capacity to schedule jobs on time, $x \in \{0, \ldots, d\}$
x'	Expression of x in stage $l-1$ when a regular job is scheduled in l, $x' \in \{1, \ldots, d\}$, $x' > x$
y	Number of on-time scheduled jobs, $y \in \{0, \ldots, n\}$
$(x, y, i)_l$	State in stage l of the dynamic program with time capacity of x, y assigned jobs, and job i as the potentially last tested job to be scheduled on time
$f(x, y, i)_l$	Function value of state $(x, y, i)_l$ in stage l

Translated to the Knapsack Problem, x can be seen as the capacity of the bag, which has to be filled. The goal is the same as in the Knapsack Problem, namely to maximize the total weight of jobs (equal to the usage value of the items) assigned to the bag or in x (before d). As a result, the product of the weights and time units of tardy jobs, tantamount to $\sum w_j Y_j$, is minimized.

As mentioned above, the dynamic program has a maximum of $n+1$ stages, starting with state $(d, 0, i)_0$ in stage 0 characterized by a total capacity of d time units to schedule jobs on time, and $y = 0$ already scheduled jobs. Before considering the assignment of any job, $|J \backslash Q|$ branches are generated to set a regular job as i each in the initialization, i.e., to determine i as the last regular job that might be considered last for scheduling. We leave stage 0 to consider in each branch the assignment of one job per stage out of the set of regular jobs $J \backslash Q \backslash \{i\}$ under consideration of the objective function.

In stage 1, the first regular job is considered. If this job is not assigned, we stay in state $(d, 0, i)$ in stage 1, i.e., we have $(d, 0, i)_1$. If we assign job $j1$, we obtain state $(d-p_{j1}, 1, i)_1$ with $d - p_{j1}$ time units left to possibly schedule jobs on time in the following stages. This procedure must be modified for special special jobs. Let us assume, we are still in stage 1 and have a special job sj with $\sigma(sj) = 2$, which is tantamount to the requirement of assigning job sj at position 2. If the aforementioned job $j1$ is scheduled in the first stage, job sj must be assigned immediately after $j1$ to ensure the correct positioning of sj. Assuming jobs $j1$ and sj, we start again with state $(d, 0, i)_0$, but now go directly to state $(d-p_{j1}-p_{sj}, 2, i)_1$ instead of $(d-p_{j1}, 1, i)_1$ if $j1$ is determined to be scheduled. In a scenario in stage 1 with another special job $sj2$ to be assigned at position 3 ($\sigma(sj2) = 3$), state $(d-p_{j1}-p_i-p_{i2}, 3, i)_1$ must hold when assigning job $j1$, etc. Figure 5.10 shows the decision tree for the small examples of assigning $j1$ and $j1$ together with sj.

Fig. 5.10: Decision tree: Example with one regular job $j1$ (left) and $j1$ and special job sj (right) for $1|fp, d_j = d|\sum w_j Y_j$



Given the case of free machine capacity before d ($x > 0$) in stage $|J\backslash Q|$ (last stage of the algorithm), job i is assigned after all other regular (and special) jobs have been considered. Here no decision has to be made whether to assign i or not. If $x > 0$ is given, $x \leq p_i$ time units of job i are assigned and the algorithm terminates.

To evaluate which state $(x, y, i)_l$ leads to the optimal solution, $f(x, y, i)_l$ indicates the state's function value. f is determined based on the weight and processing time of the corresponding job. Let us look at the previous example with jobs $j1$, sj, and $sj2$ and assume that they are completed on time. Job i is still the one scheduled in the last

iteration. Then it holds $f(d,0,i)_0 = 0$ and $f(d-p_{j1}-p_{sj}-p_{sj2},3,i)_1 = w_{j1}p_{j1}+w_{sj}p_{sj}+w_{sj2}p_{sj2}$. If a next scheduled job j starts before d but is completed after d, i.e. $S_j < d < C_j$, only the time units of j processed before d are relevant for determining f. In this case, $x = d - S_j$ must hold and $w_j x$ is added to f. Therefore, $w_j \cdot \min\{x, p_j\}$ generally holds to determine f for j.

After stage $|J\backslash Q|$, $\max\limits_{l\in\{0,\ldots,|J\backslash Q|\}}\{f(x,y,i)_l\}$ displays the maximum function value for those jobs, that are scheduled on time. In fact, the set of feasible function values obtaining the maximum value over all stages of the dynamic program can be limited to those states that meet at least one of the defined stop criteria, i.e. $x = 0$, $y = n$, or $l = |J\backslash Q|$. When the dynamic program terminates, we have to determine $w_j Y_j$, which is the difference between the sum of $w_j p_j$ for job set J and $\max\limits_{l\in\{0,\ldots,|J\backslash Q|\}}\{f(x,y,i)_l\}$ (see Equation (5.2)).

$$\sum_{j\in J} w_j Y_j = \sum_{j\in J} w_j p_j - \max_{l\in\{0,\ldots,|J\backslash Q|\}}\{f(x,y,i)_l\} \qquad \forall\ x\in[0,d],\ y\in[0,n],\ i\in J\backslash Q \quad (5.2)$$

5.4.2 Pseudo-Polynomial Algorithm

In general, each state $(x,y,i)_l$ in the algorithm is given with function value $f(x,y,i)_l$ by Equation (5.3). $f(x,y,i)_l$ includes time capacity x, the number of on-time assigned jobs y in stage l, and job i that is selected to be the last to be considered for scheduling.

$$f(x,y,i)_l \quad \forall\ x\in[0,d], y\in\{0,\ldots,n\}, i\in J\backslash Q, l\in\{0,\ldots,n\} \tag{5.3}$$

In the initialization, i.e. $l = 0$, the function value in Equation (5.4) is equal to zero for any well-defined case. It must hold $x = d$ and $y = 0$ because no job is scheduled to this point. Each regular job is selected for i, resulting in $|J\backslash Q|$ branches in the dynamic program.

$$f(d,0,i)_0 = 0 \quad \forall\ i\in J\backslash Q \tag{5.4}$$

As long as we have $x > 0$, regular job $j \in J\backslash Q$ (and special jobs) is (are) considered to be scheduled on the machine in stage l. First, we assume that j, but no special job following j ($\sigma(sj) - \sigma(j) \neq 1$), is checked to be scheduled in l. If j is not assigned, $(x,y,i)_l := (x,y,i)_{l-1}$ with the function value of the previous stage. Else, we go from

state $(x, y, i)_{l-1}$ to $(\max\{0, x - p_j\}, y + 1, i)_l$. Here the machine capacity x is reduced by p_j or x' (if $x' < p_j$) with x' as the available capacity in stage $l - 1$. y is increased by one because j is assigned. The maximum function value $f(x, y, i)_l$ is determined by Equation (5.5) based on the decision to assign or not assign j.

$$f(x, y, i)_l = \max\big\{f(x', y - 1, i)_{l-1} + \min\{x', p_j\}w_j, f(x, y, i)_{l-1}\big\}$$
$$\forall\ x \in [0, d], x' \in [1, d], y \in [0, n], \sigma(sj) \neq y + 1, sj \in Q, i \in J\backslash Q, l \in [1, |J\backslash Q|] \quad (5.5)$$

The same pattern applies to the following cases, where a regular job j and one or more special jobs are considered to be scheduled in l.

First, we describe the case where j and a special job sj must be assigned at the same iteration because of the given position of sj for the final sequence. We assume that no other special job is a direct successor of sj, i.e., $|\sigma(sj') - \sigma(sj)| \neq 1$, $(sj, sj') \in Q$. If $x \leq p_j$, i.e., only j can be scheduled, Equation (5.5) applies (with the modification that $\sigma(sj) = y + 2$ holds). Else $(x' - p_j > 0)$, j and (parts of) sj can be checked to be assigned leading to the function value in Equation (5.6) with p_j and $\max\{0, x' - p_j - p_{sj}\}$ as the consumed capacity of j and sj.

$$f(x, y, i)_l = \max\big\{f(x', y - 2, i)_{l-1} + p_j w_j + \min\{x' - p_j, p_{sj}\}w_{sj}, f(x, y, i)_{l-1}\big\}$$
$$\forall\ x \in [0, d], x' \in [2, d], x' - p_j > 0, y \in [0, n], \sigma(sj) = y,$$
$$|\sigma(sj') - \sigma(sj)| \neq 1, (sj, sj') \in Q, i \in J\backslash Q, l \in [1, |J\backslash Q|] \quad (5.6)$$

As already mentioned in Section 5.4.1 regarding the specifics of the problem (see Figure 5.10), several special jobs may be required to be scheduled in succession. In this second case, all special jobs must be considered to be assigned in stage l together with regular job j. sj is the first and K is the last special job to be scheduled together with regular job j in l. Then $|\sigma(K) - \sigma(sj)| + 1$ special jobs have to be regarded. There are three cases to consider, indexed by Ψ, depending on the amount of time capacity x' available to schedule jobs on time. Equations (5.7) to (5.9) show each of them, summarized in Table 5.5.

Tab. 5.5: Cases to assign a regular job and subsequent special jobs

Case Ψ	**Description**	**Equation**
1	Job j and all subsequent special jobs can be scheduled before d	(5.7)
2	Job j and only the first special job can be scheduled (in parts) before d	(5.8)
3	Job j and special jobs sj to (parts of) sj'' can be scheduled before d	(5.9)

Again, $f(x, y, i)_l$ of each case Ψ is the maximum of the function values of the previous stage when it is decided not to assign a job, or of the function value after assigning the considered jobs depending on the available capacity x'.

In case $\Psi = 1$, formulated in Equation (5.7), we assume that all relevant special jobs and regular job j can be assigned on time, i.e. it must hold $x' - p_j - \sum_{k=sj}^{K} p_k \geq 0$.

$$f(x,y,i)_l = \max\Big\{f\big(x', y-2-(\sigma(K)-\sigma(sj)), i\big)_{l-1} + p_j w_j + \sum_{k=sj}^{K} p_k w_k, f(x,y,i)_{l-1}\Big\}$$
$$\forall\ x \in [0,d], x' \in [1,d], y \in [0,n], |\sigma(sj) - \sigma(sj')| = 1, \big(\sigma(sj), \sigma(sj')\big) \leq \sigma(K),$$
$$(sj, sj', K) \in Q, 0 \leq x' - p_j - \sum_{k=sj}^{K} p_k, \sigma(sj) = y+2, i \in J\backslash Q, l \in [1, |J\backslash Q|] \quad (5.7)$$

Equation (5.8) illustrates $\Psi = 2$. Here, only job j and parts of or special job sj can be scheduled on time ($S_{sj} < d$ and $C_{sj} \geq d$). In this case, special jobs following sj do not have to be considered. Furthermore, the iteration branch stops with $x = 0$ after assigning j and sj.

$$f(x,y,i)_l = \max\big\{f(x', y-2, i)_{l-1} + p_j w_j + min\{x' - p_j, p_{sj}\} w_{sj}, f(x,y,i)_{l-1}\big\}$$
$$\forall\ x \in [0,d], x' \in [2,d], y \in [0,n], sj \in Q, 0 < x' - p_j \leq p_{sj}, \sigma(sj) = y,$$
$$i \in J\backslash Q, l \in [1, |J\backslash Q|] \quad (5.8)$$

Finally, $\Psi = 3$ in Equation (5.9) describes the case that job j, special jobs sj to sj', and (parts of) special job sj'' with $\sigma(sj) < \sigma(sj') < \sigma(sj'') \leq \sigma(K)$ can be assigned on time. If the jobs are scheduled, the algorithm terminates again with $x = 0$.

$$f(x,y,i)_i = \max\big\{ f\big(x, y-2-(\sigma(sj'')-\sigma(sj)), i\big)_{l-1} + p_j w_j + \sum_{k=sj}^{sj'} p_k w_k + \min\{x' - p_j - \sum_{k=sj}^{sj'} p_k, p_{sj''}\} w_{sj''}, f(x,y,i)_{l-1} \big\}$$
$$\forall\ x \in [0,d], x' \in [3,d], y \in [0,n], (sj, sj', sj'') \in Q, \sigma(sj)+1 \leq \sigma(sj') = \sigma(sj'') - 1,$$
$$0 < x' - p_j - \sum_{k=sj}^{sj'} p_k \leq p_{sj''}, i \in J\backslash Q, l \in [1, |J\backslash Q|] \quad (5.9)$$

Each branch of the dynamic program is terminated when either $x = 0$, $y = n$, or $l = |J\backslash Q|$ is given. In stage $l = |J\backslash Q|$ (if no other stop criteria apply), job i (and subsequent special jobs) are checked to be scheduled on time applying the equations above. Finally, the optimal function value F^* is determined via Equation (5.10) by selecting the maximum function value computed in the algorithm.

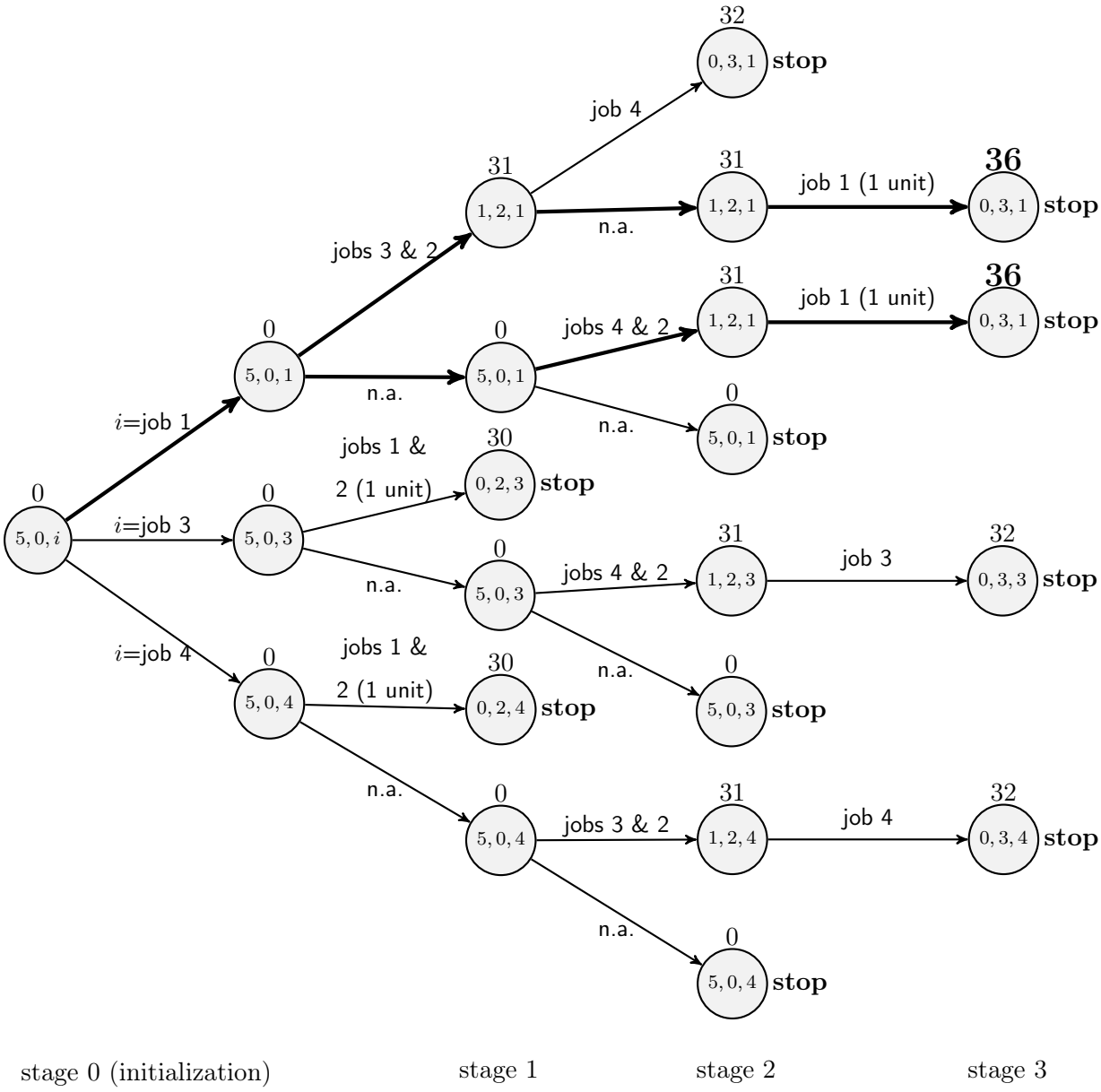
$$F^* = \max_{l \in [0, |J\backslash Q|]} \{f(x,y,i)_l\}$$
$$\forall\ x = 0 \vee y = n \vee l = |J\backslash Q|, x \in [0,d], y \in [0,n], i \in J\backslash Q \quad (5.10)$$

Since F^* does not provide the objective function value for $1|fp, d_j = d|\sum w_j Y_j$, we require Equation (5.11) to determine an optimal solution to the original problem of minimizing total weighted late work.

$$\sum_{j \in J} w_j Y_j = \sum_{j \in J} w_j p_j - F^* \quad (5.11)$$

To illustrate the approach of the dynamic program, we apply the small example from Table 5.3 with a total of four jobs, including three regular jobs and one special job, to our problem. In Figure 5.11, the complete tree of the dynamic program receiving a function value of 36 is generated by scheduling either (parts of) jobs $(3,2,1)$ or $(4,2,1)$ on time. With $\sum_{j \in J} w_j p_j = 52$, the optimal objective function value $\sum w_j Y_j$ of this instance is $52 - 36 = 16$ (see also Figure 5.8).

Fig. 5.11: Decision tree of dynamic program for example introduced in Table 5.3; n.a.=no assignment; optimal paths in bold



5.5 Conclusion

In this chapter, we study single machine scheduling problems including a subset of special jobs – compared to the remaining set of regular jobs – that must be sequenced at predefined fixed positions (fp) in the final sequence. This restricted type of jobs can be found in any practical environment where specific jobs stand for tasks that have do be performed at

fixed positions in production or logistics processes to ensure system functionality. Since, to the best of our knowledge, scheduling research dealing with fp jobs is still in its early stages, we limit ourselves to single machine problems with uniform processing times or due dates and the range of common completion time and due date related objective functions. These problems are examined regarding their runtime complexity, summarized in Table 5.6. In addition, we present a pseudo-polynomial algorithm for the problem of minimizing total weighted late work with uniform due dates, i.e. $1|fp, d_j = d| \sum w_j Y_j$. The approach is oriented on dynamic algorithms for solving the well-known Knapsack Problem.

Tab. 5.6: Complexity results for single machine problems including jobs with fixed positions (fp)

objective / layout	$\mathbf{C_{max}}$	$\sum \mathbf{C_j}$	$\sum \mathbf{w_j C_j}$	$\sum \mathbf{w_j T_j}$	$\sum \mathbf{w_j U_j}$
$\mathbf{1}\|\mathbf{fp}, \mathbf{p_j} = \mathbf{p}\|\gamma$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}\big(n \log(n)\big)$	$\mathcal{O}(n^3)$	$\mathcal{O}\big(n^2\big)$

objective / layout	$\mathbf{L_{max}}$	$\sum \mathbf{U_j}$	$\sum \mathbf{Y_j}$	$\sum \mathbf{w_j Y_j}$	$\sum \mathbf{T_j}$
$\mathbf{1}\|\mathbf{fp}, \mathbf{d_j} = \mathbf{d}\|\gamma$	$\mathcal{O}(n)$	$\mathcal{O}\big(n \log(n)\big)$	$\mathcal{O}(n)$	$\mathcal{NP}$-hard*†	$\mathcal{O}\big(n \log(n)\big)$
$\mathbf{1}\|\mathbf{fp}, \mathbf{p_j} = \mathbf{p}\|\gamma$	$\mathcal{O}\big(n \log(n)\big)$	$\mathcal{O}\big(n^2\big)$	$\mathcal{O}\big(n \log(n)\big)$	$\mathcal{O}(n^3)$	$\mathcal{O}\big(n \log(n)\big)$

*in the ordinary sense for ≥ 2 special jobs

†pseudo-polynomial algorithm in Section 5.4.2

As mentioned above, the field of scheduling problems regarding fp jobs still offers plenty of scope for further research. Besides Calinescu et al. (2017) and Jaehn (2024), this work is, to the best of our knowledge, the only one that studies problems with fp jobs. Numerous single machine layouts, and of course other machine environments, are worthy to look at regarding their complexity and solution approaches – both exact and heuristic algorithms. In a next step, these approaches can be applied to deal with practical cases involving the restriction of fp jobs.

Chapter 6

The Two-Stage no-wait Hybrid Flow Shop Scheduling Problem

In this chapter, from Hipp et al. (2024), we leave single machine environments and address hybrid flow shop (HFS) scheduling problems. HFS systems represent the typical flow shop production system adding parallel machines on at least one stage of operation, which can be found, inter alia, in steel and chemical industry. This chapter deals with the problem of obtaining fast and high quality solutions for minimizing makespan or total completion time (hereafter referred to as flow time) in a two-stage HFS scheduling problem including no-wait constraint. This means that jobs must be processed continuously after entering the machine environment. Due to the importance of efficient solutions in practical production applications, several heuristic algorithms are applied for instances up to 200 jobs. Dispatching rules that allow simple implementation with reasonable computational times, and constructive heuristics suitable for makespan and flow time scheduling problems are evaluated and compared with an optimization model. Based on the characteristics of HFS problems, two sequencing approaches are combined with machine assignment strategies. Then, a computational test using an extensive benchmark is conducted. The algorithms in use are evaluated and ranked based on different performance criteria, followed by an one sided hypothesis test of paired samples to compare their overall performance. Besides the expected results of heuristic approaches outperforming the respective optimization model, we show that dispatching rules such as Shortest Task Time or Shortest Processing Time Rule perform equivalent or even best for the whole range of instances.

