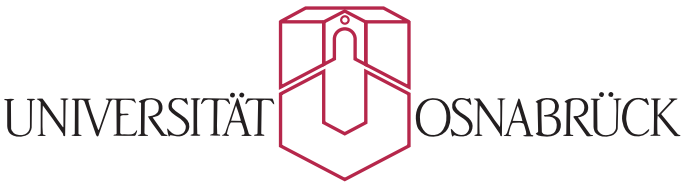


Scheduling of flow shops with synchronous movement

Dissertation
zur Erlangung des Doktorgrades (Dr. rer. nat.)
des Fachbereichs Mathematik und Informatik
der Universität Osnabrück

vorgelegt
von
Dipl. Math. Stefan Waldherr

31. Juli 2015

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich in den vier Jahren der Promotion unterstützt haben.

Mein besonderer Dank gilt Prof. Dr. Sigrid Knust für ihre ausgezeichnete und aufopferungsvolle Betreuung und die mir gegebene Möglichkeit an diesem interessanten Thema in einem angenehmen Umfeld zu arbeiten. Sie unterstützte mich enorm in allen organisatorischen und fachlichen Angelegenheiten und war maßgeblich für den reibungslosen Ablauf der Kooperation mit dem in dieser Arbeit vorgestellten Praxispartner verantwortlich. Ihre zahlreichen wertvollen Anregungen trugen immens zum Gelingen dieser Arbeit bei.

Weiterhin danke ich allen Kollegen und Studenten, durch die ich in ergiebigen Gesprächen wertvolle neue Ideen entwickeln konnte. Insbesondere danke ich Jana Lehnfeld für ihre große Hilfe beim Korrekturlesen der vorliegenden Arbeit.

Ebenfalls bedanke ich mich bei meinen ehemaligen studentischen Hilfskräften Sven Boge, Sebastian Brockmeyer und Matthias Kampmeyer für ihre Hilfe beim Implementieren der in dieser Arbeit vorgestellten Algorithmen.

Großer Dank gebührt meinen Eltern, Georg und Ute, dafür dass sie es mir meine schulische und universitäre Ausbildung ermöglicht und mich in allen Lebenslagen unterstützt haben.

Herzlich möchte ich mich bei allen Freunden bedanken, die mir immer ein großer Rückhalt waren und mir immer eine ausgezeichnete Ablenkungen von meiner Arbeit beschert haben. Ganz besonderer Dank gilt meiner Verlobten Anna, deren Liebe und Unterstützung mir in den letzten Jahren immer ein großer Rückhalt war. Auch für ihre Unterstützung beim Korrekturlesen der Arbeit bedanke ich mich herzlich.

Contents

Chapter 1

Introduction

Scheduling problems have been a very important subject in combinatorial optimization for a long time. In general, a scheduling problem consists of finding a schedule for a number of tasks which need to be completed. If costs and payoffs are associated with schedules, the goal is to find a schedule which maximizes the payoff. In machine scheduling, tasks relate to the assembly and processing of products. The production process can be affected by a vast number of constraints: the available number and type of machines; limits in resources and workforce; or constraints that are enforced by the products, machines or the particular work process.

With increased flexibility in production, the importance of machine scheduling increases as well. Nowadays, customers expect a high product diversity. This leads to a shift away from mass production, where one product is manufactured in a single variant, to the newer paradigm of mass customization. The goal of mass customization according to Joe Pine, one of its first analysts, is "developing, producing, marketing, and delivering affordable goods and services with enough variety and customization that nearly everyone finds exactly what they want" (Pine, 1999, p. 44). This wish for diversity forces companies to steadily increase their product variety, for instance by manufacturing one basis product in a high number of variants. An example of this can be found in the automobile industry: Stautner (2001) describes that BMW can theoretically offer up to 10^{32} variants of its cars, several thousand of which are actually ordered. This trend also reaches out to other areas like electronics, clothings and furniture manufacturing. In Chapter 6 we will take a look at a company manufacturing shelf boards for kitchen elements which offers over a thousand variants of its shelf boards.

Since in mass production only a single variant was produced, no further considerations had to be given to the sequence of products to be assembled. Neither changeover times nor product-specific sequence dependencies had to be taken into account. When producing a high variety of a basis product, however, reconfigurations may be necessary and production times may vary drastically between variants. Depending on the types of sequence dependencies and the costs in time of switching from one variant to another, two philosophies of manufacturing stand in contrast to mass production of a single product:

One possibility is to manufacture similar product variants in batches and to reconfigure all workstations after the production of each batch. This is of interest in production lines where changeover between different variants comes at high costs or when one machine can process several products simultaneously. Consider for example a company that bottles different beverages and a change in production requires intense cleaning of the whole production units such that no traces of the previous product can be found in the succeeding product. See, e.g., Potts and Kovalyov (2000) for a survey in batch scheduling.

On the other hand, if variants of a basis product are very similar to each other and no complicated reconfiguration of machines is necessary, they can be produced in an arbitrary sequence in mixed model lines. Processing times on each workstation may vary and thus finding a sequence to minimize the total production time is of importance. Within this work we will focus on mixed model lines. In the following we will present the production system in the scope of this thesis. We will introduce the machine environment and discuss the transportation of products within the production process.

The machine environment

Traditionally, automated production systems are of great importance in industry. Therein, required work pieces move along a conveyor belt or some other underlying transportation system between distinct workstations until the product is assembled completely. Work pieces can either be fixed directly to the conveyor belt or be transported in work piece carriers. The workstations are equipped with dedicated specialized machines or human workforce which work on the material in several operations. Each of these operations requires a certain amount of time, not only depending on the distinct operations but on the variant of the product to be manufactured as well.

In this work we will focus on production processes in which all products are manufactured via the same fixed sequence of workstations. In the scheduling literature, such a sequential process is called a flow shop, an example of a flow shop with three workstations can be seen in Figure 1.1. Each product moves along the conveyor belt from one workstation to the next until it is completely assembled.

According to Groover (2007), automated production systems can be classified into three distinct system configurations:

1) In-line configurations, which consist of a sequence of stations in a straight line (Figure 1.1).

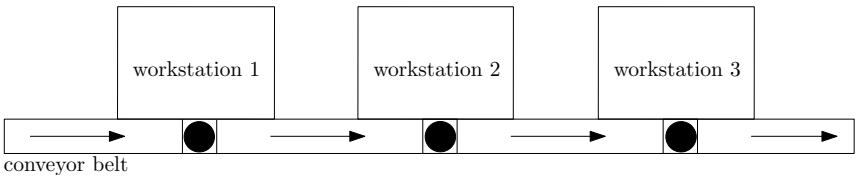


Figure 1.1: An assembly line with three workstations in a flow shop

2) Segmented in-line configurations, which consist of a sequence of straight line segments that are usually perpendicular to each other (Figure 1.2).

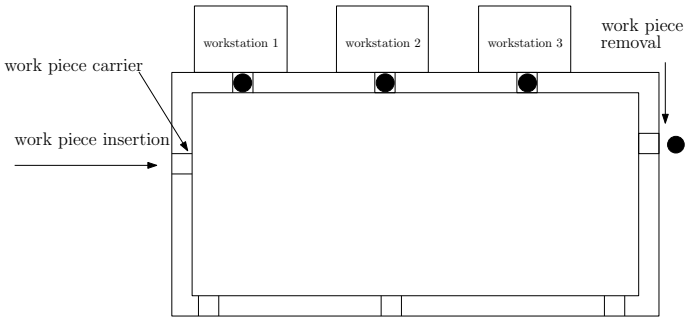


Figure 1.2: Segmented in-line configuration forming a rectangle

3) Rotary configurations, where workstations are located around a circular transportation system (Figure 1.3). This configuration is similar to a segmented in-line configuration where the lines form a rectangle.

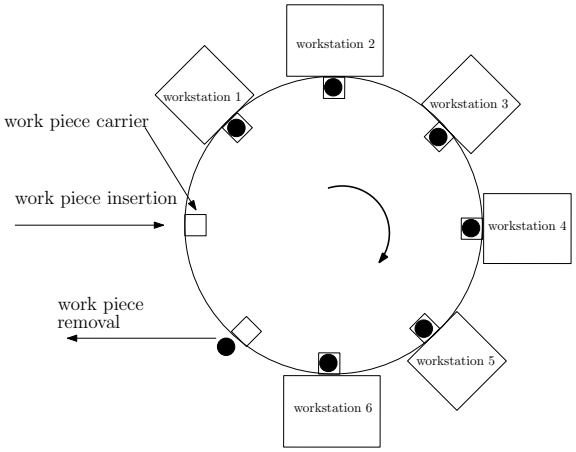


Figure 1.3: Rotary configuration

Groover states that "by comparison with the in-line and segmented in-line configurations, rotary indexing systems are commonly limited to smaller workparts and fewer workstations, and they cannot readily accommodate buffer storage capacity. On the positive side, the rotary system usually involves a less expensive piece of equipment and typically requires less floor space." (see Groover, 2007, p. 467-468).

It often occurs that certain steps of the production process take considerably longer time than others. In this case we will say that these steps dominate the process and refer to the corresponding workstations which are involved in these steps as dominating machines.

Transportation of products

It is integral to consider the underlying production system when scheduling the sequences in a flexible manufacturing environment that produces a high variety of products. This is especially true for mixed model lines in which processing times may vary a lot between variants. A very interesting aspect therein lies in the transportation between individual workstations. According to Boysen et al. (2008), movement of work pieces along an assembly line can be classified into paced and unpaced systems.