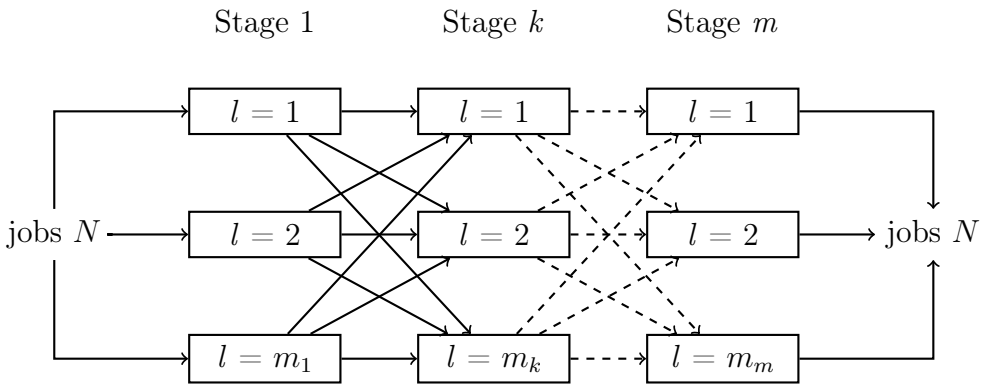
6.1 Introduction

In scheduling, one of the main difficulties is to display relevant real-world production systems in suitable mathematical models and furthermore, to get high quality output data for these layouts in a reasonable time. In this respect, flow shop scheduling systems with parallel machines on at least one stage, i.e., the hybrid flow shop (HFS), pose complex combinational problems that are very popular in research because of their practice-oriented layout. Ruiz and Vázquez-Rodríguez (2010) and Ribas et al. (2010) showed this already in 2010 by categorizing more than 200 papers dealing with HFS problems. HFS systems can be typically found in industries that have to manage, e.g., chemical reaction rates or temperature-dependent material characteristics in their processing steps (Hall and Sriskandarajah (1996)), like semi-conductor production, chemical industry, or steel production.

In general, the HFS scheduling problem consists of parallel machines on at least one stage of processing. The minimum number of serial production stages is two. Within all consecutive stages m with $k \in \{1, \ldots, m\}$ each stage contains a subset of $l \in \{1, \ldots, m_k\}$ machines per stage. Machines per stages might differ in speed. They can be classified as identical, uniform or unrelated which characterizes the influence of job-to-machine allocations. Each job $j \in \{1, \ldots, n\}$ with $n \in \mathbb{N}$ and processing time p_{jkl} has to be processed on at least one stage k on exactly one machine l. This expression can be reduced to p_{jk} by considering identical machines, where the speed of all machines per stage k is uniform. In this case, the machine assignment has no immediate influence on the completion time. The HFS layout is illustrated in Figure 6.1 and requires not only sequencing but also assignment when solving the corresponding scheduling problem (Framiñán et al. (2014)).

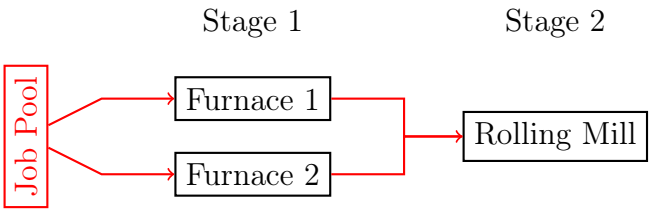
In this chapter, we consider the situation of a typical steel manufacturer, such as several those located in the Ruhr area in Germany, which is known for its steel industry. In particular, we focus on the production of steel sheets. Manufacturing steel sheets consists of two main steps (see Figure 6.2). First, raw steel is heated to approximately 1600 degrees and refined in a furnace to mold sheets of different sizes in the second step (Lancaster (1999)). While manufacturing and getting sheets in the demanded shape, it

Fig. 6.1: The hybrid flow shop layout (Framiñán et al. (2014))



is crucial to ensure a certain temperature level, as the sheets cannot be reshaped once they have cooled down excessively – i.e. waiting times must be prevented. Consequently, production planners have to assign arriving jobs (raw steel) to a furnace and be aware of preventing waiting times between the two production steps. This scheduling problem can be addressed by a HFS with no-wait constraint. No-wait constraint displays the necessity that a job is processed continuously, without waiting time between subsequent operations, as soon as it enters the production system. Because of a relatively short time horizon of sheet production planning and the risk of causing high costs by cooling down the sheets during production, it is fundamental to generate proper schedules in reasonable time.

Fig. 6.2: Machine layout in steel sheet production



The reminder of this chapter is organized as followed. In Section 6.2, the examined machine layout is presented and a literature review is carried out, summarizing the state of the art, no-wait connected, and HFS problem related papers. The mathematical model that is used as a benchmark is formulated in Section 6.3, followed by the studied heuristic approaches in Section 6.4. In Section 6.5, an extensive computational study is carried out and evaluated by a t-test for the differences of means of paired samples. A conclusion and outlook are given in Section 6.6.

6.2 Problem Statement and Literature Review

According to the aforementioned classification scheme of Graham et al. (1979), deterministic scheduling problems can be described by an $\alpha|\beta|\gamma$ triplet where α defines the machine layout, β the model constraints, and γ the objective function. For HFS problems, α is specified by $\alpha_1\alpha_2((\alpha_3\alpha_4^k)_{k=1}^{\alpha_2})$ with α_1 defining the overall machine layout, α_2 the number of consecutive stages, and α_3 in combination with α_4 the type and number of machines per stage (Vignier et al. (1999)). In this chapter, we focus on a two-stage HFS with identical parallel machines on the first stage, so we have two entries in our scheme, the first of which is characterized by P. In the β field, the mentioned no-wait constraint is denoted by $no-wait$. Because the start times of jobs on a stage must be equal to the completion times of the previous stage in no-wait scheduling layouts, determining the job sequence before assigning them to the first processing stage makes any further sequencing redundant. Passing of jobs between stages is not permitted. The objective functions studied are noted in the γ field and are makespan C_{max} and flow time minimization F. Makespan denotes the completion time of the last job on the last machine, flow time corresponds to the sum of completion times of all jobs on the last stage. The complete formulations are shown below. These two objectives are not only the most examined ones in flow shop scheduling research, but also display important performance measures in production systems. As shown by Gupta (1988) and Hoogeveen et al. (1996), even a HFS scheduling problem with two parallel machines on the first stage and a single machine on the second one is proved to be $\mathcal{NP}$-hard in the strong sense. The same holds for its inverse case and for the no-wait HFS considered in this work (Gupta et al. (1997)). Finally, all relevant data is considered static and deterministic.

The discussed scheduling problem, illustrated in Figure 6.2, can finally be stated as follows. The number of stages of the HFS (FH) is $M = 2$. The first stage $k = 1$ comprises two identical machines with $M_1 = \{1, 2\}$ and therefore $m_1 = 2$. In addition, stage two has $M_2 = \{1\}$ machine with $m_2 = 1$. All jobs are released at the release date $r_j = 0$, which implies the equality of flow time and completion time $F_j = C_{m\pi}$ for each job j on position π. Furthermore, no preemption of job operations is allowed. Once the processing of job j has started, no interruption is allowed until it is finished. Following

the classification of the shown $\alpha|\beta|\gamma$ triplet, the addressed HFS scheduling problem can be described by

$$FH2(P2^{(1)}, 1^{(2)})|no-wait|C_{max}$$

and

$$FH2(P2^{(1)}, 1^{(2)})|no-wait|F$$

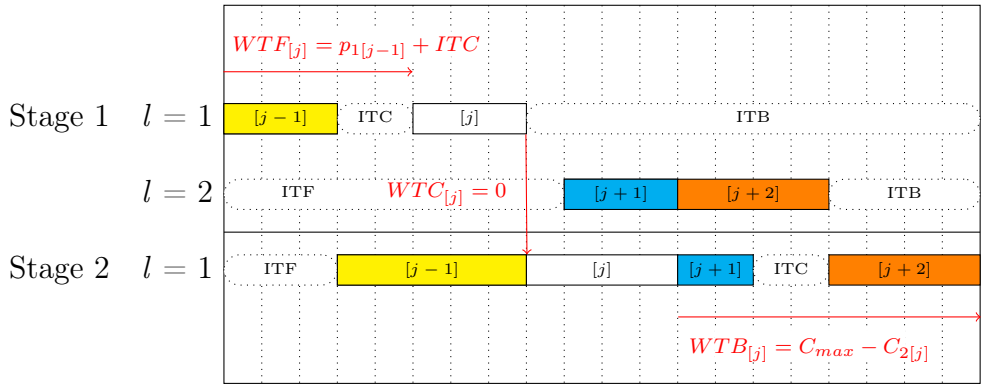
for single objectives makespan $C_{max} = \max_{j\in\{1,\dots,n\}}\{C_{mj}\}$ and flow time $F = \sum_{j=1}^{n} C_{mj}$.

The no-wait policy has a major impact on the idle times of machines, namely, it influences them in an adverse way. A simple example of the presented HFS can be used to show how the no-wait constraint enforces machine idle times in the front (ITF) and at the end (ITB) of the entire process, as well as during operation (ITC). Waiting time between operations, called core waiting time (WTC), has to be equal to zero if the no-wait policy is assumed. Because of the fact that two machines are available on the first stage and only one machine on the second one, the machine on the second stage can easily be identified as a potential bottleneck. The impact is illustrated in Figure 6.3. Increasing front waiting times (WTF) of the jobs on the first stage, due to the no-wait dependent right-shifting, does not automatically increase the makespan or flow time. The example schedule shows that idle times on the second stage, in case of $C_{1,j+2} > C_{2,j+1}$, enlarges completion time C_{2j+2} and thus has major impact on the objective value. Therefore, the aim is to start the processing on the second stage machine as early as possible and minimize its idle time.

As mentioned above, even the two-stage HFS layout is $\mathcal{NP}$-hard in the strong sense. Hence, this chapter mainly deals with several heuristics that are typically applied to solve HFS scheduling problems. We especially focus on dispatching rules. Dispatching rules are the most approachable heuristics than can achieve good quality solutions under defined circumstances at the same. In contrast to highly complex heuristics, dispatching rules and simple heuristics may not get as good results as other heuristics[1], but can imply the advantage of fast coding and runtimes. This is of high relevance for practical applications (Ruiz and Vázquez-Rodríguez (2010)). Manufacturers are not willing to

1 it is not set that a more complex algorithm guarantees better results than simple algorithms (Pinedo (2022))

Fig. 6.3: Schedule for two-stage HFS with no-wait constraint



WTF = Front waiting time ITF = Front Idle Time
WTC = Core Waiting Time ITC = Core Idle Time
WTB = Back Waiting Time ITB = Back Idle Time

make large investments in planning software, but are satisfied if good solutions can be ensured along with fast runtimes by reliable methods.

In the following, we focus on existing research dealing with (related) HFS scheduling problems in a brief literature review.

Literature Surveys and Taxonomy

Ruiz and Vázquez-Rodríguez (2010) give a comprehensive overview of solving HFS problems using exact algorithms, heuristics and metaheuristics. It is shown that only a few papers consider the application of dispatching rules in the HFS context. Makespan criterion is the most studied followed by a significantly smaller percentage of papers dedicated to flow time. Additionally, Ribas et al. (2010) include references structured according to problem classification and solution approaches. The survey by Allahverdi (2016) classifies more than 300 reviewed papers, paying attention not only to no-wait HFS layouts, but also to other scheduling problems regarding no-wait constraint. An older survey is provided by Linn and Zhang (1999), which discusses exact and approximate algorithms for the $2-$, $3-$, and $k-$stage HFS problems. In a taxonomy given by Quadt and Kuhn (2007), heuristic approaches are grouped into holistic and decomposition procedures (stage-, job-, problem oriented). Current surveys are compiled by Lee and Loong (2019), and Hwang and Lin (2018) who concentrate on two-stage HFS problems with dedicated machines and stage one or two as the bottleneck. Since the surveys of Hwang and Lin (2018) and Lee

and Loong (2019) do not examine no-wait related works as intensive as the ones of the 2000s, we primarily focus on scientific works since the millennium.

Papers Considering no-wait HFS Scheduling Problems

Exact methods for no-wait constrained layouts are studied by Guirchoun et al. (2005) dealing with server-processor relations and minimizing flow time. Mixed Integer Linear Programming (MILP) models for unlimited and zero interstage wait policies applied to chemical batch scheduling are developed by Lui and Karimi (2008). Makespan as well as due date related objective functions are explored. For the same production setting, Moon et al. (1996) propose MILP models for minimizing idle time and makespan by exploiting ideas of the Traveling Salesman Problem (TSP). Another MILP for the no-wait HFS is formulated by Meng et al. (2019), who evaluate and compare several MILP models for different HFS problems minimizing makespan according to their runtime complexity. Wang et al. (2018) also formulate a MILP, but for a particular two-stage no-wait HFS with m machines on the first and a no-idle machine on the second stage. Besides the MILP, two list scheduling heuristics are compiled and compared with other classical heuristics. Carpov et al. (2012) deal with a similar setting. List scheduling approaches are addressed to solve, inter alia, a two-stage no-wait no-idle HFS with precedence constraints. A branch-and-bound algorithm for a two-stage HFS is proposed by Wang et al. (2015) with upper bounds deduced from Johnson's algorithm (Johnson (1954)), the Longest Processing Time (LPT) rule and the heuristic developed by Gilmore and Gomory (1964). In another work, Wang et al. (2019) compare these approaches with the tabu search algorithm (Glover and McMillan (1986)). A computational test with randomly generated problem instances as well as a paired sample t-test are carried out. In a third work, Wang et al. (2020) add setup times to the mentioned layout and generate lower bounds for the branch-and-bound algorithm by applying Hungarian method (see Kuhn (1955) and Munkres (1957)) and compare this approach to tabu search and three dispatching rules.

Due to the proven $\mathcal{NP}$-hardness of HFS problems, it is common to decompose the overall problem. Typically, this is done by generating a job sequence in the first place and assigning jobs to machines in a second step. Wang et al. (2005) compile a heuristic algorithm for the two-stage HFS makespan minimization problem with no-wait constraint

between both stages and no-idle machines on the second stage. Jobs are sequenced according to their second stage processing time in a non-increasing manner using a modified LPT rule. Afterwards, they are scheduled on stage one and subsequently on stage two. A minimum deviation algorithm for a two-stage no-wait HFS with makespan objective is proposed by Xie et al. (2004). The algorithm is also based on a consecutive solution of sequence generation and machine assignment. Numerical experiments show the effectiveness of the heuristic approach. Chang et al. (2004) consider separated setup and removal times in a two-stage HFS with one machine at the first and two parallel machines on the second stage. A algorithm to solve flow shop scheduling problems is used to sequence the jobs with an assignment problem arising at the second stage where the last available machine is chosen. To evaluate the proposed algorithm, a computational study is performed applying a large randomly generated testbed. When it comes to non-hybrid flow shop scheduling with no-wait constraint, an algorithm that is often referred to is the already mentioned heuristic of Gilmore and Gomory (1964). It can be applied to two machine flow shop problems under certain conditions. Sriskandarajah and Sethi (1989) take advantage of this fact by splitting a HFS problem with $k = 2$ stages and $m_1 = m_2 \geq 2$ machines into several traditional flow shops with $k = 2$ stages and $m_1 = m_2 = 1$ machines. In addition, Sriskandarajah (1993) evaluates several scheduling algorithms for the no-wait HFS, including the Gilmore and Gomory heuristic, and analyzes their worst-case performance.

Metaheuristics have been shown to be widespread methods to address HFS scheduling problems. Grabowski and Pempera (2000) apply a tabu search based algorithm to minimize makespan. Besides the no-wait constraint, a no-store restriction and bottleneck machines are considered. A tabu search algorithm is developed for solving the HFS with no-wait, finite capacity and time windows ensuring a maximized value of orders served by Garcia and Lozano (2005). A genetic algorithm for makespan minimization is exploited by Wang and Liu (2013) using several constructive heuristics to derive upper bounds. The t-test of paired two samples for means is conducted to evaluate the results. Naderi et al. (2014b) formulate two MILP models that optimally solve the k-stage HFS scheduling problem for small instances. Large instances are approached with a hunting search metaheuristic based on the hunting behavior of animals. Several metaheuristics dealing with makespan objective are evaluated by Ghaleb and Alharkan (2015) implementing ex-

tensive computational experiments in the multi-stage no-wait HFS environment. They show that tabu search outperforms other procedures tested. Ramezani et al. (2015) deal with a HFS with up to eight stages including setup times besides no-wait constraint. To minimize makespan, they compile a metaheuristic that is a hybrid of variable neighborhood, simulated annealing and invasive weed optimization. Invasive weed optimization is orientated towards the behavior of weed colonies in agriculture. Another metaheuristic is applied by Rabiee et al. (2016) for a no-wait HFS with unrelated machines including several constraints, i.e. machine eligibility, setup times, and ready times. To reach the goal of minimizing mean tardiness of jobs, a biogeographically based algorithm is formulated. It transfers ideas of population movement, whereby it can be chosen between different habitats. In the end, it is about emigration and immigration between several possible solutions. A testbed with up to six stages and 120 jobs is implemented. In the bi-objective case, Jolai et al. (2013) focus on scheduling the no-wait two-stage HFS under the regular objectives of makespan and tardiness using simulated annealing.

Further Problem Related Papers

Shortest Processing Time (SPT), LPT, Minimum Deviation (MD), and Cumulative Minimum Deviation (CMD) rules are used by Narasimhan and Panwalkar (1984) to minimize waiting in process on the second stage of a two-stage HFS. The rules' performances are empirically evaluated with problem sizes up to 70 jobs. Narasimhan and Mangiameli (1987) formulate an extension of this procedure with the Generalized Cumulative Minimium Deviation rule (GCMD) by observing several waiting and idle time related objective functions. The adaptation and application of the heuristic of Nawaz et al. (1983) and other well-known constructive flow shop heuristics to the HFS problem are examined by Jungwattanakit et al. (2005). Ruiz et al. (2008) describe a formal model for HFS problems integrating the possibility of jobs skipping stages, the so-called hybrid flexible flow shop. For exact methods, a MILP is proposed and several dispatching rules as well as a modified NEH heuristic are statistically evaluated. Another study regarding HFS layouts with identical machines per stage is presented by Fernandez-Viagas et al. (2019). More than 40 algorithms are implemented, distinguished by their assignment and sequencing rules. For the same setting, four constructive heuristics, two memory-based and two ones based on

Johnson's algorithm, are evaluated and compared with 20 established heuristics (e.g. NEH and CDS of Campbell et al. (1970)) by Fernandez-Viagas et al. (2018). González-Neira and Montoya-Torres (2017) apply the Greedy Randomized Adaptive Search Procedure (GRASP), a popular greedy-based local search metaheuristic, to similar layouts with identical machines, optimizing makespan and due date related objective functions.

Examining total weighted completion time or weighted flow time, a particle swarm optimization algorithm, besides other metaheuristics, is presented and tested with randomly generated problem instances by Tang and Wang (2010). Makespan and mean flow time are investigated by Brah and Loo (1999) applying flow shop heuristics to the HFS allowing waiting times of jobs. Öztop et al. (2019) focus on minimizing flow time in the setting of a five-stage HFS with identical machines. Four variants of metaheuristics based on iterated greedy (Ruiz and Stützle (2007)) and a variable block insertion heuristic are compared to results generated with a MILP. Finally, Alfaro-Fernández et al. (2020) present a method for automatically obtaining appropriate algorithms for specific HFS layouts. This Automatic Algorithm Design (AAD) is implemented to optimize the single objectives makespan, flow time, and total weighted earliness and tardiness without any manual adjustment.

All of these presented studies either examine waiting time related HFS problems or focus on dispatching rules and fast processing constructive heuristics in flow shop environments. In this work, both research fields are combined. Several dispatching rules are compared with heuristics for the declared no-wait HFS scheduling problems

$$FH2(P2^{(1)}, 1^{(2)})|no-wait|C_{max}$$

and

$$FH2(P2^{(1)}, 1^{(2)})|no-wait|F$$

by applying the same instances to receive information about which algorithm fits best for which type of problem.

6.3 Mathematical Model Formulation

In the following, a mixed integer linear programming model (MILP) is formulated[2] for the two-stage HFS with no-wait constraint to minimize makespan

$$FH2(P2^{(1)}, 1^{(2)})|no-wait|C_{max}$$

and flow time

$$FH2(P2^{(1)}, 1^{(2)})|no-wait|F.$$

In the context of the computational study (Section 6.5), this MILP is used as a benchmark to gain first findings by comparing its results with those of the heuristics. Subsequent indices, parameters and variables of Table 6.1 apply.

Tab. 6.1: Notation of the MILP to solve $FH2(P2^{(1)}, 1^{(2)})|no-wait|\gamma$ minimizing C_{max} and F

n	Number of jobs, $n \in \mathbb{N}$
k	Stage index, $k \in \{1, 2\}$
j	Job index, $j \in \{1, \ldots, n\}$
π	Position of job j in sequence, $\pi \in \{1, \ldots, n\}$
m_k	Number of machines in stage k, $m_k \in \mathbb{N}$
l	Machine index in stage k, $l \in \{1, \ldots, m_k\}$
p_{kj}	Processing time of job j in stage k, $p_{kj} \in \mathbb{N}$
$S_{k\pi}$	Starting time of the job on position π in stage k
$C_{k\pi}$	Completion Time of the job on position π in stage k
$x_{kj\pi l}$	Binary variable, $= 1$ if job j in stage k on position π is processed on machine l, otherwise $= 0$
Z	Objective
C_{max}	Makespan
F	Flow time

2 In fact, we formulate one MILP that can be applied to both objective functions, but two seperate MILPs have to be implemented.

$$Min\ Z = C_{max} \tag{6.1}$$

$$Min\ Z = F \tag{6.2}$$

subject to

$$\sum_{\pi=1}^{n}\sum_{l=1}^{m_k} x_{kj\pi l} = 1 \qquad \forall\ k, j \tag{6.3}$$

$$\sum_{j=1}^{n}\sum_{l=1}^{m_k} x_{kj\pi l} = 1 \qquad \forall\ k, \pi \tag{6.4}$$

$$S_{11} = 0 \tag{6.5}$$

$$S_{k+1\pi} = S_{k\pi} + \sum_{j=1}^{n}\sum_{l=1}^{m_k} p_{kj} \cdot x_{kj\pi l} \qquad \forall\ k = 1, \pi \tag{6.6}$$

$$S_{k\pi} \geq C_{k+1\pi-1} - \sum_{j=1}^{n}\sum_{l=1}^{m_k} p_{k\pi} \cdot x_{kj\pi l} \qquad \forall\ k = 1,\ \pi \in \{2, \ldots, n\} \tag{6.7}$$

$$C_{k\pi} = C_{k-1\pi} + \sum_{j=1}^{n}\sum_{l=1}^{m_k} p_{k\pi} \cdot x_{kj\pi l} \qquad \forall\ k = 2, \pi \tag{6.8}$$

$$C_{k-1\pi} = S_{k\pi} \qquad \forall\ k = 2, \pi \tag{6.9}$$

$$S_{k\pi}, C_{k\pi} \geq 0 \qquad \forall\ k, \pi \tag{6.10}$$

$$x_{kj\pi l} \in \{0, 1\} \qquad \forall\ k, j, \pi, l \tag{6.11}$$

$$C_{max} \geq C_{m\pi} \qquad \forall\ \pi \tag{6.12}$$

$$F = \sum_{\pi=1}^{n} C_{m\pi} \tag{6.13}$$

The single objectives makespan (6.1) and flow time (6.2) are optimized in separate models, each subject to constraints (6.3) to (6.11). Constraint (6.3) ensures that each job j is allocated to only one position π and one machine on each stage k. Conversely, Constraint (6.4) states that every job j is operated by exactly one machine l on each stage k. The start time of the first job on the first machine is set to zero in (6.5).

For successive jobs, start times $S_{k\pi}$ are calculated with Equation (6.6). Processing time p_{kj} does not directly depend on any machine l of a stage k as all machine speeds per stage are identical. The start time of the job at position π on the first stage is connected to the completion time of the previous job $\pi - 1$ on stage two to ensure machine availability on the second stage, leading to Inequation (6.7). In Equation (6.8), the completion time of job π on stage two $C_{2\pi}$ is determined in consideration of the no-wait constraint. Starting time $S_{k\pi}$ of job π on stage two has to be equivalent to its completion time on stage one (6.9). Constraints (6.10) and (6.11) define the non-negativity of all decision variables and the range of values for the binary variable. The optimization of makespan and flow time (see (6.1) and (6.2)) are determined in equations (6.12) and (6.13), respectively.

6.4 Solution Approaches

Since the studied problems are $\mathcal{NP}$-hard in the strong sense and as a result, the formulated MILP model is not able to solve large instances to optimality (Gupta et al. (1997)). A popular way to find high quality solutions is then to implement heuristic approaches. Since the HFS layout is a composition of flow shop and parallel machine models, following approaches are deployed. The scheduling problem is divided into subproblems by sequencing and assigning each job to one machine per stage. Sequencing means finding a solution with a unique job sequence to achieve the chosen objective under all constraints. Because more than one machine is available in the first stage, assigning these sequenced jobs to machines is additionally required[3]. To solve these subproblems consecutively, all jobs are first sequenced using dispatching rules or a flow shop algorithm, and then scheduled on the machines with assignment rules. Again using the example from Figure 6.3, we get the job sequence $(j-1, j, j+1, j+2)$. The assignment consists of the decision to schedule jobs $j-1$ and j on machine one and jobs $j+1$ and $j+2$ on machine two on stage one. Two approaches are applied as each of them reduces the original problem in its own way. The comparison will be an indication of whether there has been a significant difference in the performance of the two systems.

3 Job assignment is typically required in parallel machine scheduling problems.

6.4.1 Sequencing

Two sequencing approaches introduced by Framiñán et al. (2003), a stage-oriented decomposition (A1) and the reduction to a flow shop problem (A2), are conducted. Appropriate dispatching rules and simple heuristics are implemented in the course of these approaches.

Approach 1: stage oriented decomposition (A1)

Because of the no-wait constraint, start time S_{2j} of job j on the second stage is equal to its completion time C_{1j} on the first stage, $S_{2j} = C_{1j}$. Therefore, the following indicator values are formulated using the processing time p_{1j} of the first stage.

- processing time on the first stage, p_{1j}:
  The first stage contains two identical, parallel machines with processing time p_{1j} which only depends on the respective job j.
- sum of absolute differences of processing time of jobs on stage one, a_{1j}:
  Indicator a_{1j} displays the deviation of processing times of job j and job j' with $j \neq j'$. It is used to explore the influence of jobs' processing time equality and is determined with Equation (6.14) (Framiñán et al. (2003)).

$$a_{1j} = \sum_{j'=1 \; j' \neq j}^{n} |p_{1j} - p_{1j'}| \qquad \forall \; j \in \{1, \ldots, n\} \tag{6.14}$$

To generate sequence S, job $j' \in U$ is selected from a number of unscheduled jobs $U \in \{1, \ldots, n\}$. Depending on the used dispatching rule, the next job in the sequence is chosen according to Equation (6.15) or (6.16). Each job g chosen according to Equation (6.15) takes position S_g. In this case, an unsequenced job with minimal processing time is selected. All jobs h of position S_h are determined with Equation (6.16) after selecting the job with maximum processing time (Framiñán et al. (2014)).

$$S_g : p_{1g} = \min_{j'=1}^{U}(p_{1j'}) \qquad resp. \quad a_{1g} = \min_{j'=1}^{U}(a_{1j'}) \tag{6.15}$$

$$S_h : p_{1h} = \max_{j'=1}^{U}(p_{1j'}) \qquad resp. \quad a_{1h} = \max_{j'=1}^{U}(a_{1j'}) \tag{6.16}$$

The sequencing decision is made with each of the following dispatching rules.

(1) STT All jobs are sorted in ascending order of their indicator value (Shortest Task Time Rule). Job g from set U is appended to sequence S_{STT} according to equations (6.15) and (6.17). Unlike SPT Rule, which sums up processing times of all stages, only processing times of the first stage are considered.

$$S_{STT} = (S_1, S_2, S_g, \ldots, S_n) \quad \forall\, g \in \{1, \ldots, n\} \tag{6.17}$$

(2) LTT All jobs are sorted in descending order of their indicator value for stage one (Longest Task Time Rule). Job h from set U is appended to sequence S_{LTT} in Equation (6.18) with Equation (6.16).

$$S_{LTT} = (S_1, S_2, S_h, \ldots, S_n) \quad \forall\, h \in \{1, \ldots, n\} \tag{6.18}$$

(3) HILL Job g is added to the sequence S_{HILL} in Equation (6.19) according to (6.15) so that jobs with high indicator values are positioned in the middle and those with lower values at the beginning and end of the schedule, as shown by Framiñán et al. (2003).

$$S_{HILL} = (S_1, S_3, S_g, \ldots, S_n, \ldots, S_{g+1}, S_4, S_2) \quad \forall\, g \in \{1, \ldots, n-1\} \tag{6.19}$$

(4) VALLEY In contrast to HILL, VALLEY selects job h according to Equation (6.16) and adds it to the sequence S_{VALLEY} (6.20) so that jobs with low indicator values are positioned in the middle and those with high values at the beginning and end (Framiñán et al. (2003)).

$$S_{VALLEY} = (S_1, S_3, S_h, \ldots, S_n, \ldots, S_{h+1}, S_4, S_2) \quad \forall\, h \in \{1, \ldots, n-1\} \tag{6.20}$$

(5) HI/LOHI All jobs are sequenced in alternating manner, starting with the minimal indicator value followed by a job with a high indicator value (LOHI). The HI/- criterion means that the sequence starts with high differences between the jobs' indicator values.

To generate sequence $S_{HI/LOHI}$ in (6.21), positions S_g and S_h are appended with help of the subsets $g \in \{1, \ldots, n\}$ and $h \in \{1, \ldots, n\}$ with $g \neq h$ (Framiñán et al. (2003)).

$$S_{HI/LOHI} = (S_g, S_h, \ldots, S_n) \quad \forall\ g = \{1, \ldots, n\},\ h \in \{1, \ldots, n\},\ g \neq h \tag{6.21}$$

(6) HI/HILO At the beginning, jobs with high differences in processing times (HI-) are sequenced, with maximal and minimal processing times alternating (HILO). Positions S_h and S_g are appended to $S_{HI/HILO}$ in (6.22) with subsets $g \in \{1, \ldots, n\}$ and $h \in \{1, \ldots, n\}$ with $g \neq h$ (Framiñán et al. (2003)).

$$S_{HI/HILO} = (S_h, S_g, \ldots, S_n) \quad \forall\ g \in \{1, \ldots, n\},\ h \in \{1, \ldots, n\},\ g \neq h \tag{6.22}$$

(7) LO/LOHI The jobs are sequenced similarly to HI/LOHI rule (LOHI), starting with small differences between the jobs (LO/-) (Framiñán et al. (2003)).

$$S_{LO/LOHI} = (S_g, S_h, \ldots, S_1) \quad \forall\ g \in \{1, \ldots, n\},\ h \in \{1, \ldots, n\},\ g \neq h \tag{6.23}$$

(8) LO/HILO Similar to LO/LOHI the sequence, $S_{LO/HILO}$ (6.24) is built by alternating positions S_h and S_g Framiñán et al. (2003).

$$S_{LO/HILO} = (S_h, S_g, \ldots, S_1) \quad \forall\ g \in \{1, \ldots, n\},\ h \in \{1, \ldots, n\},\ g \neq h \tag{6.24}$$

STT is chosen because SPT, as closely related algorithm, is well known to achieve high quality results for both makespan and flow time objectives (Haupt (1989)). Analogous to the STT rule, the processing on the second stage can begin as early as possible with HILL. It is to be investigated whether "buffer zones" of low processing times at the start and end of the schedule have a positive impact by enabling shifting. LTT and VALLEY are chosen as counterparts to STT and HILL. The alternating rules may balance the right-shifting of jobs caused by the no-wait constraint.

Approach 2: dedicated flow shop algorithms (A2)

For the second approach, both stages are considered assuming we have only one machine on the first stage. The indicator values are modified as follows:

- processing time on both stages p_j:
  Processing time p_j is the sum of a job's j processing time over all stages k (6.25).

$$p_j = \sum_{k=1}^{m} p_{kj} \qquad \forall\ j \in \{1, \ldots, n\} \tag{6.25}$$

- sum of absolute difference of job's j processing time on all stages a_j in (6.26):
  For job j, processing time deviation to every job j' with $j \neq j'$ is accumulated over all k stages (Framiñán et al. (2003)).

$$a_j = \sum_{k=1}^{m} \sum_{j'=1, j' \neq j}^{n} |p_{kj} - p_{kj'}| \quad \forall\ j \in \{1, \ldots, n\} \tag{6.26}$$

The priority rules mentioned above are applied in the same way as explained before, using p_j and a_j instead of p_{1j} and a_{1j} (see Equation (6.14)). This means that the processing times of all stages are now considered, and SPT and LPT are applied instead of STT and LTT. Three additional constructive heuristics, NEH, the one of Liu and Reeves (LR), as well as the approach of Gilmore and Gomory (GILGOM) are outlined in the following. These algorithms are selected because each of them is known to be very well suited for the problem under consideration. It has been shown in numerous studies that NEH is one of the best constructive heuristics for the permutation flow shop (PFS) scheduling problem with makespan objective in the static deterministic case (Framiñán et al. (2002)). LR can be seen as a counterpart for minimizing flow time. Finally, GILGOM is an exact algorithm for the special case of the no-wait two-machine flow shop problem minimizing makespan (Reddi and Ramamoorthy (1972)).

(1) Heuristic of Nawaz, Enscore & Ham (NEH) The heuristic of Nawaz et al. (1983) is originally formulated for the n-machine PFS scheduling problem of minimizing makespan. In a PFS, the determined job sequence before the first stage holds for each stage of the flow shop. Essentially, the algorithm follows the subsequent steps:

Step 1 Calculate the sum of processing times $p_j \ \forall \ j \in \{1, \ldots, n\}$ according to Equation (6.25).

Step 2 All jobs j are sorted in non-increasing order of p_j which results in a sequence of π_n positions.

Step 3 For the jobs in the first and second position, makespan values are determined for both feasible sequences, i.e., $1-2$ and $2-1$. The sequence with the lowest makespan is kept.

Step 4 The solution is iterated by adding a job to the partial sequence, calculating the makespan for every feasible partial schedule by keeping the relative positions of already scheduled jobs. The sequence with the lowest makespan is kept.

Step 5 Step 4 is repeated until the last job is sequenced with $\pi_j = n$.

(2) Heuristic of Liu & Reeves (LR) The algorithm of Liu and Reeves (2001) is formulated for flow time minimization in a PFS using composite solution techniques.

Step 1 Weighted machine idle time $IT_{jj'}$

Supposing that job j' is appended to a partial schedule π as successor of job j with $j \neq j'$ and $j, j' \in \{1, \ldots, n\}$, the idle time between machines l and $l-1$ is determined by Equation (6.27). Idle time is weighted with factor w_{lj}, which decreases with increasing l.

$$IT_{jj'} = \sum_{l=2}^{m} w_{lj} \max\{(C(l-1, j') - C(l, \pi), 0)\} \quad \forall \ j \neq j', \pi \in \{1, \ldots, n\} \tag{6.27}$$

Step 2 Artificial flow time $AT_{jj'}$

To take the unscheduled jobs of set U into account, an artificial flow time $AT_{jj'}$ is calculated if job j' follows j. An average processing time p of all unscheduled jobs of U is added to the completion time of job j' in (6.28).

$$AT_{jj'} = C(m, j') + C(m, p) \quad \forall\, j \neq j' \in \{1, \ldots, n\} \tag{6.28}$$

Step 3 Index function $f_{jj'}$

Index function $f_{jj'}$ contains the computed weighted machine idle time $IT_{jj'}$ and the artificial flow time $AT_{jj'}$ in (6.29). As artificial flow time evolves towards the actual processing time, an increasing number of jobs are scheduled. This number of jobs leads to a decreasing influence of idle time, which is displayed by the decreasing weight of $IT_{jj'}$. Job j' with the lowest index function value $f_{jj'}$ is appended to the schedule at position $j + 1$.

$$f_{jj'} = (n - j - 2)IT_{jj'} + AT_{jj'} \quad \forall\, j \neq j' \in \{1, \ldots, n\} \tag{6.29}$$

(3) Heuristic of Gilmore & Gomory (GILGOM) The algorithm of Gilmore and Gomory for makespan minimization is based on the TSP (Gilmore and Gomory (1964)). It is applicable in the single machine case, assuming that each job requires a different starting condition A_j on the machine to be processed and is completed in a certain state B_j. For job j' following j, the machine state has to be changed at a cost of $c_{jj'}$. According to Reddi and Ramamoorthy (1972), the algorithm can be transferred to a two-machine flow shop scheduling problem with no-wait constraint. $c_{jj'}$ is given by functions $f(x)$ and $g(x)$. $f(x)$ declares setup costs between B_j and $A_{j'}$, $g(x)$ between $A_{j'}$ and B_j correspondingly. Under the assumptions $f(x) = 1$ and $g(x) = 0$, A_j and B_j can be seen as processing time p_{1j} on the first and p_{2j} on the second machine. For the calculation of the changing costs $c_{jj'}$, two cases can be derived, shown in equations (6.30) and (6.31). In the first case, idle time on the second machine is generated by job j' following j (Equation (6.30)).

The case in Equation (6.31) leads to waiting time of job j' with $g(x) = 0$, which is not permitted. This can therefore be interpreted as a no-wait constraint.

$$c_{jj'} = \int_{C_{2j}}^{C_{1j'}} f(x)dx \quad if\ C_{1j'} \geq C_{2j} \quad \forall\ j \neq j' \in \{1, \ldots, n\} \tag{6.30}$$

$$c_{jj'} = \int_{C_{1j'}}^{C_{2j}} g(x)dx \quad if\ C_{2j} > C_{1j'} \quad \forall\ j \neq j' \in \{1, \ldots, n\} \tag{6.31}$$

6.4.2 Assignment

After the job sequence is generated, all jobs are scheduled on one machine per stage. Since the overall problem is proved to be $\mathcal{NP}$-hard for both layouts, we apply assignment rules typically applied to parallel machine scheduling problems, namely selecting the *first available machine* and the machine with *minimum idle time*. All of these approaches can be applied to both objectives, makespan and flow time minimization.

In the considered setting, the assignment only needs to be determined for the first stage with the parallel machines, since the second stage consists of exactly one machine.

(1) FAM - First Available Machine The job at position π is assigned to the machine that can start processing as soon as possible. The decision is based on completion time $C_{\pi-1}$ of the job at the previous position $\pi - 1$ on the corresponding machine, as shown in Framiñán et al. (2014).

(2) MIT - Minimum Idle Time Idle time of the machine at the second stage has an influence on both, the makespan and the flow time. The assignment using MIT rule is applied backwards considering the earliest possible start time $S_{1\pi}$ of the job at position π on stage one. If $S_{1\pi} > C_{1,\pi-1}$, job π is assigned to the machine that leads to least idle time on stage one. In case of $S_{1\pi} \leq C_{1,\pi-1}$, the first stage machine that causes the least idle time on stage two is chosen to harmonize utilization.

To illustrate the impact of these different assignment approaches, a small example is presented in Table 6.2. A test instance with $n = 10$ jobs is applied for $FH2(P2^{(1)}, 1^{(2)})|no-wait|C_{max}$ and $FH2(P2^{(1)}, 1^{(2)})|no-wait|F$.

Tab. 6.2: Processing times p_{kj} of an example instance with $n = 10$ jobs

stage k \ job j	1	2	3	4	5	6	7	8	9	10
1	46	80	64	40	68	60	18	56	176	34
2	22	1	54	8	48	52	53	25	5	85
$S_{NEH} = (S_4, S_1, S_7, S_8, S_6, S_5, S_3, S_{10}, S_9, S_2)$										

The job sequence $\Pi = (4, 1, 7, 8, 6, 5, 3, 10, 9, 2)$ of this instance is determined with the NEH heuristic (see (1) of Section 6.4.1). Applying approach NEH FAM, a makespan of 430 time units and flow time of 2385 time units are yielded, shown in Figure 6.4. With NEH MIT we receive $C_{max} = 393$ and $\sum C_j = 2311$ time units (see Figure 6.5). In Table 6.3 the objective values of each approach are summarized. We can see that NEH MIT leads to the best solutions. For C_{max}, they are optimal because no idle times occur on the machine in stage 2. In detail, if we compare the results of MIT and FAM in Figures 6.4 and 6.5, switching machine assignments for jobs 7, 8, 6, 5, and 3 eliminates idle time on stage two before job 9, which leads to the optimal schedule. Although these assignment changes result in an improved makespan, they do not have a proportionally noteworthy impact on flow time (makespan: $\sim$ 9 % improvement vs. flow time: $\sim$ 3 % improvement). As a result, a general advantage of one of the two assignment rules is not derivable based on this example.

Fig. 6.4: No-wait schedule applying approach NEH FAM

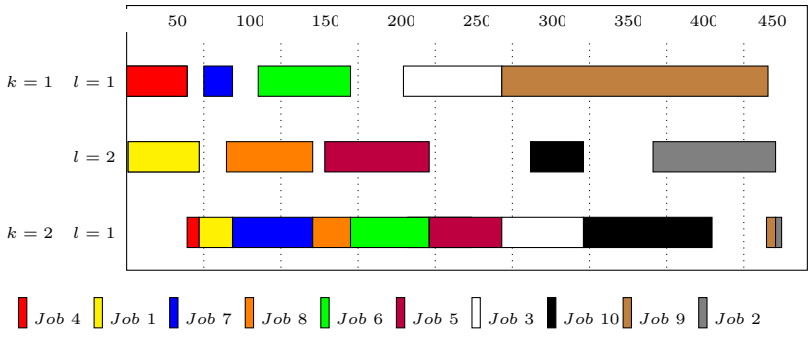
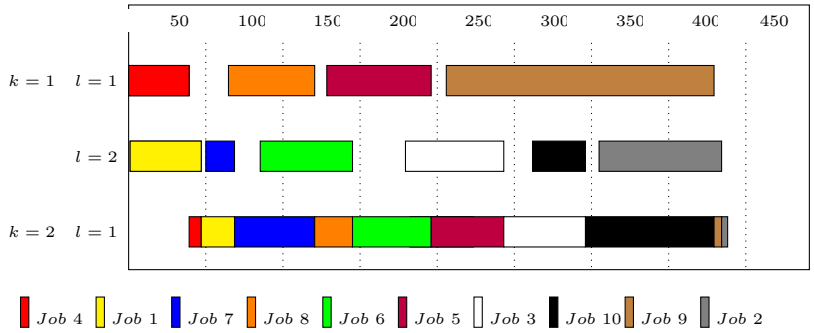


Fig. 6.5: No-wait schedule applying approach NEH MIT



Tab. 6.3: Results for C_{max} and F of example instance with $n = 10$ jobs

	makespan	deviation	flow time	deviation
NEH FAM	430	9.41 %	2385	3.20 %
NEH MIT	393	0.00 %	2311	0.00 %

6.4.3 Overview of the Solution Procedures

Combining all the presented assignment criteria, indicator values, and sequencing procedures, four generic approaches can be derived, i.e., FAM A1, FAM A2, MIT A1, and MIT A2, consisting of 16 and 19 approaches for the two objective functions C_{max} and F respectively (see Figure 6.6).