Within a paced system the conveyor belt moves at a constant rate between workstations. Machines or human workers either remove the work pieces from the conveyor system and replace them after their respective operation is completed, or they move along the conveyor belt and work on the product during movement. In contrast, within an unpaced production system work pieces are transported from one workstation to the next when processing of the product on the workstation is completed.

Unpaced production systems can be further classified into synchronous and asynchronous systems. In asynchronous systems, a product moves from one workstation to the next immediately after the operation on the former workstation is completed. However, transportation of a product to the next workstation may not be possible because the succeeding station still being blocked by another product. One possibility to solve this problem is to include buffers which, however, may not always be achievable due to space limitations.

In synchronous systems, all products may only advance from their current workstation to the next when operations on all workstations are completed. For example, this is the case when there is only one conveyor belt and products can not be removed from the conveyor system during the production process. Sequencing of paced and unpaced asynchronous assembly lines is a well discussed topic in the literature (see e.g. Boysen et al. (2008) and Boysen et al. (2009)). Also, most works on flow shops only consider cases with asynchronous transportation. On the other hand, unpaced synchronous assembly lines are considered rather scarcely. Within this thesis we will consider flow shop production systems with unpaced and synchronous movement.

In production processes with synchronous movement, dominating machines are of even greater importance as due to the nature of the system, movement only takes place as soon as all current operations are completed. In these instances, dominating machines determine the pace of the whole process. As a result, the complexity of scheduling the production process may decrease when dealing with machine dominance.

Thesis outline

This thesis presents a thorough introduction and analysis of scheduling of flow shops with synchronous movement. Despite a big prevalence of synchronous production lines in the industry, only very little work has been published on this topic. Furthermore, most results only cover the problem on a very elementary level or assume further constraints that are defined by the studied application. In this thesis flow shops with synchronous movement are systematically embedded into the flow shop scheduling framework. The problem is concisely defined for the most common objective functions as well as for many extensions

and additional constraints that can be observed in real world applications. The thesis offers an exhaustive study of complexity and settles the status of a big amount of the discussed problems. Several exact and heuristic solution algorithms are proposed and extensively evaluated.

The remainder of this thesis is structured as follows. In Chapter 2 we will give a formal description of flow shops with synchronous movement. The problem is defined and discussed for several objective functions, most notably the minimization of the makespan, the minimization of the maximum delay from due dates and the total completion time. Due to practical motivation, special cases depending on dominating machines and further additional constraints are investigated as well. In Chapter 3 the computational complexity to find optimal schedules for flow shops with synchronous movement considering various objective functions as well as the defined special cases are analyzed. Chapters 4 and 5 present exact and heuristic methods to find (near-) optimal solutions for flow shops with synchronous movement. Chapter 6 thoroughly describes a project in cooperation with a practitioner where flow shops with synchronous movement and resource constraints appear in a real world application and compares the implemented heuristic approach with the actual production of the industrial partner. Computational results for the approaches described in this thesis are presented in Chapter 7. Chapter 8 concludes the thesis, reciting the main results and stating open questions in the area of flow shops with synchronous movement.

Chapter 2

Problem description

In this chapter we will give a formal description of flow shops with synchronous movement and dominating machines. In Section 2.1 we will introduce classical flow shop scheduling based on Brucker (2007) and Pinedo (2012), describing the basic structure and the most prevalent constraints considered in the literature. Thereafter, in Section 2.1.1 we will focus on problems closely related to the ones considered within this work, describe the notion of dominating machines in Section 2.1.2 and discuss synchronous movement in Section 2.2. In Section 2.3 we will discuss the effects of dominating machines on flow shops with synchronous movement in more depth, classifying them depending on the number and position of the dominating machines. Finally, in Section 2.4 some extensions to flow shops with synchronous movements will be considered, particularly the effects of circular production systems and resource constraints.

2.1 Flow shop scheduling

A flow shop consists of m machines $M_1, \ldots, M_m$ and a set J of n jobs, where each job j consists of m operations $O_{1j}, O_{2j}, \ldots, O_{mj}$. Operation O_{ij} has to be processed without interruption on machine M_i for p_{ij} time units and operation O_{ij} may not start before operation $O_{i-1,j}$ is completed. A schedule of a flow shop consists of an allocation of jobs to machines in such a way that each machine processes at most one job at the same time and each job is processed on at most one machine at the same time. If at any given time a machine does not process a job, the machine is called idle. The time at which operations of a job j have been completed on all machines so that the job can leave the system is called its completion time C_j.

Graham et al. (1979) introduced an $\alpha|\beta|\gamma$-classification scheme for scheduling problems. The α-field specifies the machine environment and the number of machines, the β-field the underlying job and machine characteristics and the γ-field the objective function to be optimized. Within this scheme, a flow shop with characteristics β and objective function γ is described by $F|\beta|\gamma$. In the following, we will briefly discuss some of the more contemplated job characteristics. The characteristics of flow shops with synchronous movement will be discussed in Section 2.2.

- **Due dates and release dates.** For each job $j = 1, \ldots, n$ we may be given a due date d_j, i.e. a date until when the job should be completed. If a job is completed after its due date, the job is called late, otherwise it is called early. We define the lateness of a job j by $L_j = C_j - d_j$ and its tardiness by $T_j = \max(0, C_j - d_j)$. Further, we define a binary variable U_j, which is set to 1 iff the job is late. In addition, jobs may also have release dates r_j, indicating that operation O_{1j} may not start prior to this date r_j.

- **Permutation flow shop.** In a permutation flow shop the jobs have to processed in the same order on all machines. Then, a schedule can be represented by a permutation (or sequence) $\sigma = (\sigma_1, \ldots, \sigma_n)$ of the jobs in J. The schedule can be constructed by starting each operation O_{i,σ_k} as soon as possible after the completion of operation $O_{i,\sigma_{k-1}}$ when machine M_i is free. We refer to this schedule as the left-aligned schedule for the sequence σ.

- **Blocking and no-wait flow shop.** In a no-wait flow shop, for all jobs j operation O_{ij} must start immediately after operation $O_{i-1,j}$ is completed. Within a blocking flow shop no job may be processed on a machine M_i for $i < m$ if the preceding job has not been transferred to the next machine M_{i+1}. We will discuss blocking and no-wait flow shops in more detail in Section 2.1.1. Note that no-wait and blocking flow shops are permutation flow shops by definition.

- **No (machine) idle times.** This constraint prohibits idle time between consecutive operations on all machines. As soon as the operation O_{ij} starts on machine M_i for the first job j, the machine has to process all operations of all jobs for the next $\sum_j p_{ij}$ time periods.

- **Preemption.** In the default case, preemption of jobs is not allowed, i.e. once operation O_{ij} starts on machine M_i for job j, the job has to be processed on that machine for the total of p_{ij} time units. If preemption is allowed, the operation may be interrupted at any time and resumed later. While an operation is being preempted, a different operation may be executed on this machine.

Throughout this work we will restrict ourselves to permutation flow shops without preemption. Further, release dates will not be of importance and we will only consider due dates when discussing objective functions that take into account the lateness of individual jobs. Table 2.1 shows the objective functions that will be considered within this work as well as their corresponding notation used in the γ-field of the scheduling classification scheme.

2.1.1 Flow shops with blocking or no-wait constraints

In a no-wait flow shop, operation O_{ij} of a job $j \in J$ on machine M_i for $i > 1$ must start immediately after operation $O_{i-1,j}$ is completed. Examples for production processes with a no-wait constraint arise in metal and chemical processing where the material must be kept at a constant temperature and condition and thus the whole work processes must be

γ	optimization target
$C_{\max}$	Minimize the makespan of the schedule, i.e. the completion time of the last job to be processed.
$L_{\max}$	Minimize the maximum lateness $L_{max} = \max_j L_j$.
$\sum U_j$	Minimize the number of late jobs.
$\sum T_j$	Minimize the total tardiness (also called sum of tardiness or mean tardiness).
$\sum C_j$	Minimize the total completion time (also called sum of completion times or mean completion time). There may be a weight w_j defined for each job j, indicating its importance. In this case, objective functions $\sum w_j U_j, \sum w_j T_j, \sum w_j C_j$ may describe the minimization of the weighted sums, respectively.

Table 2.1: Considered objective functions for scheduling problems

executed without interruption. To achieve this, machine M_i must be able to process job j as soon as operation $O_{i-1,j}$ on machine M_{i-1} is completed for all $i > 1$. Thus, idle times on early machines may occur, as they may not start processing the next job because a succeeding machine would not be available in time. This situation is depicted in Figure 2.1: Operation $O_{1,4}$ of Job 4 could start on machine M_1 immediately after job 3 has been moved to the next machine, but this would lead to a conflict on machine M_3 because of the large processing time of job 3 on machine M_3. Thus, start of operation $O_{1,4}$ has to be delayed until it is made sure that all succeeding operations can be executed without waiting.

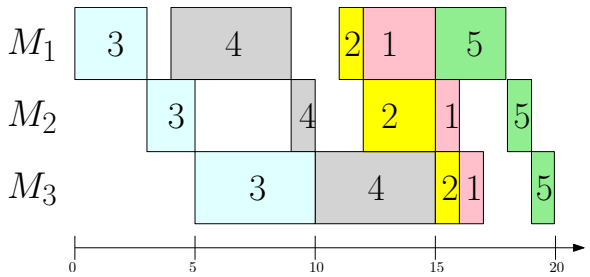


Figure 2.1: Schedule adhering to the no-wait constraint

Another case in which idle times may occur is when machines are blocked. In the default flow shop case an unlimited buffer is assumed between all pairs of machines and thus after an operation is completed on a machine, the respective job may be removed from the machine and stored until it can be processed on the next machine. The operation of the succeeding job may start immediately afterwards. However, in practice the space and thus the buffer may be limited and a job can only be transported to the next machine. If the preceding job is still being processed on the next machine, the machine is blocked

and the job can not be transported. This may lead to backlog, as the job that can not be transported now also blocks its current machine and so forth.

Example 2.1. *To demonstrate the notions of no-wait and blocking flow shops we consider the following example of a flow shop with three machines.*

j	1	2	3	4	5
p_{1j}	3	1	3	5	3
p_{2j}	1	3	2	1	1
p_{3j}	1	1	5	5	1

Figures 2.1 and 2.2 show optimal solutions to the problems $F3|$*no-wait*$|C_{\max}$ *and* $F3|$*blocking*$|C_{\max}$*, respectively. As can be seen, the schedules differ in the permutation of jobs and their objective values. While in the no-wait flow shop the permutation* $(3, 4, 2, 1, 5)$ *is best with a makespan of* 20*, within the flow shop without buffers the jobs can be scheduled optimally in order* $(2, 3, 4, 1, 5)$ *resulting in a makespan of* 18*.*

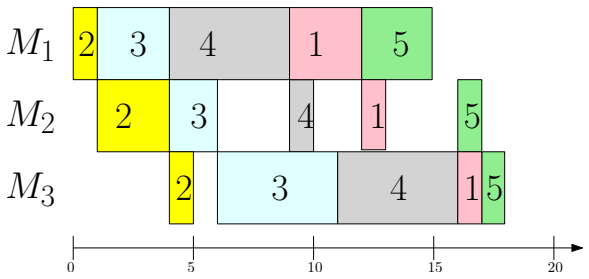


Figure 2.2: Schedule adhering to the blocking constraint

For only two machines, the blocking and no-wait constraints lead to equivalent problems, see e.g. Pinedo (2012). Flow shops with no-wait and blocking constraints have been an active topic in the literature for a long time, see e.g. Hall and Sriskandarajah (1996) for an early survey. Interesting cases also occur when the blocking or no-wait constraint is only active between two adjacent machines within a larger flow shop or when blocking and no-wait constraints are mixed.

2.1.2 Flow shops with dominating machines

In accordance to Adiri and Pohoryles (1982), Monma and Rinnooy Kan (1983) and Ho and Gupta (1995) we say that a machine M_k dominates M_l (denoted $M_k \cdot> M_l$) whenever

$$\min_j p_{kj} \geq \max_j p_{lj},$$

i.e. the smallest processing time of any job on machine M_k is at least as large as the largest processing time of any job on machine M_l. Adiri and Pohoryles further introduced

the notions of increasing and decreasing series of dominating machines, indicated by *idm* or *ddm* in the β-field of the classification scheme. In these cases for the m machines either

$$M_1 <\cdot M_2 <\cdot \cdots <\cdot M_m$$

or

$$M_1 \cdot> M_2 \cdot> \cdots \cdot> M_m$$

holds. Within their work, they discussed both series of dominating machines in flow shops with no-idle time and no-wait constraints. Čepek et al. (2002) showed that there exist polynomial time algorithms for any regular objective function for no-idle time and no-wait flow shops with an increasing or decreasing series of dominating machines. For classical flow shops without no-idle time or no-wait constraints, Ho and Gupta provided polynomial time algorithms for several objective functions when there is an increasing or decreasing series of dominating machines. Xiang et al. (2000) discussed series of dominating machines which are increasing-decreasing (idm-ddm) or decreasing-decreasing (ddm-idm), i.e. in which there is a machine M_k, such that

$$M_1 <\cdot M_2 <\cdot \cdots <\cdot M_{k-1} <\cdot M_k \cdot> M_{k+1} \cdot> \ldots \cdot> M_m$$

or

$$M_1 \cdot> M_2 \cdot> \cdots \cdot> M_{k-1} \cdot> M_k <\cdot M_{k+1} <\cdot \ldots <\cdot M_m$$

holds. Wang and Xia (2005) then discussed no-idle and no-wait flow shops with series of machines that are increasing-decreasing.

There exist various other notions of machine dominance in the literature. Additionally to the definition given above, Monma and Rinnooy Kan discuss variants in which machine M_k dominates M_l if there exists an integer q such that the sum of every q processing times on machine M_k is larger than the sum of every q processing times on machine M_l, which is a generalization that covers the former definition with the case $q = 1$. Van den Nouweland et al. (1992) call a machine M_k dominant if $\sum_{l=k}^{r} p_{lj} \geq \sum_{l=k}^{r} p_{(l+1)j}$ holds for $r = k, \ldots, m-1$ and $\sum_{l=r}^{k} p_{lj} \geq \sum_{l=r}^{k} p_{(l-1)j}$ holds for $r = 2, \ldots, k$. Čap et al. (2005) refer to this notion of dominance as "weak dominance" in comparison to the "strong dominance" of a dominating machine in an increasing-decreasing series of dominating machines in the definition above. They further showed that strong dominance always implies weak dominance.

Within this work, we generalize this definition of machine dominance. We call a set $\{M_i | i \in \mathcal{I}\}$ of machines dominating if

$$\min_{\substack{j \\ i \in \mathcal{I}}} p_{ij} \geq \max_{\substack{j \\ h \notin \mathcal{I}}} p_{hj},$$

i.e. the set of machines dominates all other machines but there is no further dominance within the set. If the set $\{M_i | i \in \mathcal{I}\}$ describes a set of dominating machines, we will use the notation 'dom($\mathcal{I}$)' in the β-field of the scheduling classification scheme. For short, we will also write 'dom $(k_1, \ldots, k_l)$' instead of 'dom $(\{k_1, \ldots, k_l\})$' to denote dominating machine with indices $k_1, \ldots, k_l$.

There are two possibilities for the processing times on non-dominating machines:

- The processing times on the non-dominating machines are arbitrary values, or
- the processing times on the non-dominating machines are job-independent, i.e. $p_{ij} = p_i$ for all j and all $i \notin \mathcal{I}$.

Job-independent processing times in the second case may be present in practice where non-dominating machines resemble work processes like insertion or removal of work pieces that have the same processing time regardless of the actual job. Even if times are not exactly equal, the difference between individual processing times might be negligible in practice and assuming a constant time may simplify the problem without causing a large error.

Other concepts of dominance that are different from the above definition can be found in the literature. Instead of stating that the processing times of a process on one machine are larger than the processing times of any other process on other machines, it may be that the processing time on one machine is the largest processing time of each job. Smith et al. (1975) discussed ordered flow shops, in which the processing times of all jobs are distributed the same way, i.e. for all r and for all jobs the r-th largest processing time is on the same machine and further, if for two jobs, i, j, the processing time of job i is smaller than the processing time of job j on one machine M_k, it is smaller on all machines. Achugbue and Chin (1982) proposed the less restrictive notion of j-minimal (j-maximal) jobs, in which the j-th processing time of a job is at most (at least) as large as the other processing times. While we do not focus on these concepts in this thesis, we include them here for completeness.

2.2 Flow shops with synchronous movement

A flow shop with synchronous movement is a variant of a non-preemptive permutation flow shop that was described in the previous section. All transfers of jobs from machines M_i to machine M_{i+1} for $i = 1, \ldots, m-1$ take place at the same time. Therefore, an operation of a job on the next machine may only start after all current operations of the jobs are completed on all machines, i.e. after the maximal processing time of the jobs that are currently processed. If the processing time for a job on a certain machine is smaller than this maximum, the corresponding machine is idle until the job may be moved to the next machine. In the following we will use the notions "flow shop with synchronous movement" and "synchronous flow shop" interchangeably.

Example 2.2. *Let us revisit Example 2.1 with three machines and five jobs:*

j	1	2	3	4	5
p_{1j}	3	1	3	5	3
p_{2j}	1	3	2	1	1
p_{3j}	1	1	5	5	1

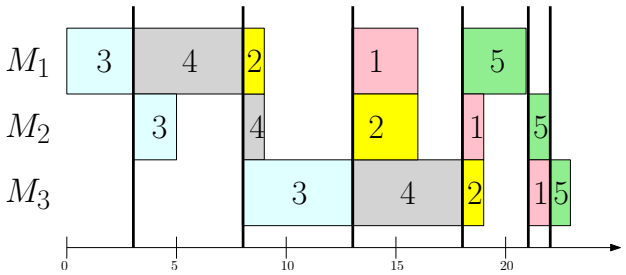


Figure 2.3: Example of a flow shop with synchronous movement

Figure 2.3 shows a schedule for the sequence $(3, 4, 2, 1, 5)$*. The vertical lines show when transfer of jobs to the next station takes place. Because of the synchronous movement constraint, the third operation of job* 3 *can not start immediately after the completion of the second operation as the job is only transfered to machine* M_3 *when the first operation of job* 4 *is completed on machine* M_1*. Similarly, there is a large idle time on machine* M_1 *after the completion of job* 2 *on machine* M_1 *as all transfers have to wait until the completion of job* 3 *on machine* M_3*. Figure 2.4 shows a schedule for the permutation* $(3, 1, 4, 2, 5)$ *with an optimal makespan of* 19*.*

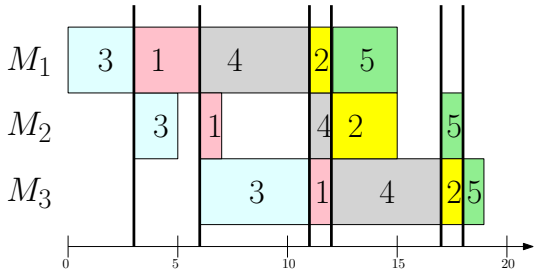


Figure 2.4: A schedule with optimal makespan

Flow shop scheduling with synchronous movement was first introduced by Soylu et al. (2007). The authors discussed complexity results and presented approaches to minimize the makespan. In Huang and Hung (2010) and Huang (2008), the notation 'synmv' in the β-field of the scheduling classification scheme was introduced to indicate synchronous movement in flow shops. In line with prior work we will refer to the time period between

two synchronous movements as a cycle of the synchronous flow shop and index the cycles starting from cycle 1 (when the first job is processed on machine M_1) to cycle $n + m - 1$ (when the last job is processed on machine M_m). There exist two possibilities of when to call a job completed, either when the cycle in which the job is processed on the last machine is completed or alternatively after its processing on the last machine is completed (thus ignoring the idle time that might occur caused by the jobs on other machines in this cycle). The exact completion time depends on the construction of the assembly system and whether the completed job can be removed immediately from the last machine or whether it can only be accessed after the cycle has completed. Within this work we will consider the former case and consider a job completed at the end of the respective cycle. Note that we can always transform an assembly system in which the jobs are completed as soon as their operation on the last machine is finished into one in which the cycle has to be completed by adding a virtual machine with processing time of 0 after the last machine.