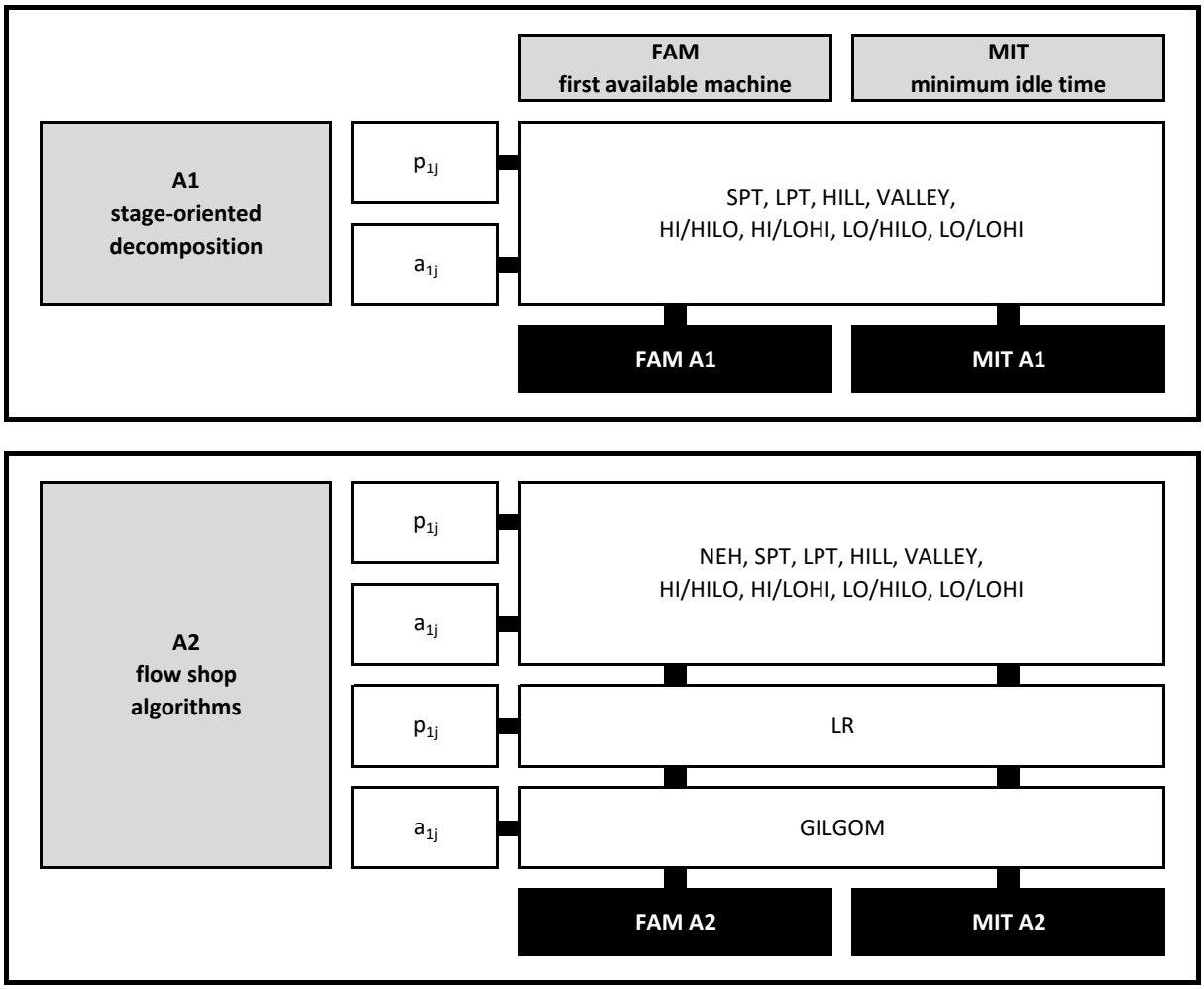
6.5 Computational Studies

6.5.1 Test Data and Computational Experiments

A suitable benchmark for HFS algorithms is presented by Pan et al. (2017). The data is provided on the SOA platform[4] based on the work of Vallada et al. (2015). The original testbed includes due dates and consists of several uniform stage and machine combinations for HFS systems. In this work, we apply the original processing times of the first machine of stage 1 to both machines of our layout because of the identical machines. Due to the higher capacity of the first stage – two machines in contrast to one machine on the second stage – the corresponding processing times are further multiplied

4 Sistemas de Optimización Aplicada, `http://soa.iti.es/problem-instances`

Fig. 6.6: Solution approaches to solving the two-stage no-wait HFS



by a factor of two to impede stage two from becoming an a priori bottleneck. For each job size class $n = (10, 15, 20, 50, 100, 150, 200)$, $T = 90$ instances are used, ensuring a normally distributed test data with $N(0, 1)$ (Bamberg and Baur (2002)). These 630 data sets are implemented for both objective functions, resulting in a total testbed of 1,260 instances.

All presented heuristic approaches (see Section 6.4) are implemented in MATLAB 2020b[5] and the formulated MILP model (see Section 6.3) in CPLEX 12.10.0[6], respectively. The computations run on Intel Core i3-8100 with 3.60 GHz, 8 GB RAM.

5 https://www.mathworks.com/products/matlab.html
6 https://www.ibm.com/analytics/cplex-optimizer

This chapter targets the statistical evaluation of the best heuristic procedures solving $FH2(P2^{(1)}, 1^{(2)})|no-wait|C_{max}$ and $FH2(P2^{(1)}, 1^{(2)})|no-wait|F$. A left-sided test for the differences of means of paired samples is carried out on a 5% level of significance α. Due to multiple testing on a testbed, a Bonferroni-Holm Correction is applied (Steland (2004)). For each tested hypothesis $i \in \{1, \ldots, k\}$, the local level of significance $\alpha_{local}(i)$ is calculated with $\frac{\alpha_{global}}{(k-i+1)}$. The hypotheses are sorted in increasing order of their corresponding probability values $p(i)$. Beginning with the first $p(i)$-value that is equal to or greater than the corresponding $\alpha_{local}(i)$, all remaining H0(i) have to be accepted. For each case i, H0(i) describes the hypothesis to be confirmed. This procedure is executed as a right-sided test. Due to the symmetry of the Gaussian distribution, the considered left-sided test requires complementary $p(i)$-values and the $\big(1-\alpha_{local}(i)\big)$ level of significance. A $p(i)$-value above $\big(1-\alpha_{local}(i)\big)$ leads to a rejection of H0(i). With an increasing number of hypotheses tested on a testbed, H0 is more likely to be accepted due to a decreasing $\alpha_{local}(i)$. Hence, the number of simultaneously tested procedures has to be kept at a reasonable level. The entire analysis consists of the following three main steps.

Step 1 All approaches are ranked according to their reached ARPD level (6.32), meaning the achieved *A*verage *R*elative *P*ercentage *D*eviation from the best known solution over all n. For each job class of instance t, the current result $X_{Heu}(t)$ is related to the best solution $X_{Min}(t)$. Furthermore, the MILP model is used as a benchmark to verify the effectiveness of the heuristic approaches.

$$ARPD = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{X_{Heu}(t) - X_{Min}(t)}{X_{Min}(t)} \cdot 100\right) \tag{6.32}$$

Step 2 The left-sided test is carried out separately for each objective function and each relevant heuristic for FAM A1, FAM A2, MIT A1, and MIT A2 (see Figure 6.6 for an overview of all approaches).

Step 3 Concentrating on the best heuristics identified in step two, the left-sided test is carried out again for each objective function.

6.5.2 Comparison of MILP Model to Heuristic Approaches

In the first step, the requirement to implement heuristics is reviewed by comparing their results with those of the MILP model. Since it is demanded to maintain solutions in a reasonable time, a runtime limit in CPLEX is determined of 600 seconds for $n = (10, 15, 20)$ and 1,800 seconds for $n = (50, 100, 150, 200)$ jobs. Considering all instances, MATLAB generates results in less than 600 seconds for each algorithm.

MILP vs. Heuristics: Results for Makespan

To compare the performance of all 17 or 20 approaches – 16 (approach 1) or 19 (approach 2) heuristics and the MILP – the mean ARPD over all $7 \cdot 90 = 630$ instances is generated for each approach solving makespan to create a ranking. Because we focus on whether or not the MILP compiles high quality solutions for small and large instances, only the mean ARPD of the best performing heuristic is compared to the optimization model in Table 6.4. Tables including ARPD values of the six best heuristics compared to the MILP model can be found in the appendix in tables B.1 and B.2.

Tab. 6.4: ARPD-values for C_{max}: MILP vs. best heuristic

makespan								
	FAM A1		MIN A1		FAM A2		MIN A2	
n	MILP	HILL	MILP	HILL	MILP	NEH	MILP	NEH
10	5.98	19.64	6.56	20.10	8.66	17.09	9.02	16.47
15	11.61	2.90	11.78	4.42	13.96	3.69	7.96	4.09
20	0.00	1.45	0.00	2.18	1.60	2.17	1.60	2.32
50	3.81	0.68	22.43	1.57	24.68	1.18	24.68	1.19
100		0.24	4.17	0.86	11.76	0.60	11.76	0.66
150		0.18		0.76		0.55		0.59
200		0.14		0.45		0.36		0.34
Mean	**8.53**	**6.30**	**8.99**	**4.34**	**12.13**	**3.66**	**11.01**	**3.66**
Rank	**3**	**1**	**3**	**1**	**8**	**1**	**7**	**1**

It must be considered that the MILP model does not generate feasible solutions for all 90 instances for $n = (10, 15, 20, 50, 100)$ and is not able to generate any feasible solution for $n = (150, 200)$. For $n = (20, 100)$ only one instance results in a feasible solution

and for $n = (15, 50)$ 16 and 4 solutions are compiled. Instead of replacing these cases of unfeasibility by an arbitrarily high number as a penalty, they are extracted, accepting falsification and improvement of MILP's mean ARPD by this method.

But even though the MILP's ARPD values are positively falsified, at least two heuristics outperform the optimization model on average over all n and solution approaches. The MILP model outperforms all heuristics only for $n = 10$, but performs worst in terms of runtime. Two or even seven algorithms applying A2 generate way better results for $n = (50, 100, 150, 200)$ with significantly better runtimes. Especially the fact that the MILP model does not compile any solution for $n = (150, 200)$ and only one for $n = (20, 100)$, demonstrates the need of applying heuristics with regard to practical applications.

MILP vs. Heuristics: Results for Flow Time

Focusing on flow time, the MILP performs better compared to $FH2(P2^{(1)}, 1^{(2)})|\text{no-wait}, prmu|C_{max}$ in the way that solutions are generated for all 630 instances. Again, the results of the optimization are compared with those of the best heuristic in Table 6.5. Tables with the six best heuristics can be found again in the appendix, see tables B.3 and B.4. The rank declares the position among all algorithms.

Tab. 6.5: ARPD-values for FT: MILP vs. best heuristic

flow time								
	FAM A1		MIN A1		FAM A2		MIN A2	
n	MILP	STT	MILP	STT	MILP	SPT	MILP	SPT
10	0.01	12.23	0.01	12.08	0.14	6.39	0.21	6.18
15	0.51	10.21	0.52	10.16	1.85	4.05	1.88	4.02
20	3.24	4.33	3.26	4.33	7.65	1.94	7.72	1.91
50	66.34	0.00	66.42	0.05	84.21	0.69	84.24	0.70
100	80.87	0.00	80.88	0.00	102.42	0.53	102.49	0.55
150	85.44	0.00	85.44	0.00	107.64	0.28	107.70	0.30
200	88.38	0.00	88.38	0.00	110.72	0.25	110.76	0.26
Mean	**46.40**	**3.82**	**46.42**	**3.80**	**59.23**	**2.02**	**59.29**	**1.99**
Rank	**15**	**1**	**17**	**1**	**19**	**1**	**20**	**1**

For small instances $n = (10, 15, 20)$, the MILP outperforms all heuristic approaches or compiles comparable solutions. But if we focus on large instances $n = (50, 100, 150, 200)$, it is easy to see that all heuristics partly perform better than the exact model (FAM A2, MIN A2). These results for minimizing flow time also demonstrate why heuristics are required to generate reliable solutions for different instances regarding their size and processing time relations.

6.5.3 Heuristics: Results for Makespan

In the following, we focus on the heuristics themselves. Since most approaches show high ARPD levels, only the six best and the worst are considered. The analysis is sorted by the solution approaches FAM, MIT, A1, A2 (see Figure 6.6) and the objectives makespan and flow time. To ensure a clear overview, not all discussed results are shown in this Section and Section 6.5.4, but the Tables showing all analyses are deposited in the appendix.

Step 1 and 2: FAM A1, FAM A2

In Table 6.6, the six best and the worst heuristics for FAM A1 and FAM A2 are ranked according to their mean ARPD values. Along with the respective algorithm, the implemented indicator value (iv: a_{1j}, p_{1j} – see descriptions in Section 6.4.1) is stated. In all cases, the dispatching rule HI/HILO performs worst, and none of the alternate dispatching rules reaches a reasonable ranking. Applying FAM as the assignment rule does not result in compensating for the right-shifting caused by the no-wait constraint despite having algorithms based on high differences between the indicator values of successive jobs. It is also easy to see that both indicator values of the absolute difference of jobs' processing times perform poorly. HILL and STT perform best for FAM A1 and are outperformed by the NEH heuristic within FAM A2.

Tab. 6.6: ARPD-values for the best six FAM C_{max} approaches

FAM A1							
				Best Six			Worst
	HILL	STT	LTT	VALLEY	LTT	STT	HI/HILO
$n\backslash iv$	p_{1j}	p_{1j}	p_{1j}	p_{1j}	a_{1j}	a_{1j}	p_{1j}
10	2.94	4.12	15.81	19.15	16.50	17.44	29.14
15	1.57	2.95	11.99	14.04	13.73	12.90	25.32
20	1.38	2.17	9.33	10.98	11.79	12.96	24.99
50	0.42	0.87	3.73	4.54	8.00	8.25	20.12
100	0.24	0.58	2.00	2.44	6.72	6.97	19.46
150	0.18	0.48	1.53	1.59	5.96	6.18	18.69
200	0.14	0.35	1.03	1.30	5.13	5.33	18.29
Mean	**0.98**	**1.65**	**6.49**	**7.72**	**9.69**	**10.00**	**22.29**
Rank	**1**	**2**	**3**	**4**	**5**	**6**	**16**
FAM A2							
				Best Six			Worst
	NEH	HILL	SPT	LR	GILGOM	LPT	HI/HILO
$n\backslash iv$	p_j	p_j	p_j	$f_{jj'}$	$c_{jj'}$	p_j	p_j
10	3.60	7.10	9.29	6.94	6.27	16.54	32.87
15	3.03	4.53	6.72	5.08	4.79	12.03	28.79
20	2.17	3.31	5.84	4.60	4.17	9.34	29.10
50	1.06	1.05	2.52	3.14	4.33	3.98	24.11
100	0.60	0.70	1.53	3.04	4.24	2.25	23.98
150	0.55	0.47	1.05	2.80	3.86	1.61	22.90
200	0.36	0.49	0.97	2.84	3.99	1.25	22.91
Mean	**1.62**	**2.52**	**3.99**	**4.06**	**4.52**	**6.72**	**26.38**
Rank	**1**	**2**	**3**	**4**	**5**	**6**	**19**

iv: indicator value

Based on the ranking in Table 6.6, three hypotheses $i = (1, 2, 3)$ are tested in Table 6.7 for FAM A1 verifying HILL(p_{1j}) and STT(p_{1j}) rules due to their lead in ARPD values compared to LTT(p_{1j}).

The results for FAM A1 are shown in Table 6.8. H declares the respective hypothesis in this and all subsequent tables dealing with the tests.

Tab. 6.7: Tested hypotheses for FAM A1 C_{max}

(1) H0(1): $\text{HILL}(p_{1j}) \geq \text{STT}(p_{1j})$ H1(1): $\text{HILL}(p_{1j}) < \text{STT}(p_{1j})$

(2) H0(2): $\text{STT}(p_{1j}) \geq \text{LTT}(p_{1j})$ H1(2): $\text{STT}(p_{1j}) < \text{LTT}(p_{1j})$

(3) H0(3): $\text{HILL}(p_{1j}) \geq \text{LTT}(p_{1j})$ H1(3): $\text{HILL}(p_{1j}) < \text{LTT}(p_{1j})$

Except for $n = 15$, all H0 hypotheses are rejected, confirming $\text{STT}(p_{1j})$ for $n = 15$ and $\text{HILL}(p_{1j})$ for all other n as the best procedures from the FAM A1 approach.

Tab. 6.8: Accepted hypotheses for FAM A1 C_{max}

FAM A1										
	$i = 1$			$i = 2$			$i = 3$			
n	p-value	$1 - \alpha_{local}$	H	p-value	$1 - \alpha_{local}$	H	p-value	$1 - \alpha_{local}$	H	best
10	0.9418	0.9833	H0	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})
15	0.9988	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**STT**(p_{1j})
20	0.9909	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})
50	0.9988	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})
100	0.9995	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})
150	1	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})
200	0.9997	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})

For FAM A2, the six best procedures are tested in Table 6.9, which are GILGOM, $\text{HILL}(p_j)$, $\text{LPT}(p_j)$, LR, NEH, and $\text{SPT}(p_j)$. As the relation of NEH, GILGOM and LR cannot be derived clearly from H0(1) to H0(6) for $n = (10, 15, 20)$, H0(7) and H0(8) are issued. The results are shown in Table 6.10. For $n = (15, 20, 50, 100, 150, 200)$, there is no significant difference between NEH and $\text{HILL}(p_j)$ confirming both as best but equivalent procedures for FAM A2. Only for $n = 10$ NEH emerges as the single best algorithm.

Tab. 6.9: Tested hypotheses for FAM A2 C_{max}

(1)	H0(1):	NEH $\geq$ HILL(p_j)	H1(1):	NEH $<$ HILL(p_j)
(2)	H0(2):	HILL(p_j) $\geq$ SPT(p_j)	H1(2):	HILL(p_j) $<$ SPT(p_j)
(3)	H0(3):	SPT(p_j) $\geq$ LR	H1(3):	SPT(p_j) $<$ LR
(4)	H0(4):	LR $\geq$ GILGOM	H1(4):	LR $<$ GILGOM
(5)	H0(5):	GILGOM $\geq$ LPT(p_j)	H1(5):	GILGOM $<$ LPT(p_j)
(6)	H0(6):	NEH $\geq$ LPT(p_j)	H1(6):	NEH $<$ LPT(p_j)
(7)	H0(7):	NEH $\geq$ GILGOM	H1(7):	NEH $<$ GILGOM
(8)	H0(8):	NEH $\geq$ LR	H1(8):	NEH $<$ LR

Tab. 6.10: Accepted hypotheses for FAM A2 C_{max}

makespan													
		$i=1$			$i=2$			$i=3$			$i=4$		
n	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	
10	0.9999	0.9833	H1	0.9962	0.9917	H1	0	0.9938	H0	0.1762	0.9929	H0	
15	0.9706	0.9917	H0	1	0.9833	H1	0.0002	0.9938	H0	0.2514	0.9929	H0	
20	0.9564	0.9917	H0	1	0.9833	H1	0.0003	0.9938	H0	0.2033	0.9929	H0	
50	0.4247	0.9929	H0	1	0.9875	H1	0.9955	0.9917	H1	1	0.9900	H1	
100	0.7019	0.9929	H0	1	0.9917	H1	1	0.9875	H1	1	0.9900	H1	
150	0.2358	0.9929	H0	1	0.9917	H1	1	0.9875	H1	1	0.9900	H1	
200	0.9115	0.9929	H0	1	0.9917	H1	1	0.9833	H1	1	0.9900	H1	

		$i=5$			$i=6$			$i=7$			$i=8$		
n	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	best
10	1	0.9500	H1	1	0.9750	H1	0.9986	0.9900	H1	0.9998	0.9875	H1	**NEH**
15	1	0.9500	H1	1	0.9750	H1	0.9906	0.9900	H1	0.9986	0.9875	H1	**NEH**, **HILL**(p_j)
20	1	0.9750	H1	1	0.9500	H1	0.9999	0.9900	H1	1	0.9875	H1	**NEH**, **HILL**(p_j)
50	0.1131	0.9938	H0	1	0.9500	H1	1	0.9750	H1	1	0.9833	H1	**NEH**, **HILL**(p_j)
100	0	0.9938	H0	1	0.9750	H1	1	0.9500	H1	1	0.9833	H1	**NEH**, **HILL**(p_j)
150	0	0.9938	H0	1	0.9833	H1	1	0.9500	H1	1	0.9750	H1	**NEH**, **HILL**(p_j)
200	0	0.9938	H0	1	0.9875	H1	1	0.9500	H1	1	0.9750	H1	**NEH**, **HILL**(p_j)

Step 1 and 2: MIT A1, MIT A2

Focusing on the results in Table 6.11 applying assignment rule MIT, we can summarize that the alternating dispatching rules and the indicator value based algorithms applying a_j and a_{1j} are outperformed by processing time oriented heuristics using p_j and p_{1j}.