In synchronous flow shops, jobs are often transported with the help of work piece carriers or are attached to fixtures on the conveyor system. In some cases, the number of work piece carriers or fixtures (and thus the number of jobs transported simultaneously) may exceed the number of operations that need to be executed for each job. Consider a production system with a conveyor belt that can hold four jobs but with only two workstations at the beginning and the end of the conveyor belt, respectively (cf. Figure 2.5). After the job is processed on the first workstation it has to be moved to the second one. However, it takes two more cycles until the job reaches the second workstation and is processed there.

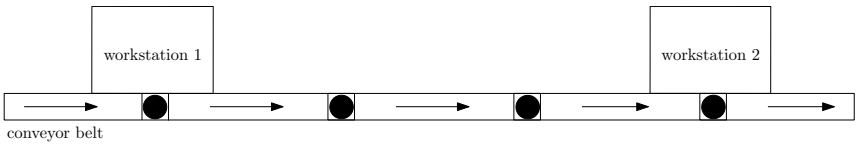


Figure 2.5: Assembly line with two workstations and two cycles between the two operations

In the production system depicted in Figure 2.5, the second operation of a job j takes place at the same time as the first operation of the third-next job in the sequence. Thus, the time required for a job to be transported from workstation 1 to workstation 2 depends on the time of the next two cycles. To incorporate this case into our notation we can perceive the intermediary cycles as two additional virtual transportation operations of the job and as moving through two additional machines on which processing times are equal to zero. Within our framework workstation 1 would be denoted as machine M_1, workstation 2 as machine M_4 and the two intermediary transport operations as the virtual machines M_2 and M_3, each with a processing time of zero. Without loss of generality, within this work we will thus only consider production units in which the number of work piece carriers or fixtures which are moved synchronously equals the number of machines.

In the next section we will discuss the effects of machine dominance on flow shops with synchronous movement. Examples of dominated machines (or operations) may be removal

of jobs from the production unit or the virtual transportation operations discussed in the previous paragraph.

2.3 Flow shops with synchronous movement and dominating machines

Machine dominance leads to a simplification of the problem. As soon as the first job is processed on the first dominating machine, the processing times on all non-dominating machines become irrelevant and we only have to consider the dominating machines. This holds true until the last job is moved from the last dominating machine. In Figure 2.6 a schedule for a synchronous flow shop with only one dominating machine M_k and the sequence $(1, 2, \ldots, n)$ is shown. For the first $k-1$ cycles (before job 1 is processed on M_k) and the last $m-k$ cycles (after job n is processed on M_k), no job is processed on the dominating machine. Thus, for these cycles no further machine dominance occurs and the situation is the same as for a general synchronous flow shop without machine dominance.

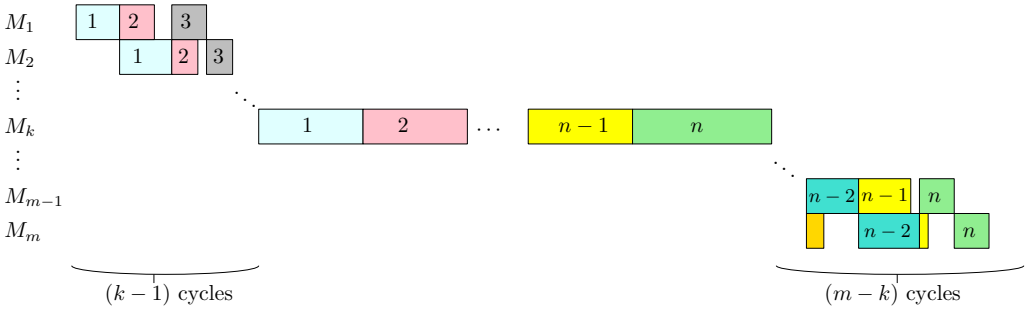


Figure 2.6: A synchronous flow shop with one dominating machine M_k

In the case of job-independent processing times, the time required in cycles where no job is processed on a dominating machine is constant independent of the job sequence. We can thus simply ignore the cycles in which there is no job being processed on a dominating machine and add a constant to all completion times. As the constant to be added does not depend on the sequence of the jobs and the value of the constant does not affect an optimal sequence, in this situation we may even assume that all processing times on the non-dominating machines are the same and equal to zero. To denote this special situation, we add '$p_{ij}^{ndom} = 0$' to the β-field.

In case of arbitrary processing times on the non-dominating machines, if the first dominating machine has index k_1 and the last dominating machine has index k_2, we can iterate over all possible sequences of the first $k_1 - 1$ and the last $m - k_2$ jobs. For all jobs in between, the processing times on the non-dominating machines are not relevant as a job is processed on at least one dominating machine in each of these cycles. Therefore, after fixing the first $k_1 - 1$ and last $m - k_2$ jobs, a reduced scheduling problem for the remaining jobs can be solved assuming zero processing times on the non-dominating machines as explained above. Afterwards, the fixed jobs are added to the front and the end of an

optimal sequence of the reduced problem. Finally, the completion times of the jobs and the objective value can be calculated from the obtained sequence.

There are $\mathcal{O}(n^{m-k_2+k_1-1})$ possibilities to fix the job sequences in the first $k_1 - 1$ and the last $m - k_2$ cycles. Thus, if a polynomial time algorithm exists to solve the reduced problem with zero processing times on the non-dominating machines, the problem with arbitrary processing times can also be solved in polynomial time by trying all possible sequences for the first and last cycles if the number of machines m is fixed.

Within this work we will classify flow shops with synchronous movement depending on the number and positions of dominating machines. With only one or two dominating machines, the flow shop with synchronous movement is closely related to other known scheduling problems. We will look into this in the following sections and describe similarities and differences. In Section 2.3.1 the case with only one dominating machine is compared to scheduling problems with only a single machine. The main focus lies on the difference of completion times within the two problems. In Section 2.3.2 we describe similarities between synchronous flow shops with two dominating machines and the two-machine no-wait flow shop.

2.3.1 Synchronous movement with one dominating machine

If in a m-machine flow shop there is only one dominating machine as seen in Figure 2.6, the synchronous flow shop $F|\text{synmv}, \text{dom}\,(\mathcal{I})\,|\gamma$ with $|\mathcal{I}| = 1$ is closely related to the single machine problem $1||\gamma$. Given an instance of a single machine problem with jobs $j = 1, \ldots, n$ and processing times p_j, we can transform it into a synchronous flow shop problem with the same jobs by setting their processing times on the dominating machine to p_j and the processing times on all other machines to zero. If $C^1_{\max}$ is the makespan of a sequence of jobs on the single machine, the synchronous flow shop has the same makespan $C^S_{\max} = C^1_{\max}$. Further, if the dominating machine is the last machine and the processing times on the dominated machines are zero, the individual completion times of all jobs do not differ from the completion times in the single machine problem. Thus, for $|\mathcal{I}| = 1$ the problems $F|\text{synmv}, \text{dom}\,(\mathcal{I})\,, p^{ndom}_{ij} = 0|\gamma$ and $1||\gamma$ are equivalent if the dominating machine is the last machine and processing times on the dominated machines are zero.

On the other hand, if the dominating machine is not the last machine, the problems are no longer equivalent. The completion time of the last job in a sequence and thus the makespan are still the same when the processing time on the dominated machines are zero. However, the completion times of other jobs within the sequence cannot be determined that easily. Let C^1_j be the completion time of job j in the single machine problem. Consider a synchronous flow shop with a single dominating machine M_k . Then, the end of the processing on machine M_k for a job j is equal to C^1_j in the synchronous flow shop, but the completion time C^S_j of job j in the synchronous flow shop is determined by the length of the next $m - k$ cycles, i.e. the processing times of the next $m - k$ jobs in the sequence. Figure 2.7 depicts this situation in a synchronous flow shop with six machines where machine M_4 is dominating and the processing times on the dominated machines are zero. While the completion time C^1_2 of job 2 in the corresponding single machine problem is equivalent to the completion time of the cycle in which job 2 is processed on the dominating machine,

its completion time in the synchronous flow shop is two cycles later. Thus, the completion time C_2^S differs from C_2^1 by the sum of the processing times of jobs 3 and 4. In Section 3.3.1 we will discuss whether this makes any difference for schedules when the objective function is minimizing the number of late jobs or the total completion time.