Tab. 6.11: ARPD-values for the best six MIT C_{max} approaches

MIT A1							
	Best Six						Worst
	HILL	STT	HILL	STT	LO/LOHI	LTT	HI/HILO
$n\backslash iv$	p_{1j}	p_{1j}	a_{1j}	a_{1j}	p_{1j}	a_{1j}	a_{1j}
10	4.06	5.21	12.43	13.79	12.34	14.88	16.70
15	3.28	4.64	10.68	9.82	11.43	12.16	15.57
20	2.13	2.92	8.59	9.71	9.21	9.27	14.33
50	1.34	1.79	5.29	5.08	5.74	5.03	9.58
100	0.86	1.20	3.24	3.52	4.00	3.42	9.75
150	0.76	1.07	3.09	2.96	3.19	2.86	8.68
200	0.45	0.66	2.58	2.78	2.90	2.52	8.49
Mean	**1.84**	**2.50**	**6.56**	**6.81**	**6.97**	**7.16**	**11.87**
Rank	**1**	**2**	**3**	**4**	**5**	**6**	**16**
MIT A2							
	Best Six						Worst
	NEH	HILL	GILGOM	SPT	LR	LPT	HI/HILO
$n\backslash iv$	p_j	p_j	$c_{jj'}$	p_j	$f_{jj'}$	p_j	a_j
10	3.80	7.89	6.22	9.94	7.89	17.62	20.74
15	3.56	5.41	4.01	7.70	6.02	13.11	19.58
20	2.32	4.19	3.77	6.68	5.49	10.28	18.33
50	1.07	1.48	3.71	2.96	3.56	4.44	14.07
100	0.66	0.95	3.70	1.78	3.27	2.51	13.12
150	0.59	0.66	3.45	1.24	2.98	1.81	12.82
200	0.34	0.54	3.48	1.01	2.87	1.29	12.73
Mean	**1.76**	**3.02**	**4.05**	**4.47**	**4.58**	**7.30**	**15.91**
Rank	**1**	**2**	**3**	**4**	**5**	**6**	**19**

Similar to FAM A1, the MIT A1 approach indicates HILL(p_{1j}) and STT(p_{1j}) yielding best solutions according to their ARPD ranking. With HILL(a_{1j}) having a high difference from the leading rules, hypothesis tests are carried out on a $i = (1, 2, 3)$ testbed, which is illustrated in Table B.5 in the appendix. As a result of these tests, HILL(p_{1j}) can be considered as the best procedure within the MIT A1 approach for every n without indicating any significant difference to the FAM A1 results (see Table 6.12).

Tab. 6.12: Accepted hypotheses for MIT A1 C_{max}

MIT A1										
	$i = 1$			$i = 2$			$i = 3$			
n	p-value	$1 - \alpha_{local}$	H	p-value	$1 - \alpha_{local}$	H	p-value	$1 - \alpha_{local}$	H	best
10	0.9332	0.9833	H0	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})
15	0.9987	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**STT**(p_{1j})
20	0.9911	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})
50	0.9988	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})
100	0.9995	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})
150	1	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})
200	0.9997	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**HILL**(p_{1j})

Applying approach MIT A2, six hypotheses $i = (1, \ldots, 6)$ are formulated in Table B.6 of the appendix to test selected approaches based on their ARPD ranking in Table 6.11. Three additional hypotheses $i = (7, 8, 9)$ are added to clarify the significance of NEH compared to GILGOM, LR and SPT(p_j). Similar to FAM A2, NEH and HILL(p_j) provide best results for $n = (50, 100, 150, 200)$ regarding C_{max} (see Table 6.13). For $n = (10, 20)$, NEH is the single best algorithm, while NEH and GILGOM perform best for $n = 15$.

Tab. 6.13: Accepted hypotheses for MIT A2 C_{max}

makespan															
	$i=1$			$i=2$			$i=3$			$i=4$			$i=5$		
n	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H
10	1	0.9875	H1	0.0287	0.9929	H0	1	0.9833	H1	0.0007	0.9938	H0	1	0.9750	H1
15	0.9945	0.9900	H1	0.0071	0.9929	H0	1	0.9833	H1	0.0002	0.9938	H0	1	0.9750	H1
20	0.9992	0.9900	H1	0.2776	0.9929	H0	1	0.9833	H1	0.0006	0.9938	H0	1	0.9750	H1
50	0.9452	0.9917	H0	1	0.9833	H1	0.0089	0.9929	H0	0.9941	0.9900	H1	1	0.9875	H1
100	0.9625	0.9900	H0	1	0.9750	H1	0	0.9929	H0	1	0.9833	H1	0	0.9917	H0
150	0.7123	0.9900	H0	1	0.9833	H1	0	0.9929	H0	1	0.9750	H1	0	0.9917	H0
200	0.0217	0.9900	H0	1	0.9833	H1	0	0.9917	H0	1	0.9500	H1	0	0.9929	H0

	$i=6$			$i=7$			$i=8$			$i=9$			
n	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	best
10	1	0.9500	H1	0.9948	0.9917	H1	1	0.9900	H1				**NEH**
15	1	0.9500	H1	0.7088	0.9917	H0	0.9999	0.9875	H1				**NEH, GILGOM**
20	1	0.9500	H1	0.9959	0.9917	H1	1	0.9875	H1				**NEH**
50	1	0.9500	H1							1	0.9750	H1	**NEH, HILL(p_j)**
100	1	0.9500	H1							1	0.9875	H1	**NEH, HILL(p_j)**
150	1	0.9500	H1							1	0.9875	H1	**NEH, HILL(p_j)**
200	1	0.9750	H1							1	0.9875	H1	**NEH, HILL(p_j)**

Step 3: Validation of Best Approach for Makespan

For the minimization of makespan, it was shown statistically that the four heuristics HILL, SPT, NEH and GILGOM predominantly provide the best solutions within the approaches FAM A1, FAM A2, MIT A1 and MIT A2. With respect to the previous rankings, which include the six best approaches, VALLEY, LO/LOHI, LPT and LR are also acknowledged. Based on the preceding hypothesis testing, four hypotheses are considered in Table 6.14 to compare the best methods across FAM and MIT. In the case of more than one peak per test, only one rule has been chosen representatively for testing.

Tab. 6.14: Tested hypotheses comparing best FAM C_{max} and MIT C_{max} approaches

(1) H0(1): NEH MIT $\geq$ NEH FAM
H1(1): NEH MIT $<$ NEH FAM

(2) H0(2): NEH FAM $\geq$ HILL(p_{1j}) MIT
H1(2): NEH FAM $<$ HILL(p_{1j}) MIT

(3) H0(3): HILL(p_{1j}) MIT $\geq$ HILL(p_{1j}) FAM
H1(3): HILL(p_{1j}) MIT $<$ HILL(p_{1j}) FAM

(4) H0(4): NEH MIT $\geq$ HILL(p_{1j}) FAM
H1(4): NEH MIT $<$ HILL(p_{1j}) FAM

The results shown in Table 6.15 first confirm MIT as the more valid solution approach for $FH2(P2^{(1)}, 1^{(2)})|no-wait|C_{max}$ compared to FAM. Furthermore, NEH MIT is best for all instance sizes n, besides HILL(p_j) MIT for $n = (50, 100, 150, 200)$, and GILGOM MIT for $n = 15$.

In conclusion, sequencing algorithm NEH combined with assignment rule MIT promises the best performance for minimizing makespan for our testbed over all job sizes and approaches.

Tab. 6.15: Accepted hypotheses for C_{max}

makespan													
	$i=1$			$i=2$			$i=3$			$i=4$			
n	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	best
10	0.9995	0.9833	H1	1	0.9750	H1	0.9429	0.9875	H1	1	0.9500	H1	**NEH MIT**
15	0.9898	0.9833	H1	1	0.9750	H1	0.8413	0.9875	H1	1	0.9500	H1	**NEH MIT**, **GILGOM MIT**
20	0.9999	0.9833	H1	1	0.9750	H1	0.8925	0.9875	H1	1	0.9500	H1	**NEH MIT**
50	1	0.9833	H1	1	0.9750	H1	0.8413	0.9875	H1	1	0.9500	H1	**NEH MIT**, **HILL(p_j) MIT**
100	1	0.9833	H1	1	0.9750	H1	0.8413	0.9875	H1	1	0.9500	H1	**NEH MIT**, **HILL(p_j) MIT**
150	1	0.9833	H1	1	0.9750	H1	0.5000	0.9875	H1	1	0.9500	H1	**NEH MIT**, **HILL(p_j) MIT**
200	0.9990	0.9833	H1	1	0.9750	H1	0.5000	0.9875	H1	1	0.9500	H1	**NEH MIT**, **HILL(p_j) MIT**

6.5.4 Heuristics: Results for Flow Time

The same three steps of evaluating the results regarding makespan minimization are conducted for flow time objective as well.

Step 1 and 2: FAM A1, FAM A2

Similar to the results above for makespan, HI/HILO dispatching rule performs worst for both FAM A1 and FAM A2 approach combinations (see Table 6.16). STT and SPT rule outperform all other heuristics with an ARPD less than 1% for STT(p_{1j}) and SPT(p_j). Apart from better results achieved by the LR heuristic with an ARPD of 2.37%, there is a significant gap of at least 13% to all other algorithms.

Because of the clear result schemes, only three hypotheses $i = (1, 2, 3)$ are examined for FAM A1, dealing with STT(p_{1j}), HILL(p_{1j})and HILL(a_{1j}) (see Table B.7 in the appendix). The test results confirm that STT is dominant for this approach (see Table 6.17). Except for $n = 200$, every H0 is rejected, proving $STT(p_{1j})$ to be the best performing algorithm.

Tab. 6.16: ARPD-values for the best six FAM F approaches

FAM A1							
				Best Six			Worst
	STT	HILL	STT	LO/LOHI	VALLEY	LO/LOHI	HI/HILO
$n\backslash iv$	p_{1j}	p_{1j}	a_{1j}	p_{1j}	p_{1j}	a_{1j}	p_{1j}
10	0.68	11.34	29.63	29.11	43.37	32.08	61.93
15	0.45	10.47	24.02	30.63	34.87	28.18	55.19
20	0.41	14.99	25.75	29.98	34.25	30.86	60.53
50	0.00	14.35	18.30	25.33	22.12	25.77	51.04
100	0.00	14.72	16.68	24.70	18.85	25.25	50.43
150	0.00	13.93	15.49	23.71	17.09	24.48	48.28
200	0.00	14.54	14.80	23.55	16.70	25.36	47.84
Mean	**0.22**	**13.48**	**20.67**	**26.72**	**26.75**	**27.43**	**53.61**
Rank	**1**	**2**	**3**	**4**	**5**	**6**	**16**

FAM A2							
				Best Six			Worst
	SPT	LR	NEH	HILL	GILGOM	SPT	HI/HILO
$n\backslash iv$	p_j	$f_{jj'}$	p_j	p_j	$c_{jj'}$	a_j	p_j
10	1.68	2.45	18.85	18.46	22.95	38.18	68.08
15	1.25	2.38	16.86	17.71	23.25	34.48	63.11
20	1.24	2.25	16.46	19.97	24.45	34.71	68.27
50	0.69	2.25	12.99	20.66	26.78	30.47	62.35
100	0.53	2.58	11.05	22.31	28.21	29.63	63.13
150	0.28	2.29	10.94	21.76	26.28	28.95	60.48
200	0.25	2.42	9.31	21.96	26.55	28.65	60.35
Mean	**0.84**	**2.37**	**13.78**	**20.41**	**25.50**	**32.15**	**63.68**
Rank	**1**	**2**	**3**	**4**	**5**	**6**	**19**

Tab. 6.17: Accepted hypotheses for FAM A1 F

FAM A1										
	$i=1$			$i=2$			$i=3$			
n	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	best
10	1	0.9750	H1	1	0.9833	H1	1	0.9500	H1	**STT**(p_{1j})
15	1	0.9750	H1	1	0.9833	H1	1	0.9500	H1	**STT**(p_{1j})
20	1	0.9500	H1	1	0.9833	H1	1	0.9750	H1	**STT**(p_{1j})
50	1	0.9500	H1	1	0.9833	H1	1	0.9750	H1	**STT**(p_{1j})
100	1	0.9500	H1	1	0.9833	H1	1	0.9750	H1	**STT**(p_{1j})
150	1	0.9500	H1	1	0.9833	H1	1	0.9750	H1	**STT**(p_{1j})
200	1	0.9500	H1	0.7995	0.9833	H0	1	0.9750	H1	**STT**(p_{1j})

Analogously, three hypotheses $i = (1, 2, 3)$ comparing SPT(p_j), LR and NEH (see Table B.8 in the appendix) are tested for FAM A2. As expected, SPT(p_j) again performs significantly better in all but two cases where H0 cannot be rejected (see Table 6.18). In these cases with $n = (10, 20)$, LR heuristic yields equivalent results.

Tab. 6.18: Accepted hypotheses for FAM A2 F

FAM A2										
	$i=1$			$i=2$			$i=3$			
n	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	best
10	0.8186	0.9833	H0	1	0.9750	H1	1	0.9500	H1	**STT**(p_{1j}), **LR**
15	0.9881	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**STT**(p_{1j})
20	0.9591	0.9833	H0	1	0.9750	H1	1	0.9500	H1	**STT**(p_{1j}), **LR**
50	1	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**STT**(p_{1j})
100	1	0.9750	H1	1	0.9833	H1	1	0.9500	H1	**STT**(p_{1j})
150	1	0.9500	H1	1	0.9833	H1	1	0.9750	H1	**STT**(p_{1j})
200	1	0.9500	H1	1	0.9833	H0	1	0.9750	H1	**STT**(p_{1j})

Step 1 and 2: MIT A1, MIT A2

ARPD values for flow time criteria derived by MIT assignment differ in small ranges from the corresponding FAM results, but especially the top three heuristics are the same,

namely STT(p_{1j}), HILL(p_{1j}), and STT(a_{1j}) for A1 as well as SPT(p_j), LR($f_{jj'}$) and NEH(p_j) for A2 (see Table 6.19). The alternating rules perform slightly better under MIT, but are still outperformed by SPT and LR heuristics.

Tab. 6.19: ARPD-values for the best six MIT F approaches

MIT A1							
				Best Six			Worst
	STT	HILL	STT	LO/LOHI	LO/HILO	HILL	LTT
$n\backslash iv$	p_{1j}	p_{1j}	a_{1j}	p_{1j}	p_{1j}	a_{1j}	p_{1j}
10	1.00	11.70	27.43	23.48	35.13	29.25	50.95
15	0.68	10.73	21.84	24.74	22.51	25.78	44.00
20	0.46	15.04	23.42	22.97	28.81	27.34	46.21
50	0.05	14.41	16.29	18.15	19.66	21.08	34.91
100	0.00	14.72	14.47	16.84	17.96	18.53	33.18
150	0.00	13.93	13.59	15.73	16.50	17.80	31.37
200	0.00	14.54	13.39	15.84	16.13	17.37	30.80
Mean	**0.31**	**13.58**	**18.63**	**19.68**	**22.39**	**22.45**	**38.77**
Rank	**1**	**2**	**3**	**4**	**5**	**6**	**16**
MIT A2							
				Best Six			Worst
	SPT	LR	NEH	HILL	GILGOM	SPT	LPT
$n\backslash iv$	p_j	$f_{jj'}$	p_j	p_j	$c_{jj'}$	a_j	p_j
10	1.72	2.74	18.87	18.54	22.45	36.01	61.25
15	1.30	2.41	16.86	17.69	21.33	32.17	55.40
20	1.22	2.32	16.46	20.02	23.11	32.36	57.05
50	0.70	2.21	12.99	20.66	25.53	28.02	49.53
100	0.55	2.55	11.09	22.35	27.34	27.01	48.95
150	0.30	2.30	10.96	21.78	25.58	26.11	46.81
200	0.26	2.41	0.00	21.98	25.89	26.13	46.03
Mean	**0.86**	**2.42**	**12.46**	**20.43**	**24.46**	**29.69**	**52.15**
Rank	**1**	**2**	**3**	**4**	**5**	**6**	**19**

Comparable to Section 6.5.4, three hypotheses are tested. The results for MIT A1 and A2 are very similar to those for minimizing flow time with FAM A1 (Table 6.17) and

FAM A2 (Table 6.18) and can be found in tables B.9 and B.10 in the appendix. The slightly different p-values compared to FAM A1 F and FAM A2 F again imply STT(p_{1j}), SPT(p_j), and in two cases LR as best performing algorithms ($n = (10, 20)$ with MIT A2).

Step 3: Validation of Best Approach for Flow Time

Due to step two of the analysis, meaning the evaluation of the overall best approaches for flow time minimization, heuristics SPT, STT, and LR yield the best results followed by HILL, VALLEY, LO/LOHI, GILGOM and LO/HILO. Since SPT and STT are the dominant rules, the four hypotheses of Table 6.20 are tested. According to Table 6.21, no significant difference between FAM and MIT is derivable for $n = (15, 50, 100, 150, 200)$. Within the experiments performed, SPT(p_j) FAM and SPT(p_1) MIT as well as LR MIT for $n = (10, 20)$ outperform the other heuristics for $FH2(P2^{(1)}, 1^{(2)})|no - wait|F$.

Tab. 6.20: Tested hypotheses comparing the best FAM F and MIT F approaches

(1)	H0(1):	SPT(p_j) MIT $\geq$ SPT(p_j) FAM
	H1(1):	SPT(p_j) MIT $<$ SPT(p_j) FAM
(2)	H0(2):	SPT(p_j) FAM $\geq$ STT(p_{1j}) MIT
	H1(2):	SPT(p_j) FAM $<$ STT(p_{1j}) MIT
(3)	H0(3):	STT(p_{1j}) MIT $\geq$ STT(p_{1j}) FAM
	H1(3):	STT(p_{1j}) MIT $<$ STT(p_{1j}) FAM
(4)	H0(4):	SPT(p_j) MIT $\geq$ STT(p_{1j}) FAM
	H1(4):	SPT(p_j) MIT $<$ STT(p_{1j}) FAM

Tab. 6.21: Accepted hypotheses for F

flow time													
	$i=1$			$i=2$			$i=3$			$i=4$			
n	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	best
10	0.9994	0.9833	H1	1	0.9750	H1	0.9616	0.9875	H0	1	0.9500	H1	**SPT(p_j) MIT**, **LR MIT**
15	0.9671	0.9833	H1	1	0.9750	H1	0.9582	0.9875	H0	1	0.9500	H1	**SPT(p_j) MIT**, **SPT(p_j) FAM**
20	0.9878	0.9833	H1	1	0.9750	H1	0.8790	0.9875	H0	1	0.9500	H1	**SPT(p_j) MIT**, **LR MIT**
50	0.9162	0.9875	H0	1	0.9750	H1	0.9265	0.9833	H0	1	0.9500	H1	**SPT(p_j) MIT**, **SPT(p_j) FAM**
100	0.9279	0.9833	H0	1	0.9750	H1	0.8413	0.9875	H0	1	0.9500	H1	**SPT(p_j) MIT**, **SPT(p_j) FAM**
150	0.9778	0.9833	H0	1	0.9500	H1	0.5000	0.9875	H0	1	0.9750	H1	**SPT(p_j) MIT**, **SPT(p_j) FAM**
200	0.9608	0.9833	H0	1	0.9500	H1	0.8413	0.9875	H0	1	0.9750	H1	**SPT(p_j) MIT**, **SPT(p_j) FAM**

6.6 Conclusion and Outlook

In this chapter, a variety of heuristic approaches are applied and evaluated to solve the single objectives makespan and flow time (equivalent to total completion time) in the two-stage no-wait hybrid flow shop (HFS) scheduling problem, i.e., $FH2(P2^{(1)},1^{(2)})|no-wait|C_{max}$ and $FH2(P2^{(1)},1^{(2)})|no-wait|F=\sum_{\pi=1}^{n} C_{m\pi}$. Machine layouts considering no-wait constraint can be found in several industries dealing with sensitive material characteristics. This chapter is especially motivated by scheduling problems in steel industry where the production of sheets represents a typical no-wait HFS. Because of high industrial relevance of obtaining reliable results in a reasonable time, we concentrate on simple constructive heuristics that generate solutions within a short runtime. Since the evenly addressed layouts with two stages and two machines on the first stage are even $\mathcal{NP}$-hard, 102 approaches with different combinations of sequencing and assignment rules as well as six indicators are tested on a testbed with a total of 1,230 instances. Additionally, the results of all heuristic algorithms are set into relationship to a benchmark generated by a mixed integer linear programming model (MILP).

First, it can be shown that the MILP only generates results in reasonable time for up to $n=20$ jobs for both makespan and flow time objectives. Instances with 50 or more

jobs are not able to be solved properly despite significantly longer runtimes compared to all heuristic approaches.