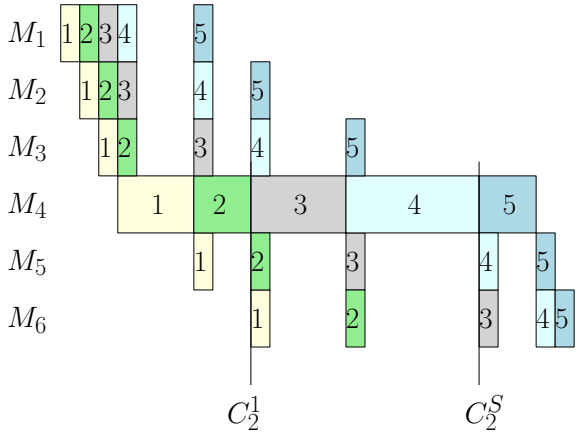


Figure 2.7: Delayed completion in flow shops with additional non-dominating machines after a dominating machine

2.3.2 Synchronous movement with two dominating machines

If there are only two machines in total, the constraints of no-wait, blocking and synchronous movement are equivalent, see e.g. Soylu et al. (2007). A schedule that satisfies the blocking constraint also satisfies the synchronous movement constraint and vice versa; the no-wait constraints are satisfied when we shift the jobs on the first machine to the right for each cycle (cf. Figure 2.8). On the other hand, the completion times of all jobs are identical when comparing flow shops with blocking and no-wait constraints. However, since we consider a job completing at the end of the cycle in the flow shop with synchronous movement, the individual completion time of each job is in general neither equal to its completion time in the no-wait or the blocking flow shop. However, the completion time of the last job is equal in all three cases and thus an optimal solution for either of the problems

$$F2|\text{no-wait}|C_{\max},\ F2|\text{blocking}|C_{\max}\ \text{ and }\ F2|\text{synmv}|C_{\max}$$

is an optimal solution for the two other problems as well.

Further, in the case of two adjacent dominating machines with indices $k_1, k_2 = k_1 + 1$ and zero times on the dominated machines, the problems $F|\text{synmv}, \text{dom}\,(k_1, k_2)\,, p_{ij}^{ndom} = 0|C_{\max}$, $F2|\text{no-wait}|C_{\max}$ and $F2|\text{blocking}|C_{\max}$ are still equivalent. Again, the individual completion times for all jobs except the last depend on the succeeding jobs in the sequence and can thus not be calculated this easily, similar to the case with only one dominating machine.

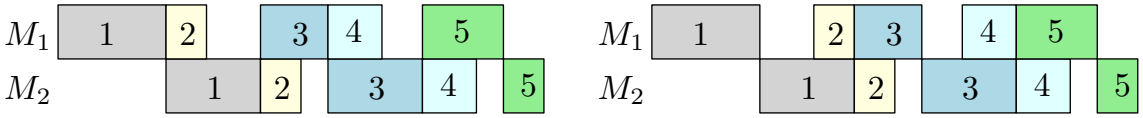


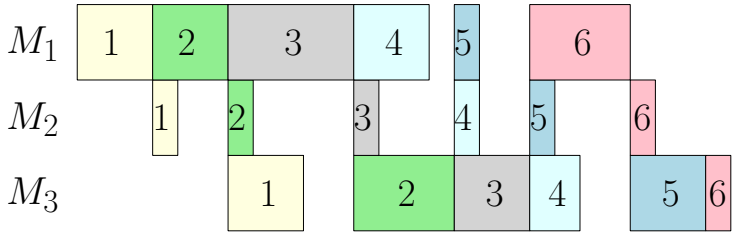
Figure 2.8: Transformation of a schedule of the synchronous flow shop into a no-wait schedule by shifting the jobs on the first machine to the right

If the two dominating machines M_{k_1} and M_{k_2} are not adjacent, there is no more equivalence between the synchronous flow shop and the flow shop with no-wait constraint. In this case an interesting effect occurs. Let $1 \leq k_1 < k_2 \leq m$ be the indices of the dominating machines and $\sigma = (\sigma_1, \ldots, \sigma_n)$ be a sequence of jobs. Because the length of one cycle in the synchronous flow shop is determined by the maximum of the processing times on those two dominating machines, the sequence splits into $k_2 - k_1$ independent subsequences σ^λ for $\lambda = 1, \ldots, k_2 - k_1$ with

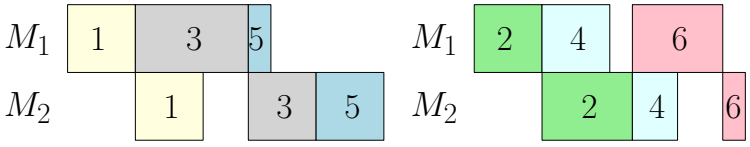
$$\sigma^\lambda = \sigma_{\lambda+\mu(k_2-k_1)} \text{ for } \mu = 0, \ldots, \left\lfloor \frac{n-\lambda}{k_2-k_1} \right\rfloor .$$

These subsequences affect the lengths of disjoint cycles of the schedule and the makespan of the whole schedule is the sum of the makespans of these subsequences, i.e. $C_{\max}(\sigma) = \sum_\lambda C_{\max}\left(\sigma^\lambda\right)$. For example, for the synchronous flow shop $F3|\text{synmv}, \text{dom}\,(1,3)\,|f$ in Figure 2.9 we have two subsequences $(1,3,5)$ and $(2,4,6)$. The idle times on the dominating machines M_1 and M_3 for each job are only caused by the preceding (or succeeding) jobs of the same subsequence. Figure 2.9(a) depicts the original schedule while Figure 2.9(b) illustrates the equivalence of the three machine schedule to the corresponding two-machine schedule of the two subsequences $(1,3,5)$ and $(2,4,6)$.

In the case of two dominating machines, M_{k_1}, M_{k_2}, the problem can also be described in form of a routing problem in a directed graph: Each job corresponds to a node in the graph and the distance c_{ij} between two nodes corresponding to jobs i, j is defined by the cycle time that would occur in a cycle when job i is processed on machine M_{k_2} and job j is processed on machine M_{k_1}, i.e. the cycle time if the two jobs succeed each other in a subsequence. To model the cycle times for cycles in which only one of the dominating machine processes a job, i.e. the beginning and end of a subsequence, another node 0 is introduced which represents a job with processing times zero on all machines. The distance between node 0 and another node corresponding to job j is determined by the cycle time that occurs when j is processed on machine M_{k_1} (or M_{k_2}, respectively) and no job is processed on the other dominating machine. Then, each subsequence σ^λ relates to a tour in the graph that starts in 0 and then visits all nodes that correspond to the respective jobs contained in the subsequence. If the two dominating machines are adjacent, this resembles a single tour. Thus, finding an optimal schedule for a synchronous flow shop with two adjacent dominating machines is equivalent to the well known traveling salesman problem with the described cost structure. If the two dominating machines M_{k_1}, M_{k_2} are not adjacent, we are interested in finding $k_2 - k_1$ tours of length $n/(k_2 - k_1)$ each (or, if



(a) Schedule for $F3|synmv, dom(1,3)|C_{\max}$



(b) Corresponding schedules of the subsequences for $F2|synmv|C_{\max}$

Figure 2.9: Flow shop with synchronous movement in which machines M_1 and M_3 are dominating

$n \bmod (k_2 - k_1) = l \neq 0$, in finding l tours of size $\lceil n/(k_2 - k_1) \rceil$ and $(k_2 - k_1) - l$ tours of size $\lfloor n/(k_2 - k_1) \rfloor$). This resembles a special case of the vehicle routing problem. Figure 2.10 shows the two tours that correspond to the schedule depicted in Figure 2.9.

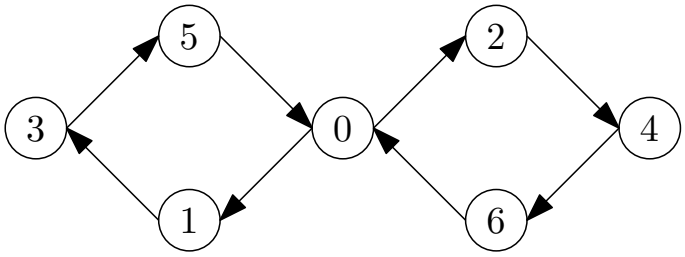


Figure 2.10: Tour representation of the schedule depicted in Figure 2.9

By regarding the scheduling problem as a routing problem it is possible to use approaches that have been studied extensively for many years.

2.4 Possible extensions

The previous sections introduced the fundamental aspects of a flow shop with synchronous movement. In this section we will look into extensions and alterations that often occur in real world applications.

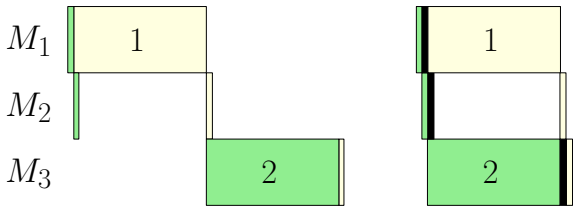


Figure 2.11: Insertion of the black job with small processing times improves the makespan of the schedule

2.4.1 Idle jobs

An interesting effect recognized by Karabati and Sayin (2003) is that within synchronous flow shops the makespan of a schedule can sometimes be improved by adding jobs with small or zero processing time on all machines (or similarly introducing gaps into the schedule where no job is processed on a machine). While this may seem counterintuitive at first, it is well justified by the characteristics of synchronous flow shops: Consider a cycle in which all jobs that are currently processed on machines $M_2, \ldots, M_m$ have a very small processing time within this cycle. Inserting a job with large processing time into machine M_1 leads to a large amount of idle time on all other machines. If we instead opt to insert no job into machine M_1, the cycle time is determined by the maxima of the processing times of the other jobs, which might be much smaller. Figure 2.11 shows an example for three machines with two jobs $1, 2$ and processing times $p_{11} = p_{32} = 1$ and zero processing times otherwise. While we achieve a makespan of 2 with both possible schedules, introducing an idle job with zero processing times on all machines allows us to decrease the makespan to 1. We denote problems in which the introduction of idle jobs is allowed with 'idle' in the β-field of the classification scheme.

For a given instance the maximal number of idle jobs for which the solution can be improved might be of interest. An upper bound is $(n-1)(m-1)$, in which case each actual job except for the last one is succeeded by $(m-1)$ idle jobs. Any additional idle job would either result in a subsequence of m successive idle jobs (leading to no gain as this results in a cycle in which only idle jobs are processed) or be added to the end or beginning of the schedule, both not leading to any improvement.

Example 2.3. *The following instance of* $F|synmv, dom(1)|L_{\max}$ *is an example in which* $(n-1)(m-1)$ *idle jobs are necessary to improve the solution. Let jobs* $j = 1, \ldots, n$ *have a processing time of* $p_{1j} = j, p_{ij} = 0$ *for all* $i = 2, \ldots, m$*. Further, set their due dates to* $d_j = \sum_{k=1}^{j} k$*. Then, the only way to achieve a schedule without any late job is to process the jobs sorted in non-decreasing order of their due dates and to let each job except the last one be followed by* $m-1$ *idle jobs with processing time of zero on all machines such that each job is completed right after it is processed on the first machine.*

This bound can be improved to $(n-1)(m-2)$ for the objective function of minimizing the makespan, in which case each actual job except for the last one is succeeded by $(m-2)$ idle jobs. Consider a subsequence of $m-1$ consecutive idle jobs scheduled between jobs i

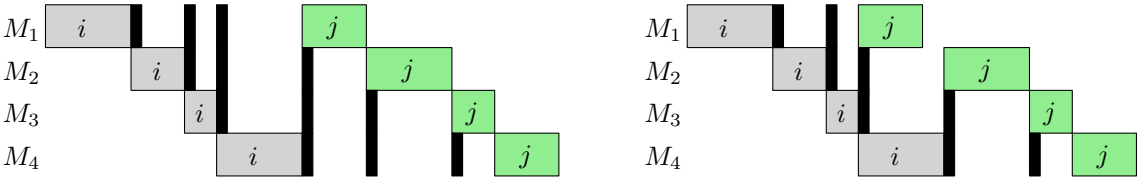


Figure 2.12: Example of idle jobs in a flow shop with four machines. Removing one of the three consecutive idle jobs between jobs i and j does not increase the makespan of the schedule

and j. Let t be the cycle in which operation O_{mi} is processed. As i is followed by $m-1$ idle jobs, there is no further operation being processed in cycle t. Further, operation of O_{1j} is the only operation being processed in cycle $t+1$ as it is preceded by $m-1$ idle jobs. Thus, both cycle times sum up to $p_{mi}+p_{1j}$. Removing one of the $m-1$ consecutive idle jobs improves the makespan of the schedule by $p_{mi}+p_{1j}-\max(p_{mi},p_{1j}) \geq 0$ as it leads to both operations being processed in cycle t. Thus, removing one idle job does not increase the makespan. An example is depicted in Figure 2.12.

2.4.2 Job splitting

In some production processes it may be possible to split operations across multiple machines. This might be the case if two adjacent machines are equipped to do the same operation and the operation can be interrupted at some point. Similarly, machinery as well as human workforce may be allowed to move alongside the conveyor belt and perform some operations. To model this, instead of processing times p_{ij} for each machine and job we may be given combined processing times $p_{[i_1,i_2],j}$ which define the processing time of job j that has to be split across machines $M_{i_1},\ldots,M_{i_2}$. In the case $i=i_1=i_2$ we will use the standard notion $p_{ij}=p_{[i_1,i_1],j}$.

Additionally, boundaries $\left[\underline{p}_{ij},\overline{p}_{ij}\right]$ may define lower and upper bounds for processing job j on machine M_i. The jobs have to be scheduled and split in such a way that for the resulting actual processing time $\tilde{p}_{ij}$ in the schedule, both, $\underline{p}_{ij} \leq \tilde{p}_{ij} \leq \overline{p}_{ij}$ and $\sum_{i=i_1}^{i_2} \tilde{p}_{ij} = p_{[i_1,i_2],j}$ hold. We will call the lower bound $\underline{p}_{ij}$ the mandatory part of j that has to be processed on M_i. If $\underline{p}_{ij}=\overline{p}_{ij}$ holds, this resembles the classical case where the processing time on the machine is fixed. Obviously, the flow shop without job splitting can be modeled as a flow shop with job splitting by setting all mandatory times to the respective processing times.

Example 2.4. *We reconsider the example used in the previous sections, but this time allow arbitrary job splitting between machines M_2 and M_3 and a fixed processing time on machine M_1.*

j	1	2	3	4	5
p_{1j}	3	1	3	5	3
$p_{[2,3],j}$	2	4	7	6	2
$\underline{p}_{2j}$	0	0	0	0	0
$\overline{p}_{2j}$	2	4	7	6	2
$\underline{p}_{3j}$	0	0	0	0	0
$\overline{p}_{3j}$	2	4	7	6	2

An optimal schedule is depicted in Figure 2.13 with a makespan of 17*. Note that the sequence* $(3, 1, 4, 2, 5)$*, which is optimal for the synchronous flow shop without job splitting in Example 2.2 can not be improved if job splitting between machines M_2 and M_3 is allowed and leads to a makespan of* 19 *in comparison to the makespan of* 17 *of an optimal schedule for the problem with job splitting.*

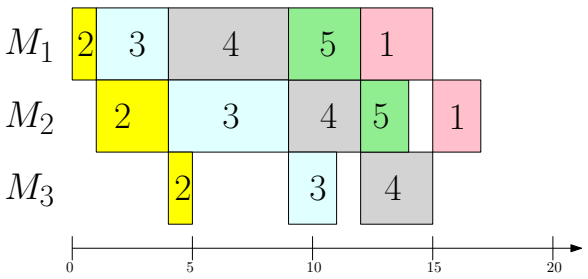


Figure 2.13: An optimal solution for the synchronous flow shop with job splitting between machines M_2 and M_3

We will use the following additional entries in the β-field of the scheduling classification:

- The notation 'split' refers to problems with job splitting in which there exist no mandatory parts. In this case, we are given only one combined processing time $p_j = p_{[1,m],j}$ and the jobs can be arbitrarily split across the machines such that for the resulting actual processing time $\tilde{p}_{ij}$ in the schedule $\sum_{i=1}^{m} \tilde{p}_{ij} = p_j$ holds.
- By contrast, 'r-split' ("restricted splitting") refers to problems with job splitting in which there exist mandatory parts.

2.4.3 Resources

We may consider renewable resources that are needed for processing of the jobs, e.g. tools required for assembly. Blazewicz et al. (1983) discuss production processes in which certain operations need additional renewable resources. If an operation O_{ij} of job j needs a resource, that resource is blocked for all other jobs while this operation is processed. They introduced the notation 'res$\lambda\sigma\rho$' in the β-field of the scheduling classification, where

- $\lambda \in \mathbb{N}$ indicates that the number of resources is constant and equal to λ. If $\lambda = \cdot$, the number of resources is part of the input.
- $\sigma \in \mathbb{N}$ indicates that the capacity of all resources is constant and equal to σ. If $\sigma = \cdot$, all resource capacities are part of the input.
- $\rho \in \mathbb{N}$ is an upper bound on the resource requirements for each job and resource. If $\rho = \cdot$, no bounds are specified.

In synchronous flow shops, it might occur that it is not possible to start processing of a job because a resource needed by an operation of the job might also be required by preceding jobs on other machines in the same cycle. In these cases, it may once again be necessary to insert idle jobs at machine M_1 as discussed in Section 2.4.1.

Another concept of resource constraints in flow shops are pallet resources introduced by Sixiang et al. (2002). Therein, a number of pallets is given and each job requires a pallet during its entire production process. After a job is completed, the pallet may be used by a new job. If no pallet is available at any given time, no job may start on the first machine. In their work, Sixiang et al. only considered pallet resources of a single type. For this case, within the synchronous flow shop, a pallet is available at every time if the number of pallets is at least as large as the number of machines. Then, the presence of pallets has no effect on the problem. On the other hand, if the number of pallets is smaller than the number of machines, we are forced to insert idle jobs (provided that they do not require a pallet) to achieve a feasible solution. We may consider problems in which pallet-type resources of distinct types exist and each job may only be processed with the help of a subset of these types. For this, we will combine the two concepts of resources and introduce the notation 'jres$\lambda\sigma\rho$' to the β-field of the scheduling classification scheme where λ, σ and ρ are defined as above. However, in contrast to 'res', distinct resources are not required for individual operations but they are required during the whole production process of a job until the job is completed.

2.4.4 Circular production and changeover times

In a circular production unit (like a rotary table, see Figure 2.14) the m machines are placed around a circular conveyor system, on which fixtures are attached that can accommodate work piece carriers or allow for the fixation of jobs on the transport system. In the following, we refer to each such fixture on which a job can be placed for processing as a station of the production unit. Additionally, the stations need to be distinctively configured depending on the job that is to be transported, e.g. using specialized work piece carriers, fixture

configurations or job specific equipment required for the production. Further, the jobs may form job families that require the same configuration.