For minimizing makespan, treating the HFS as separate flow shops (approach A2) and applying the algorithm of Nawaz et al. (1983) (NEH) performs best for all instances in combination with sequencing jobs using the assignment rule minimum idle time (MIT), which is synonymous with selecting the machine in consideration of minimizing idle time. Minimizing flow time, A2 also performs best using Shortest Processing Time (SPT) rule for sequencing combined with either first available machine rule (FAM) or MIT to assign jobs to machines. These results are expected because approach A2 allows to exploit all the findings identified for flow shop problems (Framiñán et al. (2002)), e.g. the good performance of SPT rule for specific layouts (Haupt (1989)).

In summary, the experiments show that simple dispatching rules are able to achieve equivalent (HILL vs. NEH) or even better (SPT vs. LR) results compared to constructive heuristics suited to a specific optimization problem. Decomposition approach A1 always performs worse than A2 in the observed testbed. If the sequencing problem is solved separately for each stage (A1), no-wait constraint would be violated along with the sequence-dependent connection between both processing stages. This conflict results in right-shifting of all jobs from the first stage to satisfy the no-wait constraint. Led by the machine environment with identical parallel machines on the first stage, approach A2, which assumes multiple flow shops instead of a single HFS, performs best for all testbeds. Concerning indicator values p_{1j} or p_j and a_{1j} or a_j, the predominance of the processing time related indicators can be shown. Assigning the jobs according to the least idle time caused on the second processing stage (MIT) works best for makespan, but has no significant influence on flow time as a sum-form objective. Whether this result can be generalized, or whether it is caused by the relatively simple problem description, has to be tested in further work.

Finally, it can be seen in this work that dispatching rules provide fast and high-quality solutions for the two-stage no-wait HFS, which is known to be $\mathcal{NP}$-hard and displays several production cases. In part, even established heuristic approaches are outperformed for the observed testbed. Solutions generated by MILP models come with high runtimes and worse solutions for instances with more than $n = 20$ jobs.

Although the good performance of dispatching rules is shown, there is still room for improvement. Besides the datasets from the SOA platform (Pan et al. (2017)) that provide high quality instances, a more recent testbed for HFS layouts with identical machines presented by Fernandez-Viagas and Framiñán (2020) could be applied. In addition, the above approaches should be tested with data from real-world shop floors. Furthermore, all statements are based on tests comparing the results with each other. In a next step, these results should be taken up and compared with the results of other heuristic approaches presented in the literature review (e.g., Wang et al. (2019)) as well as to suitable lower bounds. In general, the influence of typical assumed parameters of the testbed on the approaches should be evaluated, especially the range of processing times and the normal distribution. Maassen and Pérez-González (2020) examine these characteristics for flow shop system related instances. For special real-world production systems, more complex HFS layouts can also be implemented to review affected statements. In future work, the no-wait constraint could be relaxed by allowing limited waiting times and observe their influence on idle and total waiting times. Production lines could be made more flexible, allowing some jobs to be shifted to other time windows by exploiting this idea. First approaches can be found in Botta-Genoulaz (2000), Attar et al. (2014), Schulze et al. (2016), Gicquel et al. (2012) or Tan et al. (2018). In this context, optimizing energy costs or energy consumption in general could be a future-oriented goal, as focused in Ding et al. (2021). Referred to sheet production, a predefined temperature gradient of sheets could be acceptable between heating of the steel and its processing in rolling mills, or sheets could be stored in the furnace for a limited time to increase flexibility in the process.

Chapter 7

Constructive Heuristics in Hybrid Flow Shop Scheduling with Unrelated Machines and Setup Times

As shown in Chapter 6, hybrid flow shop (HFS) systems represent the typical flow shop production system with parallel machines on at least one stage of operation. In this Chapter, which is from Hipp and Geldermann (2020), unrelated machines and anticipatory sequence-dependent setup times are considered, where job families can be formed based on similar setup characteristics. This results in the opportunity of saving setups when two jobs of the same family are scheduled consecutively. Based on the algorithm of Nawaz et al. (1983) (NEH), three constructive heuristic approaches are implemented that aim at minimizing makespan, total completion time, and the total number of setup procedures.

7.1 Introduction

Flow shop scheduling with parallel machines on at least one stage, i.e., the hybrid flow shop (HFS), poses a complex combinational problem commonly found in industries such as semi-conductor production or chemical industry. In the steel industry, e.g., the processing of sheets can be modeled as a HFS with setup times. For instance, steel sheets with different characteristics which are processed in the same system lead to different job sequence-dependent machine setups. Furthermore, the processing times of sheets with

similar characteristics can differ on each machine. In this chapter, we focus on such a HFS with unrelated machines and anticipatory sequence-dependent setup times. Job families can be formed based on similar setup characteristics which provides the opportunity to save setup procedures when two jobs of the same family are scheduled subsequently. By taking into account anticipatory setups, it is possible to start the setup procedure of a machine as soon as the decision is taken which machine processes which job. Compared to non-anticipatory setups, the start of the setup procedure can take place before the respective job reaches the machine. According to the three field notation of Graham et al. (1979), the models under consideration can be classified as $FHm(RM^k)_{k=1}^m|ST_{sd,f}|C_{max}$, $FHm(RM^k)_{k=1}^m|ST_{sd,f}|F)$, and $FHm(RM^k)_{k=1}^m|ST_{sd,f}|ST$. On each stage k of the m-stage HFS system (FHm) a set RM k of non-identical machines is available. Anticipatory job sequence-dependent setup times ($ST_{sd,f}$) including different job families f are assumed. The objective is either the minimization of makespan (C_{max}), flow time (F) or setups (ST). As jobs are assumed to be available at time zero, flow time minimization equals minimizing the sum of jobs completion times at the last stage. The HFS scheduling problem including unrelated machines is proven to be $\mathcal{NP}$-hard for minimizing C_{max} (Lee and Vairaktarakis (1994)). Compared to C_{max}, F is a more complex objective function, and even the two-stage flow shop problem is known to be $\mathcal{NP}$-hard (Graham et al. (1979)). The problem $FHm(RM^k)_{k=1}^m|ST_{sd,f}|\gamma$ also includes the special case of setup times equal to zero, so $\mathcal{NP}$-hardness can be assumed. This makes it necessary to develop approximation algorithms to solve realistic problem instances. In this chapter, a HFS is modeled with unrelated machines, setup times and job families. In contrast to the current literature, no batch sizes are predefined in connection with job families. The well-performing MCH heuristic of Fernandez-Viagas et al. (2018) for HFS problems with identical machines, which is based on the algorithm of Nawaz et al. (1983) (NEH), is modified. The influence of the total number of setups and setup times is examined with regard to the performance measures. Batch sizes are not fixed to not restrict the performance of the heuristics in advance.

In Section 7.2, a brief literature review is carried out. After that, the characteristics of the examined models are presented and the heuristics based on the MCH heuristic of Fernandez-Viagas et al. (2018) are proposed in Section 7.3.

Finally, the results are presented in Section 7.4 before the chapter is summarized in Section 7.5.

7.2 Literature Review

The last comprehensive reviews for HFS literature are from 2010 and presented by Ruiz and Vázquez-Rodríguez (2010) as well as Ribas et al. (2010) which classify papers according to their covered constraints and objective functions. In recent studies, NEH based algorithms (Nawaz et al. (1983)) have been implemented among other solution approaches like meta-heuristics. NEH provides high quality solutions for flow shop scheduling problems, but has to be modified for HFS problems. For representative mixed integer linear programming models (MILP) in HFS scheduling with unrelated machines, see those of Naderi et al. (2014a) and Ruiz et al. (2008). A detailed survey on scheduling problems with setup times distinguishing between production and setup type is given by Allahverdi (2015). Focused on NEH-based procedures, Jungwattanakit et al. (2005) use several algorithms like NEH to generate high quality initial solutions for solving a dual criteria HFS with unrelated machines and setup times which are afterwards improved by meta-heuristics. A complex m-stage HFS is solved by Ruiz et al. (2008) including several constraints, inter alia setup times, with the objective of flow time minimization. In addition to a MILP, dispatching rules and NEH algorithm are implemented and compared. Simulated annealing is implemented by Naderi et al. (2009) to deal with a similar production layout to minimize flow time and by Mirsanei et al. (2011) for makespan minimization. Setup times with job families in HFS scheduling are considered by Ebrahimi et al. (2014). Shahvari and Logendran (2018) focus on batch scheduling to deal with setup times, job families, and simultaneous minimization of weighted flow time and weighted tardiness including a population based learning effect.

These studies focus either on setup times without family characteristics or on group technology with fixed batch sizes (Ribas et al. (2010)). Therefore, constructive heuristics are formulated for the case of free batching without any fixed setting in this chapter.

7.3 Heuristics for Hybrid Flow Shops with Unrelated Machines and Setup Times

The HFS scheduling problem $FHm(RM^k)_{k=1}^m|ST_{sd,f}|\gamma$ is examined with the objectives of minimizing makespan C_{max}, flow time F, or the number of setups ST. Setup times $ST_{sd,f}$ are considered which can be executed in advance (anticipatory) and job families classified by similar setup characteristics are given. In the following, two essential specifics, namely unrelated machines and setup times, are described in detail.

When scheduling jobs in the HFS problem with unrelated machines in contrast to identical ones, not only the sequencing of jobs is relevant, but also their assignment to a specific machine i per stage k. This results in job processing times p_{kij} not only referring to stage k and job j but also to machine i. Following Rashidi et al. (2010), machine speed v_{kij} is considered to relate a norm processing time p_{kj} for each job and each stage to each machine i, so that either instances for HFS with identical and unrelated machines can be adopted.

Regarding setup times st_{kfg}, sequence-dependence and anticipation of setups have to be distinguished. Sequence-dependence represents the influence of the job sequence on the total number of setup procedures required. For different job families f, it is possible to schedule subsequent jobs of the same family f to save setups. Only subsequent jobs of different job families f and g cause setup procedures. Because of anticipatory setups, the setup procedure for job j on a machine at stage k can already start as soon as the operation of job j starts on a machine at stage $k-1$. Thus, potential idle time in front of the machine at stage k can be used to execute the needed setup procedure, decreasing the waiting time of job j in front of this machine. In total, three constructive heuristics are formulated, H1, H2 and H3, one for each objective function ST, C_{max}, and F. The memory-based constructive heuristic (MCH) of Fernandez-Viagas et al. (2018), which provides high quality solutions for HFS problems with identical machines is modified for the mentioned problem at hand. As in the NEH algorithm, the job sequence is built by insertion. In each iteration, one job is added to the subsequence at each possible position of the sequence. In each iteration, notional completion times, including setup times, are calculated by summing average values for jobs' processing times per each stage and all jobs

Tab. 7.1: Overview of examined benchmark

Parameter	**Small instances**	**Large instances**
Number of jobs n	5, 7, 9, 11, 13, 15	50, 100
Processing stages m	2, 3	4, 8
Number of machines per stage m_k	1, 3	2, 4
Processing times p_{kij}	U[1,99]	U[1,99]
Setup times s_{kgf}	U[25,75]	U[75,125]
Number of setup categories $\mathcal{F}$	0, 2, 4	0, 20, 40

which are not sequenced so far. In addition, a memory list saves the sequences and values of the former iteration and compares them with the current ones in order to determine the best fitting sub-sequence in the current iteration. The jobs are assigned to a production stage follows the earliest completion time rule (ECT) (Fernandez-Viagas et al. (2018)).

7.4 Computational Study

All subsequent calculations are coded in MATLAB R2016a on an Intel Pentium 3805U with 1.9 GHz, 4 GB RAM. The examined benchmarks in Table 7.1 are provided by the Spanish research group Sistemas de Optimización Aplicata (SOA) for HFS with unrelated machines and non-family setup times and are adapted for this chapter by additionally generating values for setup categories. 144 combinations of small instances are generated and 48 combinations of large ones, each five times so that 960 instances are generated in total.

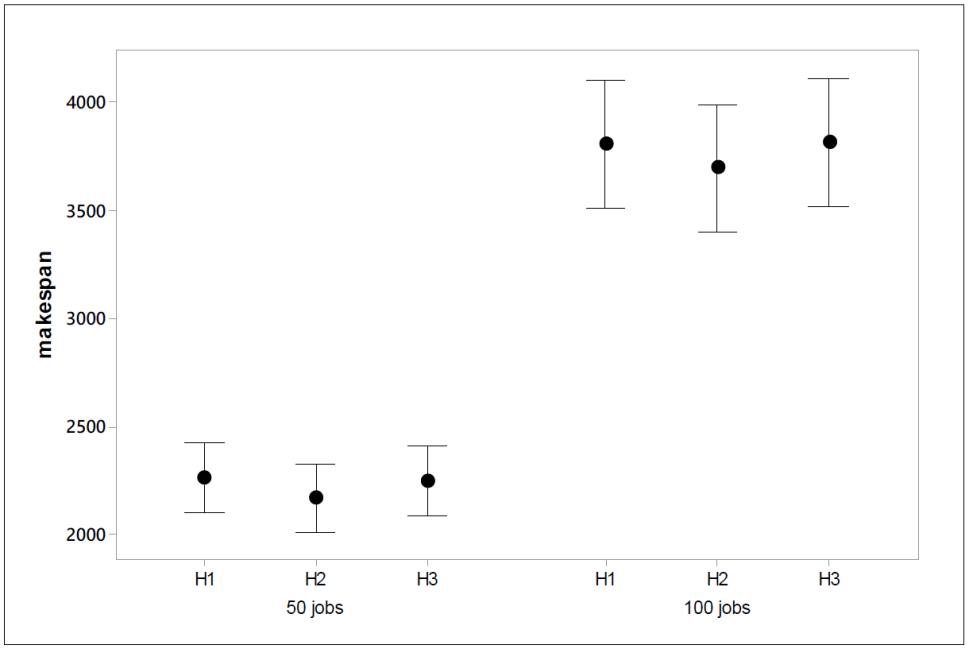
Because small and large instances show similar performance, only the results for large instances are shown in the following. Table 7.2 shows the total number of setup procedures according to family sizes $\mathcal{F}$ over all numbers of jobs n. Heuristic H1 which pursues the minimization of setups (ST) provides the lowest number of setup procedures for large instances. Heuristic H2 minimizing C_{max}, and heuristic H3 minimizing F provide values for setup procedures in a similar range. Compared to H1, the total number of setups given by H2 and H3 increase around 100 percent or 45 percent (H1 vs. H2: 120 vs. 256.8; 240 vs. 345.5).

Tab. 7.2: Number of setups for large instances

		H1	*ST*	**H2**	C_{max}	**H3**	*F*
stage m	**family** f / m_k	20	40	20	40	20	40
4	2	80.0	160.0	149.4	215.9	153.3	223.8
	4	80.0	160.0	186.9	240.6	193.3	243.9
8	2	160.0	320.0	305.4	440.3	302.4	449.1
	4	160.0	320.0	385.3	485.1	388.9	489.9
	average	120.0	240.0	256.8	345.5	259.5	351.7

Figure 7.1 shows the range of makespan values provided by all three heuristics H1, H2 and H3 for 50 and 100 jobs over all job families $\mathcal{F}$ (the same scheme is valid for flow time). Even though only heuristic H2 targets makespan minimization, the values for makespan of all heuristics are in a similar range. Consequently, the high increase of setup procedures seen in Table 7.2 between heuristic H1 and H2 respectively H3 along with increasing setup times does not automatically result in higher jobs' completion times. This can be explained by examining the idle times in the schedules. Because setups can be executed before the jobs enter the machine (character of anticipatory setups), the higher total number of setup procedures does not necessarily impact the performance negatively. The setups can be executed in the idle times of the machines so that the higher number of setups in the schedules given by heuristics H2 and H3 do not result in higher completion times of jobs.

Fig. 7.1: Range of solutions for makespan for large instances



7.5 Conclusion

In the literature, HFS scheduling with setup times and job families is typically combined with fixed batch sizes. In this work, the influence of setups on job completion time related performance measures is examined by considering the case of free batching. Three constructive heuristics based on the NEH algorithm (Fernandez-Viagas et al. (2018)) are compiled to solve the m-stage HFS with unrelated machines and anticipatory sequence-dependent setup times to minimize makespan, flow time, and the total number of setups. In addition, job families are defined based on setup characteristics. The implemented heuristics, each for one objective function, are applied on a testbed of 960 instances in total. It can be shown exemplary that an increasing number of setups does not necessarily result in increasing makespan and flow time when anticipatory setups are considered. In future work, the performance of the applied heuristics should be evaluated by comparing them with other approximation algorithms as well as with machine assignment rules other than Earliest Completion Time (ECT).

Chapter 8

Summary and Outlook

This thesis investigates machine scheduling problems with non time-based machine availability constraints, which result in the need to perform maintenance operations to secure the machine state for the execution of jobs. Since maintenance planning is then mainly influenced by the number of scheduled jobs on the machine, we call this type of constraint position-dependent (*pd*) maintenance. In this context, machines reflect any type of renewable resources and jobs reflect any type of tasks that require these resources to be processed. We embed these machine scheduling problems, including *pd* maintenance planning, in the research field of scheduling with non-continuous machine availability, and analyze single machine problems with *pd* maintenance and related problems regarding their properties and runtime complexity. Furthermore, exact solution algorithms are presented for specific single machine problems, including analysis of their runtime.

The first part of this work focuses on the basics of machine scheduling and maintenance planning. Chapter 1 gives an introduction to scheduling as a part of Operations Research and its general functions and characteristics. In Chapter 2, we present the required notation, based on the three-field notation of Graham et al. (1979), which is applied to deal with the scheduling problems in the remaining thesis. Furthermore, we introduce an extension of the Graham notation developed by Briskorn et al. (2024) to properly describe the model restrictions that correlate with *pd* maintenance.

In Chapter 3, a detailed survey based on the work of Briskorn et al. (2024) is outlined to classify our studies in the scope of maintenance scheduling research with non-time based factors. The formulated extension of the Graham notation is applied to differentiate

between several relevant factors, such as characteristics regarding the deterioration of jobs on machines or whether the processing times of jobs are influenced by the machine state. To the best of our knowledge, a classification scheme for scheduling models with availability constraints and maintenance planning that are not based on time-related factors has not been elaborated so far.

In the second part, in Chapter 4, we concentrate on single machine scheduling problems with *pd* maintenance. After presenting runtime complexity results for several problems that can be derived from the corresponding scheduling problems with continuous machine availability, two dynamic programs are developed to solve the single machine problems with *pd* maintenance, uniform processing times and release dates to optimize total weighted completion time and the number of tardy jobs in polynomial time, respectively. The approach of both algorithms is illustrated, and it is shown that the dynamic problem of minimizing total weighted completion time can also be applied to optimize the same layout with the objective of minimizing total tardiness. Table 4.12 in Section 4.2.6, together with the findings of Section 4.3, summarizes the runtime complexity results of the studied layouts. Chapter 5 looks at the problem class of problems with fixed positioned (fp) jobs. Here, special jobs, as a subset of the total job set, have to be sequenced on specific positions in the final sequence of the final schedule. As already accomplished in Chapter 4, single machine scheduling layouts are studied regarding their runtime complexity, the results of which can be derived from the respective problems without considering special jobs (see Table 5.6 in Section 5.5). In the second step, a dynamic program and its specifics are presented to solve the single machine scheduling problem with fp jobs and uniform due dates to minimize total weighted late work to optimality in pseudo-polynomial time. The algorithm is oriented on the Knapsack Problem (Kellerer et al. (2004)) and modified for our problem.

Chapters 6 and 7 form the final part of this thesis with problems related to scheduling models with *pd* maintenance. In Chapter 6, we address hybrid flow shop scheduling problems with the no-wait constraint. Since this problem is $\mathcal{NP}$-hard even for the two-stage machine setting, priority rules and heuristic algorithms based on parallel machine and flow shop approaches are implemented and analyzed, with the priority rules performing equally or better than the heuristics for all settings. Another type of machine unavailabil-

ity caused by job characteristics is shown in Chapter 7 by introducing job class-related setup times. Three constructive heuristics based on the algorithm of Fernandez-Viagas et al. (2018) are generated to separately optimize the objective functions of minimizing makespan, total completion time, and the total number of setup procedures. The results are evaluated with respect to the influence of the number of setups on the respective performance.

In summary, it can be stated that the field of position-dependent availability still offers a wide range of research opportunities. Besides the still open single machine problems and the other machine environments that have not yet been considered to the best of our knowledge, efficient algorithms might also be required to be developed for the problems that have already been studied. In addition, the examples of related job constraints, such as the consideration of jobs with fixed positions in the final sequence, demonstrate that there are closely related properties in the area of position-dependent availability to *pd* maintenance planning that open up further fields of research.

Bibliography

Agnetis, A. and Mosheiov, G. (2017). Scheduling with job-rejection and position-dependent processing times on proportionate flowshops. *Optimization Letters*, 11:885–892.

Albers, S., Müller, F., and Schmelzer, S. (2014). Speed scaling on parallel processors. *Algorithmica*, 68(2):404–425.

Alfaro-Fernández, P., Ruiz, R., Pagnozzi, F., and Stützle, T. (2020). Automatic algorithm design for hybrid flowshop scheduling problems. *European Journal of Operational Research*, 282(3):835–845.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378.

Allahverdi, A. (2016). A survey of scheduling problems with no-wait in process. *European Journal of Operational Research*, 255:665–686.

Attar, S., Mohammadi, M., Tavakkoli-Moghaddam, R., and Yaghoubi, S. (2014). Solving a new multi-objective hybrid flexible flowshop problem with limited waiting times and machine-sequence-dependent set-up time constraints. *International Journal of Computer Integrated Manufacturing*, 27(5):450–469.

Bamberg, G. and Baur, F. (2002). *Statistik*. Oldenbourg, Wien, 12 edition.

Bampis, E., Kononov, A., Letsios, D., Lucarelli, G., and Nemparis, I. (2015). From preemptive to non-preemptive speed-scaling scheduling. *Discrete Applied Mathematics*, 181:11–20.

Baptiste, P. (1999). Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling*, 2(6):245–252.

Baptiste, P. (2000). Scheduling equal-length jobs on identical parallel machines. *Discrete Applied Mathematics*, 103(1-3):21–32.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Biskup, D. (1999). Single-machine scheduling with learning considerations. *European Journal of Operational Research*, 115(1):173–178.

Błażewicz, J., Dell'Olmo, P., Drozdowski, M., and Mączka, P. (2003). Scheduling multiprocessor tasks on parallel processors with limited availability. *European Journal of Operational Research*, 149(2):377–389.

Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., and Weglarz, J. (2019). *Handbook on Scheduling*. Springer.

Bock, S., Briskorn, D., and Horbach, A. (2012). Scheduling flexible maintenance activities subject to job-dependent machine deterioration. *Journal of Scheduling*, 15:565–578.

Botta-Genoulaz, V. (2000). Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. *International Journal of Production Economics*, 64(1-3):101–111.

Brah, S. A. and Loo, L. L. (1999). Heuristics for scheduling in a flow shop with multiple processors. *European Journal of Operational Research*, 113:113–122.

Briskorn, D., Choi, B.-C., Lee, K., Leung, J., and Pinedo, M. (2010). Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, 207(2):605–619.

Briskorn, D., Hipp, A., Jaehn, F., and Thiemeyer, A. (2024). State dependent machine availability in scheduling. *Working paper*.

Briskorn, D., Jaehn, F., and Pesch, E. (2013). Exact algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, 16:105–115.

Briskorn, D. and Leung, J. Y. (2013). Minimizing maximum lateness of jobs in inventory constrained scheduling. *Journal of the Operational Research Society*, 64(12):1851–1864.

Brucker, P. and Knust, S. (2012). *Complex Scheduling*. Springer.

Calinescu, G., Jaehn, F., Li, M., and Wang, K. (2017). An fptas of minimizing total weighted completion time on single machine with position constraint. *The 28th International Symposium on Algorithms and Computation (ISAAC)*.

Campbell, H. G., Dudek, R. A., and Smith, M. L. (1970). A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16(10):B–630.

Carpov, S., Carlier, J., Nace, D., and Sirdey, R. (2012). Two-stage hybrid flow shop with precedence constraints and parallel machines at second stage. *Computers & Operations Research*, 39(3):736–745.

Chang, J., Yan, W., and Shao, H. (2004). Scheduling a two-stage no-wait hybrid flowshop with separated setup and removal times. *Proceeding of the American Control Conference Boston, Massachusetts, June 30-July 2*, 2:1412–1416.

Chekuri, C. and Khanna, S. (2001). A ptas for minimizing weighted completion time on uniformly related machines. In *Automata, Languages and Programming: 28th International Colloquium, ICALP 2001 Crete, Greece, July 8–12, 2001 Proceedings 28*, pages 848–861. Springer.

Cheng, M., Xiao, S., Luo, R., and Lian, Z. (2018). Single-machine scheduling problems with a batch-dependent aging effect and variable maintenance activities. *International Journal of Production Research*, 56(23):7051–7063.

Cheng, T., Lee, W.-C., and Wu, C.-C. (2011). Single-machine scheduling with deteriorating jobs and past-sequence-dependent setup times. *Applied Mathematical Modelling*, 35(4):1861–1867.

Cheng, T. E., Ding, Q., and Lin, B. M. (2004). A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research*, 152(1):1–13.

Cheriyan, J., Ravi, R., and Skutella, M. (2021). A simple proof of the moore-hodgson algorithm for minimizing the number of late jobs. *Operations Research Letters*, 49(6):842–843.

Chrobak, M., Durr, C., Jawor, W., Kowalik, L., and Kurowski, M. (2004). A note on scheduling equal-length jobs to maximize throughput. *arXiv preprint cs/0410046*.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2022). *Introduction to algorithms*. MIT press.

Costa, A., Cappadonna, F. A., and Fichera, S. (2016). Minimizing the total completion time on a parallel machine system with tool changes. *Computers & Industrial Engineering*, 91:290–301.

Crompton, T. R. (2000). *Battery reference book*. Newnes.

Davari, M., Ranjbar, M., De Causmaecker, P., and Leus, R. (2020). Minimizing makespan on a single machine with release dates and inventory constraints. *European Journal of Operational Research*, 286(1):115–128.

Demeulemeester, E. L. and Herroelen, W. S. (2006). *Project scheduling: a research handbook*, volume 49. Springer Science & Business Media.

DIN (2015). DIN EN ISO 9000. `https://www.din.de/de/wdc-beuth:din21:235671064` [Accessed: 2023-12-08].

Ding, J., Schulz, S., Shen, L., Buscher, U., and Lü, Z. (2021). Energy aware scheduling in flexible flow shops with hybrid particle swarm optimization. *Computers & Operations Research*, 125:105088.

Dolgui, A., Gordon, V., and Strusevich, V. (2012). Single machine scheduling with precedence constraints and positionally dependent processing times. *Computers & Operations Research*, 39(6):1218–1224.

Drozdowski, M., Jaehn, F., and Paszkowski, R. (2017). Scheduling position-dependent maintenance operations. *Operations Research*, 65(6):1657–1677.

Du, J. and Leung, J. Y.-T. (1990). Minimizing total tardiness on one machine is np-hard. *Mathematics of Operations Research*, 15(3):483–495.

Ebrahimi, M., Ghomi, S. F., and Karimi, B. (2014). Hybrid flow shop scheduling with sequence dependent family setup time and uncertain due dates. *Applied Mathematical Modelling*, 38(9-10):2490–2504.

Fernandez-Viagas, V. and Framiñán, J. M. (2020). Design of a testbed for hybrid flow shop scheduling with identical machines. *Computers & Industrial Engineering*, 141:106288.

Fernandez-Viagas, V., Molina-Pariente, J. M., and Framiñán, J. M. (2018). New efficient constructive heuristics for the hybrid flowshop to minimise makespan: A computational evaluation of heuristics. *Expert Systems with Applications*, 114:345–356.

Fernandez-Viagas, V., Pérez-González, P., and Framiñán, J. M. (2019). Efficiency of the solution representations for the hybrid flow shop scheduling problem with makespan objective. *Computers & Operations Research*, 109:77–88.

Fiszman, S. and Mosheiov, G. (2018). Minimizing total load on a proportionate flowshop with position-dependent processing times and job-rejection. *Information Processing Letters*, 132:39–43.

Framiñán, J. M., Leisten, R., and García, R. R. (2014). *Manufacturing scheduling systems.* Springer.

Framiñán, J. M., Leisten, R., and Rajendran, C. (2003). Different initial sequences for the heuristic of nawaz, enscore and ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research*, 41(1):121–148.

Framiñán, J. M., Leisten, R., and Ruiz-Usano, R. (2002). Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. *European Journal of Operational Research*, 141(3):559–569.

Garcia, J. M. and Lozano, S. (2005). Production and delivery scheduling problem with time windows. *Computers & Industrial Engineering*, 48:733–742.

Garey, M. R. and Johnson, D. S. (1979). Computers and intractability. *A Guide to the Theory of NP-Completeness*.

Gawiejnowicz, S. (1996). A note on scheduling on a single processor with speed dependent on a number of executed jobs. *Information Processing Letters*, 57(6):297–300.

Gawiejnowicz, S. (2020). A review of four decades of time-dependent scheduling: main results, new topics, and open problems. *Journal of Scheduling*, 23(1):3–47.

Geurtsen, M., Didden, J. B., Adan, J., Atan, Z., and Adan, I. (2023). Production, maintenance and resource scheduling: A review. *European Journal of Operational Research*, 305(2):501–529.

Ghaleb, M. A. and Alharkan, I. M. (2015). Efficient metaheuristics for multi-stage no-wait flexible flowshop scheduling problem. *Proceedings of the 2015 International Conference on Operations Excellence and Service Engineering*, pages 223–235.

Gicquel, C., Hege, L., Minoux, M., and Canneyt, W. V. (2012). A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints. *Computers & Operations Research*, 39(3):629–636.

Gilmore, P. C. and Gomory, R. E. (1964). Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12(5):655–679.

Glover, F. and McMillan, C. (1986). The general employee scheduling problem. an integration of ms and ai. *Computers & Operations Research*, 13(5):563–573.

González-Neira, E. M. and Montoya-Torres, J. R. (2017). A grasp meta-heuristic for the hybrid flowshop scheduling problem. *Journal of Decision Systems*, 26(3):294–306.

Gordon, V. S., Potts, C. N., Strusevich, V. A., and Whitehead, J. D. (2008). Single machine scheduling models with deterioration and learning: handling precedence constraints via priority generation. *Journal of Scheduling*, 11:357–370.

Grabowski, J. and Pempera, J. (2000). Sequencing of jobs in some production system. *European Journal of Operations Research*, 125:535–550.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326.

Grigoriu, L. and Briskorn, D. (2017). Scheduling jobs and maintenance activities subject to job-dependent machine deteriorations. *Journal of Scheduling*, 20:183–197.

Guirchoun, S., Martineau, P., and Billaut, J.-C. (2005). Total completion time minimization in a computer system with a server and two parallel processors. *Computers & Operations Research*, 32:599–611.

Gupta, J. N. D. (1988). Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, 39(4):359–364.

Gupta, J. N. D., Hariri, A. M. A., and Potts, C. N. (1997). Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. *Annals of Operations Research*, 69:171–191.

Hadidi, L. A., Al-Turki, U. M., and Rahim, M. A. (2011). An integrated cost model for production scheduling and perfect maintenance. *International Journal of Mathematics in Operational Research*, 3(4):395–413.

Hadidi, L. A., Al-Turki, U. M., and Rahim, M. A. (2015). Practical implications of managerial decisions to integrate production scheduling and maintenance. *International Journal of System Assurance Engineering and Management*, 6:224–230.

Hall, N. G. and Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510–525.

Haupt, R. (1989). A survey of priority rule-based scheduling. *OR Spectrum*, 11(1):3–16.

Hipp, A. (2024). Scheduling position-dependent maintenance operations in single machine layouts. In *Operations Research Proceedings 2023*. accepted.

Hipp, A. and Geldermann, J. (2020). Constructive heuristics in hybrid flow shop scheduling with unrelated machines and setup times. In *Operations Research Proceedings 2019*, pages 569–574. Springer.

Hipp, A. and Jaehn, F. (2024a). Single machine scheduling problems with fixed job positions and uniform job characteristics. *Working paper*.

Hipp, A. and Jaehn, F. (2024b). Single machine scheduling problems with position-dependent maintenance. *Working paper*.

Hipp, A., Jaehn, F., and Raue, C. (2024). The two-stage no-wait hybrid flow shop scheduling problem. *Working paper*.

Hoogeveen, J. A., Lenstra, J. K., and Veltman, B. (1996). Preemptive scheduling in a two-stage multiprocessor flow shop is np-hard. *European Journal of Operational Research*, 89(1):172–175.

Hopp, W. J. and Spearman, M. L. (2011). *Factory physics*. Waveland Press.

Hsu, C.-J. and Yang, D.-L. (2014). Unrelated parallel-machine scheduling with position-dependent deteriorating jobs and resource-dependent processing time. *Optimization Letters*, 8:519–531.

Hwang, F. J. and Lin, B. M. T. (2018). Survey and extensions of manufacturing models in two-stage flexible flow shops with dedicated machines. *Computers & Operations Research*, 98:103–112.

Jaehn, F. (2024). Scheduling with jobs at fixed positions. *Submitted*.

Jaehn, F. and Pesch, E. (2019). *Ablaufplanung*. Springer.

Jansen, K., Klein, K.-M., and Verschae, J. (2020). Closing the gap for makespan scheduling via sparsification techniques. *Mathematics of Operations Research*, 45(4):1371–1392.

Ji, M., He, Y., and Cheng, T. E. (2006). Scheduling linear deteriorating jobs with an availability constraint on a single machine. *Theoretical Computer Science*, 362(1-3):115–126.

Ji, P., Li, G., Huo, Y., and Wang, J.-B. (2014). Single-machine common flow allowance scheduling with job-dependent aging effects and a deteriorating maintenance activity. *Optimization Letters*, 8:1389–1400.

Johnson, S. M. (1954). Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1):61–68.

Jolai, F., Asefi, H., Rabiee, M., and Ramezani, P. (2013). Bi-objective simulated annealing approaches for no-wait two-stage flexible flow shop scheduling problem. *Scientia Iranica E*, 20(3):861–872.

Jungwattanakit, J., Reodecha, M., Chaovalitwongse, P., and Werner, F. (2005). An evaluation of sequencing heuristics for flexible flowshop scheduling problems with unrelated parallel machines and dual criteria. *Otto-von-Guericke-Universitat Magdeburg*, 28(05):1–23.

Kellerer, H., Pferschy, U., Pisinger, D., and Kellerer, H. G. (2004). *Knapsack problems*. Springer, Heidelberg [u.a.];Berlin.

Korte, B. H. and Vygen, J. (2018). *Combinatorial optimization*, volume 6. Springer.

Koulamas, C. (1994). The total tardiness problem: review and extensions. *Operations research*, 42(6):1025–1041.

Kubzin, M. A. and Strusevich, V. A. (2006). Planning machine maintenance in two-machine shop scheduling. *Operations Research*, 54(4):789–800.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97.

Kuo, W. and Yang, D.-L. (2008). Minimizing the makespan in a single-machine scheduling problem with the cyclic process of an aging effect. *Journal of the Operational Research Society*, 59(3):416–420.

Labetoulle, J., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1984). Preemptive scheduling of uniform machines subject to release dates. In *Progress in combinatorial optimization*, pages 245–261. Elsevier.

Lancaster, J. F. (1999). *Metallurgy of welding*. Elsevier.

Lawler, E. L. (1977). A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. In *Annals of discrete Mathematics*, volume 1, pages 331–342. Elsevier.

Lee, C.-Y. and Leon, V. J. (2001). Machine scheduling with a rate-modifying activity. *European Journal of Operational Research*, 128(1):119–128.

Lee, C.-Y. and Lin, C.-S. (2001). Single-machine scheduling with maintenance and repair rate-modifying activities. *European Journal of Operational Research*, 135(3):493–513.

Lee, C.-Y. and Vairaktarakis, G. L. (1994). Minimizing makespan in hybrid flowshops. *Operations Research Letters*, 16(3):149–158.

Lee, J.-Y. and Kim, Y.-D. (2017). Minimizing total tardiness in a two-machine flowshop scheduling problem with availability constraint on the first machine. *Computers & Industrial Engineering*, 114:22–30.

Lee, T. and Loong, Y. (2019). A review of scheduling problem and resolution methods in flexible flow shop. *International Journal of Industrial Engineering Computations*, 10(1):67–88.

Leisten, R. and Rajendran, C. (2015). Variability of completion time differences in permutation flow shop scheduling. *Computers & Operations Research*, 54:155–167.

Lenstra, J. K., Kan, A. R., and Brucker, P. (1977). Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, pages 343–362. Elsevier.

Liang, T.-F. (2006). Distribution planning decisions using interactive fuzzy multi-objective linear programming. *Fuzzy Sets and Systems*, 157(10):1303–1316.

Linn, R. and Zhang, W. (1999). Hybrid flow shop scheduling: A survey. *Computers and Industrial Engineering*, 37:57–61.

Liu, J. and Reeves, C. R. (2001). Constructive and composite heuristic solutions to the $p//\sum c_i$ scheduling problem. *European Journal of Operational Research*, 132:439–452.

Liu, P., Yi, N., Zhou, X., and Gong, H. (2013). Scheduling two agents with sum-of-processing-times-based deterioration on a single machine. *Applied Mathematics and Computation*, 219(17):8848–8855.

Low, C., Ji, M., Hsu, C.-J., and Su, C.-T. (2010). Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance. *Applied Mathematical Modelling*, 34(2):334–342.

Lui, Y. and Karimi, I. A. (2008). Scheduling multistage batch plants with parallel units and no interstage storage. *Computers and Chemical Engineering*, 32:671–693.

Luo, W. and Liu, F. (2017). On single-machine scheduling with workload-dependent maintenance duration. *Omega*, 68:119–122.

Maassen, K. and Pérez-González, P. (2020). Diversity of processing times in permutation flow shop scheduling problems. In *Operations Research Proceedings 2019*, pages 555–561. Springer.

Meng, L., Zhang, C., Shao, X., Zhang, B., Ren, Y., and Lin, W. (2019). More milp models for hybrid flow shop scheduling problem and its extended problems. *International Journal of Production Research*, pages 1–26.

Mirsanei, H., Zandieh, M., Moayed, M. J., and Khabbazi, M. R. (2011). A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 22:965–978.

Montoya-Torres, J. R., Botta-Genoulaz, V., Materzok, N., Gíslason, Þ. P., and Mendiela, S. (2021). Modeling the parallel machine scheduling problem with worker-and position-dependent processing times. In *Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems: IFIP WG 5.7 International Conference, APMS 2021*, pages 351–359. Springer.

Montoya-Torres, J. R., Moreno-Camacho, C. A., and Vélez-Gallego, M. C. (2023). Variable neighbourhood search for job scheduling with position-dependent deteriorating processing times. *Journal of the Operational Research Society*, 74(3):873–887.

Moon, S., Park, S., and Lee, W. K. (1996). New milp models for scheduling of multiproduct batch plants under zero-wait policy. *Industrial & Engineering Chemistry Research*, 35:3458–3469.

Moore, J. M. (1968). An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109.

Mor, B. and Mosheiov, G. (2012). Scheduling a maintenance activity and due-window assignment based on common flow allowance. *International Journal of Production Economics*, 135(1):222–230.

Mor, B. and Mosheiov, G. (2015). Scheduling a deteriorating maintenance activity and due-window assignment. *Computers & Operations Research*, 57:33–40.

Mor, B. and Mosheiov, G. (2018). A note: minimizing total absolute deviation of job completion times on unrelated machines with general position-dependent processing times and job-rejection. *Annals of Operations Research*, 271:1079–1085.

Morsy, E. and Pesch, E. (2015). Approximation algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, 18:645–653.

Mosheiov, G. (2001). Scheduling problems with a learning effect. *European Journal of Operational Research*, 132(3):687–693.

Mosheiov, G. (2005). A note on scheduling deteriorating jobs. *Mathematical and Computer Modelling*, 41(8-9):883–886.

Mosheiov, G. (2008). Minimizing total absolute deviation of job completion times: extensions to position-dependent processing times and parallel identical machines. *Journal of the Operational Research Society*, 59:1422–1424.

Mosheiov, G. (2011). Proportionate flowshops with general position-dependent processing times. *Information Processing Letters*, 111(4):174–177.

Mosheiov, G. and Sarig, A. (2008). A due-window assignment problem with position-dependent processing times. *Journal of the Operational Research Society*, 59(7):997–1003.

Mosheiov, G., Sarig, A., and Strusevich, V. (2020). Minmax scheduling and due-window assignment with position-dependent processing times and job rejection. *4OR*, 18:439–456.

Mosheiov, G. and Sidney, J. B. (2003). Scheduling with general job-dependent learning curves. *European Journal of Operational Research*, 147(3):665–670.

Mosheiov, G. and Sidney, J. B. (2005). Note on scheduling with general learning curves to minimize the number of tardy jobs. *Journal of the Operational Research Society*, 56:110–112.

Mosheiov, G. and Sidney, J. B. (2010). Scheduling a deteriorating maintenance activity on a single machine. *Journal of the Operational Research Society*, 61(5):882–887.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38.

Naderi, B., Gohari, S., and Yazdani, M. (2014a). Hybrid flexible flowshop problems: Models and solution methods. *Applied Mathematical Modelling*, 38(24):5767–5780.

Naderi, B., Khalili, M., and Khamseh, A. A. (2014b). Mathematical models and a hunting search algorithm for the no-wait flowshop scheduling with parallel machines. *International Journal of Production Research*, 52(9):2667–2681.

Naderi, B., Zandieh, M., Balagh, A. K. G., and Roshanaei, V. (2009). An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. *Expert systems with Applications*, 36(6):9625–9633.

Narasimhan, S. L. and Mangiameli, P. M. (1987). A comparison of sequencing rules for a two-stage hybrid flowshop. *Decision Sciences*, 18(2):250–265.

Narasimhan, S. L. and Panwalkar, S. S. (1984). Scheduling in a two-stage manufacturing process. *International Journal of Production Research*, 22(2):555–564.

Nawaz, M., Enscore Jr, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95.

Öztop, H., Tasgetiren, M. F., Eliiyi, D. T., and Pan, Q.-K. (2019). Metaheuristic algorithms for the hybrid flowshop scheduling problem. *Computers & Operations Research*, 111:177–196.

Pan, Q.-K., Ruiz, R., and Alfaro-Fernández, P. (2017). Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. *Computers & Operations Research*, 80:50–60.

Pei, J., Zhou, Y., Yan, P., and Pardalos, P. M. (2023). A concise guide to scheduling with learning and deteriorating effects. *International Journal of Production Research*, 61(6):2010–2031.