The jobs to be processed are placed on stations which rotate (counter-) clockwise on the circular production unit and thereby transport the corresponding job to the next machine. The circular production unit rotates after each cycle, therefore the stations are back at their starting position after S cycles, where S is the number of stations. The configuration of the station has to be changed whenever a job of a different family from the preceding job on this station is to be processed. Then, a changeover time may occur during which the circular production unit may not rotate. Because of work safety constraints, no operation on any machine of the circular production unit may be processed during the changeover. In this case, production on all machines only starts after the changeover is complete. Similar to the discussion in Section 2.2, we will define virtual machines with processing times of zero if the number of stations exceeds the number of machines. In the following, we will assume that the number of stations is equal to the number of machines.

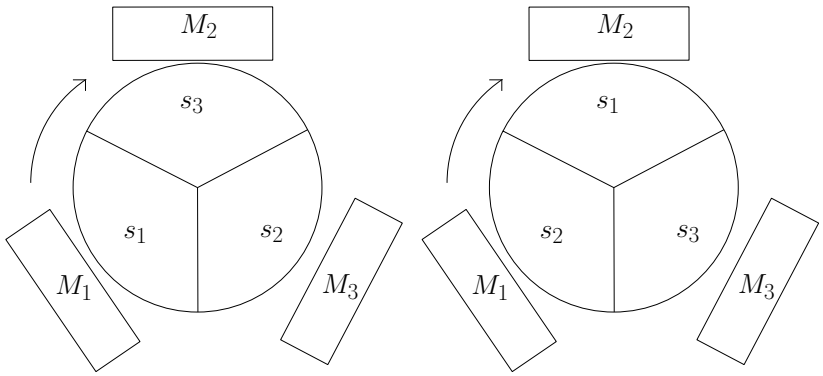


Figure 2.14: Exemplary configuration of a circular production unit with three machines and three stations. The picture to the left shows the initial positions of the stations, the picture to the right the positions of the stations after one cycle

Example 2.5. *We revisit Example 2.1 once more to show the impact of a circular production unit with changeover times on the schedule. We introduce job families to the example and define a changeover time of* 3 *between jobs that are of different families:*

j	1	2	3	4	5
p_{1j}	3	1	3	5	3
p_{2j}	1	3	2	1	1
p_{3j}	1	1	5	5	1
family	1	1	2	1	2

Then, the optimal solution determined in Section 2.2 is no longer optimal in the situation with circular production units because two changeovers are necessary after completion of jobs 3 *and* 1*, respectively (Figure 2.15). An optimal sequence in this situation is given by* $(3, 2, 4, 5, 1)$*, which needs no additional changeovers, see Figure 2.16.*

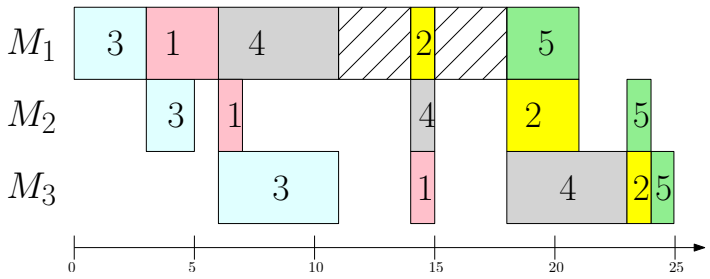


Figure 2.15: Two changeovers are necessary in the previously optimal sequence

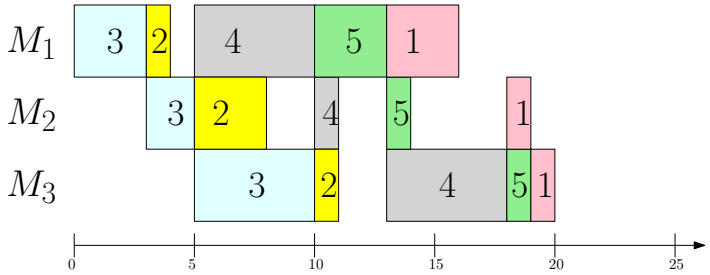


Figure 2.16: Optimal solution without changeovers

Huang (2008) discussed flow shops with synchronous movement on rotary tables. In the system described therein he considered a rotary table with a loading / unloading unit in which all jobs enter and leave the production system. Changeover times and job types are not considered, but the jobs have associated loading and unloading times and in each cycle, a job has to be unloaded first before the next job can be loaded. As the loading and unloading operation is executed on the same unit, Huang indicates this characteristic with the notation *re-LU* (i.e. re-entrance at the L/U station) in the β-field. In this work we consider cyclic production units where there are distinct machines for loading and unloading jobs.

To indicate a circular production unit in which changeovers occur, we will use the notation *circ-*s_{fg} in the β-field, where s_{fg} denotes the changeover time that occurs when changing configurations from job family f to job family g. We may consider cases in which all changeover times are constant (circ-$s_{fg} = s$) or only depend on the succeeding job family (circ-$s_{fg} = s_g$). The example given above on the production system pictured in Figure 2.14 can be written in our notation as

$$F3|\text{synmv}, \text{circ-}s_{fg}|C_{\max}.$$

Note, that without the presence of job families and changeover times, the circular nature of the production unit is of no importance. In this case, we will not specify the circular structure of the production unit in the β-field.

2.4.5 Order scheduling

In some applications several jobs may belong together, e.g. this might be the case if customers order several items of a product. In this case we consider orders Θ_o for $o = 1, \ldots, n$ where each order Θ_o has an associated product type t_o and a volume v_o indicating the number of times t_o has to be assembled to fulfill the order. For some problems we can transform the orders into a series of jobs

$$\Theta_{1,1}, \ldots, \Theta_{1,v_1}, O_{2,1}, \ldots, \Theta_{2,v_2}, \ldots, \Theta_{n,1}, \ldots, \Theta_{n,v_n}$$

and just solve the corresponding problem without orders. This is the case when considering the objective functions of minimizing the makespan $C_{\max}$ or minimizing the maximum lateness $L_{\max}$. However, if we are interested in minimizing the number of late orders $\sum U_o$ or the total completion time of the orders $\sum C_o$, the problems change.

Hitherto, order scheduling was only investigated for parallel machines, e.g. in Blocher and Chhajed (1996), denoted as the customer order lead-time problem (COLT), in Leung et al. (2007) where the total weighted completion time on uniform machines is considered or in Correa et al. (2009) where a case with preemption and unrelated machines is discussed. Leung et al. (2005) give on overview on order scheduling models. We will only consider this extension in Chapter 6 when discussing a real world application in furniture manufacturing.

Chapter 3

Complexity

In this chapter we will discuss the complexity status of synchronous flow shop problems. In Section 3.1 we will recite results for classical flow shop problems. Thereafter, we discuss the complexity of the general synchronous flow shop with three or more machines in Section 3.2 and of synchronous flow shops with dominating machines in Section 3.3. Section 3.4 deals with the complexity of various extensions of the synchronous flow shop. Finally, a concise summary of this chapter's results is presented in Section 3.5. Parts of this chapter have already been published in Waldherr and Knust (2015). For a thorough introduction into complexity theory we refer the reader to Garey and Johnson (1979).

3.1 Classical flow shop

For objective functions that are considered in this thesis there exist elementary reductions as discussed by Brucker (2007) (see Figure 3.1). An arrow $\gamma_1 \to \gamma_2$ between two objective functions γ_1 and γ_2 indicates that for a scheduling problem $\alpha|\beta|\gamma_1$ there exists a polynomial reduction to the scheduling problem $\alpha|\beta|\gamma_2$, i.e. $\alpha|\beta|\gamma_1 \propto \alpha|\beta|\gamma_2$, where α and β are arbitrary specifications identical for both problems. Further, for the classical flow shop without further constraints, $Fm||\gamma \propto F(m+1)||\gamma$ holds for all objective functions γ.

For classical flow shops, the problem $F2||C_{\max}$ is solvable in polynomial time (see Johnson (1954)) while $F2||L_{\max}$ and $F2||\sum C_j$ in the two-machine case, as well as the problem $F3||C_{\max}$ in the three-machine case are proven to be $\mathcal{NP}$-hard (see Garey et al. (1976), Lenstra et al. (1977)). In accordance to the elementary reductions we can derive from these results that the classical flow shop problem is $\mathcal{NP}$-hard for all objective functions depicted in Figure 3.1 for two or more machines except for the problem $F2||C_{\max}$.

3.2 Synchronous flow shops

As already observed in Soylu et al. (2007), the two-machine synchronous flow shop problem is closely related to the corresponding two-machine flow shop problem with no-wait or blocking constraints. All problems are equivalent to a special case of the traveling salesman

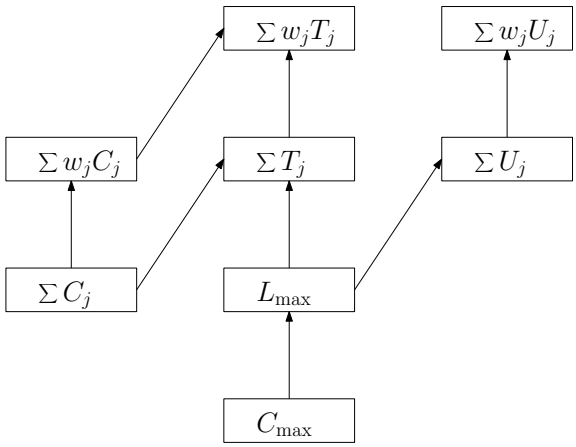


Figure 3.1: Elementary reductions for objective functions (cf. Brucker (2007) p.50)

problem, which can be solved in $\mathcal{O}(n \log n)$ by the algorithm of Gilmore and Gomory (1964). To keep this thesis self-contained the algorithm will be presented in Section 4.1.