Perez-Gonzalez, P. and Framiñán, J. M. (2018). Single machine scheduling with periodic machine availability. *Computers & Industrial Engineering*, 123:180–188.

Pinedo, M. L. (2022). *Scheduling*, volume 29. Springer.

Potts, C. N. and Van Wassenhove, L. N. (1992). Single machine scheduling to minimize total late work. *Operations Research*, 40(3):586–595.

Quadt, D. and Kuhn, H. (2007). A taxonomy of flexible flow line scheduling procedures. *European Journal of Operational Research*, 178:686–698.

Rabiee, M., Jolai, F., Asefi, H., Fattahi, P., and Lim, S. (2016). A biogeography-based optimisation algorithm for a realistic no-wait hybrid flow shop with unrelated parallel machines to minimise mean tardiness. *International Journal of Computer Integrated Manufacturing*, 29(9):1007–1024.

Ramezani, P., Rabiee, M., and Jolai, F. (2015). No-wait flexible flowshop with uniform parallel machines and sequence-dependent setup time: a hybrid meta-heuristic approach. *Journal of Intelligent Manufacturing*, 26(4):731–744.

Rashidi, E., Jahandar, M., and Zandieh, M. (2010). An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. *The International Journal of Advanced Manufacturing Technology*, 49:1129–1139.

Reddi, S. S. and Ramamoorthy, C. V. (1972). On the flow-shop sequencing problem with no wait in process. *Operations Research Quarterly*, 23:323–331.

Ribas, I., Leisten, R., and Framiñan, J. M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, 37(8):1439–1454.

Ruiz, R., Şerifoğlu, F. S., and Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, 35(4):1151–1175.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Ruiz, R. and Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. *European journal of operational research*, 205(1):1–18.

Rustogi, K. and Strusevich, V. A. (2012a). Simple matching vs linear assignment in scheduling models with positional effects: A critical review. *European Journal of Operational Research*, 222(3):393–407.

Rustogi, K. and Strusevich, V. A. (2012b). Single machine scheduling with general positional deterioration and rate-modifying maintenance. *Omega*, 40(6):791–804.

Ruther, S. (2013). Integrated aircraft routing, crew pairing, and tail assignment. *Doctoral dissertation, University of Newcastle, Newcastle, Australia*, 13585.

Salama, M. and Srinivas, S. (2021). Adaptive neighborhood simulated annealing for sustainability-oriented single machine scheduling with deterioration effect. *Applied Soft Computing*, 110:107632.

Sanlaville, E. and Schmidt, G. (1998). Machine scheduling with availability constraints. *Acta Informatica*, 35:795–811.

Schulze, M., Rieck, J., Seifi, C., and Zimmermann, J. (2016). Machine scheduling in underground mining: an application in the potash industry. *OR Spectrum*, 38(2):365–403.

Seif, J., Yu, A. J., and Rahmanniyay, F. (2018). Modelling and optimization of a bi-objective flow shop scheduling with diverse maintenance requirements. *International Journal of Production Research*, 56(9):3204–3225.

Shahvari, O. and Logendran, R. (2018). A comparison of two stage-based hybrid algorithms for a batch scheduling problem in hybrid flow shop with learning effect. *International Journal of Production Economics*, 195:227–248.

Smith, W. E. et al. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66.

Sriram, C. and Haghani, A. (2003). An optimization model for aircraft maintenance scheduling and re-assignment. *Transportation Research Part A: Policy and Practice*, 37(1):29–48.

Sriskandarajah, C. (1993). Performance of scheduling algorithms for no-wait flowshops with parallel machines. *European Journal of Operational Research*, 70:365–378.

Sriskandarajah, C. and Sethi, S. P. (1989). Scheduling algorithms for flexible flowshops: Worst and average case performance. *European Journal of Operational Research*, 43:143–160.

Steland, A. (2004). *Mathematische Grundlagen der empirischen Forschung*. Springer-Verlag, Berlin und Heidelberg.

Tan, Y., Mönch, L., and Fowler, J. W. (2018). A hybrid scheduling approach for a two-stage flexible flow shop with batch processing machines. *Journal of Scheduling*, 21(2):209–226.

Tang, L. and Wang, X. (2010). An improved particle swarm optimization algorithm for the hybrid flowshop scheduling to minimize total weighted completion time in process industry. *IEEE Transactions on Control Systems Technology*, 18(6):1303–1314.

Tian, H., Yu, S., and Luo, W. (2019). Parallel machine scheduling on jobs and partial maintenance activities due to job-dependent machine deteriorations. *American Journal of Mathematical and Management Sciences*, 38(3):250–260.

Tian, Z., Ng, C. T., and Cheng, T. E. (2006). An o (n 2) algorithm for scheduling equal-length preemptive jobs on a single machine to minimize total tardiness. *Journal of Scheduling*, 9:343–364.

Turing, A. M. (1937). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265.

Vallada, E., Ruiz, R., and Framiñán, J. M. (2015). New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240:666–677.

Vignier, A., Billaut, J.-C., and Proust, C. (1999). Les problèmes d'ordonnancement de type flow-shop hybride: état de l'art. *Revue francais d'automatique, d'informatique et de recherche opérationnelle, Recherche opérationnelle*, 33(2):117–183.

Wang, B., Huang, K., and Li, T. (2018). Two-stage hybrid flowshop scheduling with simultaneous processing machines. *Journal of Scheduling*, 21(4):387–411.

Wang, J.-J., Wang, J.-B., and Liu, F. (2011). Parallel machines scheduling with a deteriorating maintenance activity. *Journal of the Operational Research Society*, 62(10):1898–1902.

Wang, S. and Liu, M. (2013). A genetic algorithm for two-stage no-wait hybrid flow shop scheduling problem. *Computer & Operations Research*, 40:1064–1075.

Wang, S., Liu, M., and Chu, C. (2015). A branch-and-bound algorithm for two-stage no-wait hybrid flow shop scheduling. *International Journal of Production Research*, 43(4):1143–1167.

Wang, S., Wang, X., and Yu, L. (2019). Two-stage no-wait hybrid flow-shop scheduling with sequence-dependent setup times. *International Journal of Systems Science: Operations & Logistics*, pages 1–17.

Wang, S., Wang, X., and Yu, L. (2020). Two-stage no-wait hybrid flow-shop scheduling with sequence-dependent setup times. *International Journal of Systems Science: Operations & Logistics*, 7(3):291–307.

Wang, Z., Xing, W., and Bai, F. (2005). No-wait flexible flowshop scheduling with no-idle machines. *Operations Research Letters*, 33:609–614.

Xie, J., Xing, W., Liu, Z., and Dong, J. (2004). Minimum deviation algorithm for two-stage no-wait flowshops with parallel machines. *Computers and Mathematics with Applications*, 47:1857–1863.

Xu, D., Wan, L., Liu, A., and Yang, D.-L. (2015). Single machine total completion time scheduling problem with workload-dependent maintenance duration. *Omega*, 52:101–106.

Xu, Z. and Xu, D. (2015). Single-machine scheduling with preemptive jobs and workload-dependent maintenance durations. *Operational Research*, 15:423–436.

Xue, P., Zhang, Y., and Yu, X. (2014). Single-machine scheduling with piece-rate maintenance and interval constrained position-dependent processing times. *Applied Mathematics and Computation*, 226:415–422.

Yang, D.-L., Cheng, T., and Yang, S.-J. (2014). Parallel-machine scheduling with controllable processing times and rate-modifying activities to minimise total cost involving total completion time and job compressions. *International Journal of Production Research*, 52(4):1133–1141.

Yang, S. and Yang, D.-L. (2012). Single-machine scheduling problems with past-sequence-dependent delivery times and position-dependent processing times. *Journal of the Operational Research Society*, 63(11):1508–1515.

Yang, S.-J. (2013). Unrelated parallel-machine scheduling with deterioration effects and deteriorating multi-maintenance activities for minimizing the total completion time. *Applied Mathematical Modelling*, 37(5):2995–3005.

Yang, S.-J. and Yang, D.-L. (2010a). Minimizing the makespan on single-machine scheduling with aging effect and variable maintenance activities. *Omega*, 38(6):528–533.

Yang, S.-J. and Yang, D.-L. (2010b). Minimizing the total completion time in single-machine scheduling with aging/deteriorating effects and deteriorating maintenance activities. *Computers & Mathematics with Applications*, 60(7):2161–2169.

Yu, X., Zhang, Y., and Huang, K. (2014). Multi-machine scheduling with general position-based deterioration to minimize total load revisited. *Information Processing Letters*, 114(8):399–404.

Zhang, X., Yin, Y., and Wu, C.-C. (2017). Scheduling with non-decreasing deterioration jobs and variable maintenance activities on a single machine. *Engineering Optimization*, 49(1):84–97.

Zhao, C.-l. and Tang, H.-y. (2010). Single machine scheduling with general job-dependent aging effect and maintenance activities to minimize makespan. *Applied Mathematical Modelling*, 34(3):837–841.

Appendix A

Assignment of the Papers to the Respective Chapters

This dissertation is based on published papers or papers in progress or accepted at the time of submission. These papers are assigned to the chapters as follows:

- Section 2.2: *Basic Notation*, especially the introduced additional β-entries for the Graham notation:
  Briskorn et al. (2024)

- Chapter 3: *Classification of Machine Availability Models in Scheduling*:
  Briskorn et al. (2024)

- Section 4.1: *Single Machine Problems with Position-Dependent Maintenance*, and Section 4.2: *Minimizing Total Weighted Completion Time with Position-Dependent Maintenance, Release Dates, and Uniform Processing Times*:
  Hipp and Jaehn (2024b)

- Section 4.3: *Minimizing the Number of Tardy Jobs with Position-Dependent Maintenance, Release Dates, and Uniform Processing Times*:
  Hipp (2024)

- Chapter 5: *Single Machine Scheduling Problems with Fixed Positioned Jobs*:
  Hipp and Jaehn (2024a)

- Chapter 6: *The Two-Stage no-wait Hybrid Flow Shop Scheduling Problem*:
  Hipp et al. (2024)
- Chapter 7: *Hybrid Flow Shop Scheduling Problems with Setup Times*:
  Hipp and Geldermann (2020)

Appendix B

Chapter 6: Computational Results

Tab. B.1: ARPD-values for C_{max}: MILP vs. best 6 heuristics A1

C_{max} **FAM A1**							
n	MILP	HILL	SPT	LPT	VALLEY	LPT(a_j)	SPT(a_j)
10	5.98	19.64	20.89	34.47	38.19	34.70	36.01
15	11.61	2.90	4.31	13.46	15.54	15.16	14.31
20	0.00	1.45	2.23	9.41	11.06	11.86	13.03
50	21.25	0.68	1.12	3.99	4.80	8.27	8.51
100	3.81	0.24	0.58	2.00	2.44	6.72	6.97
150		0.18	0.48	1.53	1.59	5.96	6.18
200		0.14	0.35	1.03	1.30	5.13	5.33
Mean	**8.53**	**3.60**	**4.28**	**9.41**	**10.70**	**12.54**	**12.91**
Rank	**3**	**1**	**2**	**4**	**5**	**6**	**7**

C_{max} **MIT A1**							
n	MILP	HILL	SPT	HILL(a_j)	SPT(a_j)	LOLOHI	LPT(a_j)
10	6.56	20.10	21.31	29.95	31.35	29.86	32.12
15	11.78	4.42	5.81	11.91	11.01	1.67	13.36
20	0.00	2.18	2.96	8.64	9.76	9.26	9.31
50	22.43	1.57	2.02	5.54	5.32	5.98	5.27
100	4.17	0.86	1.20	3.24	3.52	4.00	3.42
150		0.76	1.07	3.09	2.96	3.19	2.86
200		0.45	0.66	2.58	2.78	2.90	2.52
Mean	**8.99**	**4.34**	**5.00**	**9.28**	**9.53**	**9.69**	**9.84**
Rank	**3**	**1**	**2**	**4**	**5**	**6**	**7**

Tab. B.2: ARPD-values for C_{max}: MILP vs. best 6 heuristics A2

C_{max} **FAM A2**							
n	MILP	NEH	HILL	SPT	LR	GG	LPT
10	8.66	17.09	20.74	23.29	20.43	19.81	31.15
15	13.96	3.69	5.19	7.40	5.75	5.46	12.75
20	1.60	2.17	3.31	5.84	4.60	4.17	9.34
50	24.68	1.18	1.17	2.64	3.26	4.44	4.10
100	11.76	0.60	0.70	1.53	3.04	4.24	2.25
150		0.55	0.47	1.05	2.80	3.86	1.61
200		0.36	0.49	0.97	2.84	3.99	1.25
Mean	**12.13**	**3.66**	**4.58**	**6.10**	**6.10**	**6.57**	**8.92**
Rank	**8**	**1**	**2**	**3**	**4**	**5**	**6**

C_{max} **MIT A2**							
n	MILP	NEH	HILL	GG	SPT	LR	LPT
10	9.02	16.47	20.79	18.96	23.21	20.68	31.47
15	7.96	4.09	5.93	4.54	8.24	6.56	13.69
20	1.60	2.32	4.19	3.77	6.68	5.49	10.28
50	24.68	1.19	1.60	3.83	3.08	3.68	4.56
100	11.76	0.66	0.95	3.70	1.78	3.27	2.51
150		0.59	0.66	3.45	1.24	2.98	1.81
200		0.34	0.54	3.48	1.01	2.87	1.29
Mean	**11.01**	**3.66**	**4.95**	**5.96**	**6.46**	**6.51**	**9.37**
Rank	**7**	**1**	**2**	**3**	**4**	**5**	**6**

Tab. B.3: ARPD-values for F: MILP vs. best 6 heuristics A1

F **FAM A1**							
n	MILP	SPT	HILL	SPT(a_j)	LOLOHI	VALLEY	LOLOHI(a_j)
10	0.01	12.23	23.86	43.86	43.36	59.67	46.56
15	0.51	10.21	21.07	35.50	42.86	47.80	40.18
20	3.24	4.33	19.29	30.27	34.70	39.31	35.60
50	66.34	0.00	14.35	18.30	25.33	22.12	25.77
100	80.87	0.00	14.72	16.68	24.70	18.85	25.25
150	85.44	0.00	13.93	15.49	23.71	17.09	24.48
200	88.38	0.00	14.54	14.80	23.55	16.70	25.36
Mean	**46.40**	**3.82**	**17.39**	**24.98**	**31.17**	**31.65**	**31.88**
Rank	**15**	**1**	**2**	**3**	**4**	**5**	**6**

F **MIT A1**							
n	MILP	SPT	HILL	SPT(a_j)	LOLOHI	LOHILO	HILL(a_j)
10	0.01	12.08	23.72	40.84	36.54	49.30	43.00
15	0.52	10.16	21.04	32.81	36.04	33.58	37.26
20	3.26	4.33	19.29	27.80	27.39	33.41	31.99
50	66.42	0.05	14.41	16.29	18.15	19.66	21.08
100	80.88	0.00	14.72	14.47	16.84	17.96	18.53
150	85.44	0.00	13.93	13.59	15.73	16.50	17.80
200	88.38	0.00	14.54	13.39	15.84	16.13	17.37
Mean	**46.42**	**3.80**	**17.38**	**22.74**	**23.79**	**26.65**	**26.72**
Rank	**17**	**1**	**2**	**3**	**4**	**5**	**6**

Tab. B.4: ARPD-values for F: MILP vs. best 6 heuristics A2

F **FAM A2**							
n	MILP	SPT	LR	NEH	HILL	GG	SPT(a_j)
10	0.14	6.39	7.13	24.18	23.79	28.56	44.34
15	1.85	4.05	5.18	19.99	20.91	26.62	38.11
20	7.65	1.94	2.94	17.26	20.76	25.28	35.56
50	84.21	0.69	2.25	12.99	20.66	26.78	30.47
100	102.42	0.53	2.58	11.05	22.31	28.21	29.63
150	107.64	0.28	2.29	10.94	21.76	26.28	28.95
200	110.72	0.25	2.42	9.31	21.96	26.55	28.65
Mean	**59.23**	**2.02**	**3.54**	**15.10**	**21.74**	**26.90**	**33.67**
Rank	**19**	**1**	**2**	**3**	**4**	**5**	**6**
F **MIT A2**							
n	MILP	SPT	LR	NEH	HILL	GG	SPT(a_j)
10	0.21	6.18	7.18	23.91	23.59	27.72	41.75
15	1.88	4.02	5.12	19.89	20.78	24.52	35.61
20	7.72	1.91	3.00	17.24	20.79	23.90	33.17
50	84.24	0.70	2.21	12.99	20.66	25.53	28.02
100	102.49	0.55	2.55	11.09	22.35	27.34	27.01
150	107.70	0.30	2.30	10.96	21.78	25.58	26.11
200	110.76	0.26	2.41	9.33	21.98	25.89	26.13
Mean	**59.29**	**1.99**	**3.54**	**15.06**	**21.70**	**25.78**	**31.11**
Rank	**20**	**1**	**2**	**3**	**4**	**5**	**6**

Tab. B.5: Tested hypotheses for MIT A1 C_{max}

(1) H0(1): $\text{HILL}(p_{1j}) \geq \text{SPT}(p_{1j})$ H1(1): $\text{HILL}(p_{1j}) < \text{SPT}(p_{1j})$

(2) H0(2): $\text{SPT}(p_{1j}) \geq \text{HILL}(a_{1j})$ H1(2): $\text{SPT}(p_{1j}) < \text{HILL}(a_{1j})$

(3) H0(3): $\text{HILL}(p_{1j}) \geq \text{HILL}(a_{1j})$ H1(3): $\text{HILL}(p_{1j}) < \text{HILL}(a_{1j})$

Tab. B.6: Tested hypotheses for MIT A2 C_{max}

(1)	H0(1):	NEH $\geq$ HILL(p_j)	H1(1):	NEH $<$ HILL(p_j)
(2)	H0(2):	HILL(p_j) $\geq$ GILGOM	H1(2):	HILL(p_j) $<$ GILGOM
(3)	H0(3):	GILGOM $\geq$ SPT(p_j)	H1(3):	GILGOM $<$ SPT(p_j)
(4)	H0(4):	SPT(p_j) $\geq$ LR	H1(4):	SPT(p_j) $<$ LR
(5)	H0(5):	LR $\geq$ LPT(p_j)	H1(5):	LR $<$ LPT(p_j)
(6)	H0(6):	NEH $\geq$ LPT(p_j)	H1(6):	NEH $<$ LPT(p_j)
(7)	H0(7):	NEH $\geq$ GILGOM	H1(7):	NEH $<$ GILGOM
(8)	H0(8):	NEH $\geq$ LR	H1(8):	NEH $<$ LR
(9)	H0(9):	NEH $\geq$ SPT(p_j)	H1(9):	NEH $<$ SPT(p_j)

Tab. B.7: Tested hypotheses for FAM A1 F

(1)	H0(1):	SPT(p_{1j}) $\geq$ HILL(p_{1j})	H1(1):	SPT(p_{1j}) $<$ HILL(p_{1j})
(2)	H0(2):	HILL(p_{1j}) $\geq$ SPT(a_{1j})	H1(1):	HILL(p_{1j}) $<$ SPT(a_{1j})
(3)	H0(3):	SPT(p_{1j}) $\geq$ SPT(a_{1j})	H1(1):	SPT(p_{1j}) $<$ SPT(a_{1j})

Tab. B.8: Tested hypotheses for FAM A2 F

(1)	H0(1):	SPT(p_j) $\geq$ LR	H1(1):	SPT(p_j) $<$ LR
(2)	H0(2):	LR $\geq$ NEH	H1(2):	LR $<$ NEH
(3)	H0(3):	SPT(p_j) $\geq$ NEH	H1(3):	SPT(p_j) $<$ NEH

Tab. B.9: Accepted hypotheses for MIT A1 F

MIT A1										
	$i=1$			$i=2$			$i=3$			
n	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	best
10	1	0.9750	H1	1	0.9833	H1	1	0.9500	H1	**SPT**(p_{1j})
15	1	0.9750	H1	1	0.9833	H1	1	0.9500	H1	**SPT**(p_{1j})
20	1	0.9500	H1	1	0.9833	H1	1	0.9750	H1	**SPT**(p_{1j})
50	1	0.9500	H1	1	0.9833	H1	1	0.9750	H1	**SPT**(p_{1j})
100	1	0.9500	H1	0.7123	0.9833	H0	1	0.9750	H1	**SPT**(p_{1j})
150	1	0.9500	H1	0.7995	0.9833	H0	1	0.9750	H1	**SPT**(p_{1j})
200	1	0.9500	H1	0	0.9833	H0	1	0.9750	H1	**SPT**(p_{1j})

Tab. B.10: Accepted hypotheses for MIT A2 F

MIT A2										
	$i=1$			$i=2$			$i=3$			
n	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	p-value	$1-\alpha_{local}$	H	best
10	0.8980	0.9833	H0	1	0.9750	H1	1	0.9500	H1	**SPT**(p_j), **LR**
15	0.9857	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**SPT**(p_j)
20	0.9726	0.9833	H0	1	0.9750	H1	1	0.9500	H1	**SPT**(p_j), **LR**
50	1	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**SPT**(p_j)
100	1	0.9833	H1	1	0.9750	H1	1	0.9500	H1	**SPT**(p_j)
150	1	0.9750	H1	1	0.9833	H1	1	0.9500	H1	**SPT**(p_j)
200	1	0.9500	H1	1	0.9833	H1	1	0.9750	H1	**SPT**(p_j)