Because we defined the completion time of a job within the synchronous flow shop as the completion time of the corresponding cycle, the individual completion times of the jobs in a synchronous flow shop are in general not identical to the completion times of the jobs under the no-wait or the blocking constraints, see Chapter 2. Hence, for other objective functions, in general we get different objective values for the three variants. Röck (1984a) proved that $F2|\text{no-wait}|\sum C_j$ and $F2|\text{no-wait}|L_{\max}$ are strongly $\mathcal{NP}$-hard. Within his proof, schedules without idle times on machines M_1 and M_2 are generated. If no idle times exist on the second machine, the completion times of the jobs are the same for the no-wait and the synchronous case. Thus, the results of Röck (1984a) can be used to show that $F2|\text{synmv}|\sum C_j$ and $F2|\text{synmv}|L_{\max}$ are strongly $\mathcal{NP}$-hard as well. In Section 3.3.2 we will extend the proof of Röck to provide a slightly more general result which shows that the problem $Fm|\text{synmv}, \text{dom}\,(\mathcal{I})\,|L_{\max}$ is $\mathcal{NP}$-hard for each fixed $m \geq 2$ and each set $\mathcal{I}$ with $|\mathcal{I}| = 2$.

$F3|\text{no-wait}|C_{\max}$ and $F3|\text{blocking}|C_{\max}$ are strongly $\mathcal{NP}$-hard as shown by Röck (1984b) and Hall and Sriskandarajah (1996), respectively. However, as discussed in Chapter 2, the equivalence between the three concepts no-wait, blocking, and synchronous movement is no longer valid for more than two machines.

The following list provides a short overview of complexity results for synchronous flow shops that have already been presented by other authors. In all cases, the makespan was the considered objective function:

- Soylu et al. (2007) claim to prove $\mathcal{NP}$-hardness for $F3|\text{synmv}|C_{\max}$. There, it is shown that the synchronous flow shop problem is a special case of the problem with blocking and then deduced that the synchronous flow shop problem is $\mathcal{NP}$-hard as

well. However, this implication seems to be flawed since a special case of an $\mathcal{NP}$-hard problem does not need to be $\mathcal{NP}$-hard as well.

- Huang (2008) proved $\mathcal{NP}$-hardness for his synchronous flow shop model with a L/U unit for a configuration with one more machine besides the L/U unit. His proof of $\mathcal{NP}$-hardness relies on the characteristic of his model that within one cycle there can be two operations of two distinct jobs on the L/U unit. Therefore, his proof can not be used for our model. If unloading times for all jobs are constant, the problem can be solved optimally in polynomial time with the algorithm of Gilmore and Gomory as this problem relates to the two-machine synchronous flow shop within our setting. Huang further presented dynamic programming approaches for the cases with non-constant unloading times and one as well as two additional machines besides the L/U unit. He did not consider the special case of constant unloading times in this setting.

- Karabati and Sayin (2003) considered cyclic production minimizing the completion time of one production cycle. They provided a proof which can be slightly altered to show $\mathcal{NP}$-hardness for the problem $F|\text{synmv}|C_{\max}$ in which the number of machines is part of the input.

In the following theorem we prove $\mathcal{NP}$-hardness for the synchronous flow shop problem with three machines. We show that the strongly $\mathcal{NP}$-hard problem 3-PARTITION (cf. Garey and Johnson (1979)) can be reduced pseudo-polynomially to $F3|\text{synmv}|C_{\max}$. The basic idea will be the core for many proofs within this chapter: For an instance of an $\mathcal{NP}$-hard problem, an instance of the synchronous flow shop is created, for which a schedule without idle times on some or all of the machines exists if and only if the instance of the $\mathcal{NP}$-hard problem is solvable.

Theorem 3.1. *Problem $F3|synmv|C_{\max}$ is strongly $\mathcal{NP}$-hard.*

Proof. We prove this via a pseudo-polynomial reduction from the strongly $\mathcal{NP}$-hard problem 3-PARTITION (cf. Garey and Johnson (1979)). Let $(3P)$ be an instance of 3-PARTITION with $3m$ integers $a_1, \ldots, a_{3m}$ satisfying $mB = \sum_{i=1}^{3m} a_i$ and $B/4 < a_i < B/2$ for $i = 1, \ldots, 3m$. W.l.o.g. we assume $a_i \geq 2$ for all i which does not change the hardness of 3-PARTITION. The goal is to find a partition into disjoint subsets $A_0, \ldots, A_{m-1}$ (each containing three elements from $\{1, \ldots, 3m\}$) with $\sum_{i \in A_\lambda} a_i = B$ for $\lambda = 0, \ldots, m-1$.

We construct the following instance (SF) of the synchronous flow shop problem with a set $\mathcal{J}$ consisting of $2mB+1$ jobs. Let $S > \sum_{i=1}^{3m} i(a_i - 1)$ and $\omega > S$ be two large constants. We create the following jobs:

- One dummy job D_0 with processing times $(0, S, \omega + 1)$.
- One dummy job D_{mB} with processing times $(\omega + mB, S, 0)$.
- For $i = 1, \ldots, mB - 1$ dummy jobs D_i with processing times $(\omega + i, S_i, \omega + i + 1)$ where

$$S_i = \begin{cases} S, & \text{if } i \bmod B = 0 \\ 0, & \text{otherwise.} \end{cases}$$

- For $i = 1, \ldots, 3m$
  - one job j_1^i with processing times (S, ω, i)
  - for $l = 2, \ldots, a_i - 1$ jobs j_l^i with processing times (i, ω, i)
  - one job $j_{a_i}^i$ with processing times (i, ω, S).

  This results in a_i jobs for each element i. As an abbreviation we will call the set $J^i = \left\{ j_l^i \mid l = 1, \ldots, a_i \right\}$ the "job family" of element i. Because we assume $a_i \geq 2$ for all i, at least two jobs j_1^i and $j_{a_i}^i$ are created for each element i.

We show that there exists a partition of $(3P)$ into m sets with sum value B each iff the synchronous flow shop (SF) has a schedule with makespan

$$C_{\max} \leq mB\omega + \frac{mB(mB+1)}{2} + (3m+1)S + \sum_{i=1}^{3m} i(a_i - 1).$$

"$\Rightarrow$:" Let $A_0, \ldots, A_{m-1}$ be a partition into m sets of sum value B each and assume that $A_\lambda = \{\lambda_1, \lambda_2, \lambda_3\}$ for $\lambda = 0, \ldots, m-1$. Then we construct a sequence $\sigma = (\sigma_0, \ldots, \sigma_{2mB})$ in the following way: For positions $i \in \{0, \ldots, 2mB\}$ with $i \bmod 2 = 0$ set $\sigma_i = D_{i/2}$ (cf. Figure 3.2).

Afterwards, for $\lambda = 0, \ldots, m-1$ insert the jobs corresponding to the set $A_\lambda = \{\lambda_1, \lambda_2, \lambda_3\}$ with $a_{\lambda_1} + a_{\lambda_2} + a_{\lambda_3} = B$ at the positions $2\lambda B + 2\mu + 1$ for $\mu = 0, \ldots, B-1$ such that job family J^{λ_1} is processed in the first a_{λ_1}, job family J^{λ_2} in the next a_{λ_2} and job family J^{λ_3} in the last a_{λ_3} of these slots (cf. Figure 3.3).

In the schedule corresponding to sequence σ there are no idle times on machine M_1, and the cycle time c_t of cycle $t = 0, \ldots, 2mB + 2$ is

$$c_t = \begin{cases} 0, & t = 0 \\ \omega + \frac{t}{2}, & t = 2i, i = 1, \ldots, mB \\ S, & t = 2\lambda B + 1, \lambda = 0, \ldots, m-1 \\ \lambda_1, & t = 2(\lambda B + l) + 1; \lambda = 0, \ldots, m-1; l = 1, \ldots, a_{\lambda_1} - 1 \\ S, & t = 2(\lambda B + a_{\lambda_1}) + 1, \lambda = 0, \ldots, m-1 \\ \lambda_2, & t = 2(\lambda B + a_{\lambda_1} + l) + 1; \lambda = 0, \ldots, m-1; l = 1, \ldots, a_{\lambda_2} - 1 \\ S, & t = 2(\lambda B + a_{\lambda_1} + a_{\lambda_2}) + 1, \lambda = 0, \ldots, m-1 \\ \lambda_3, & t = 2(\lambda B + a_{\lambda_1} + a_{\lambda_2} + l) + 1; \lambda = 0, \ldots, m-1; l = 1, \ldots, a_{\lambda_3} - 1 \\ S, & t = 2mB + 1 \\ 0, & t = 2mB + 2. \end{cases}$$

The cycle times of the cycles depicted in Figure 3.3 can be obtained from this formula by setting $\lambda = 0$ for cycles $0, \ldots, 2B$ and $\lambda = 1$ for cycle $2B + 1$, respectively.

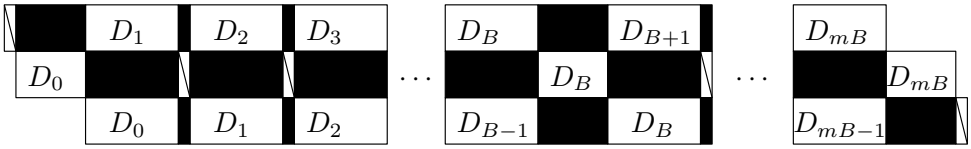


Figure 3.2: Distribution of the dummy jobs D_i. The black areas indicate idle slots that are filled with job families later on.

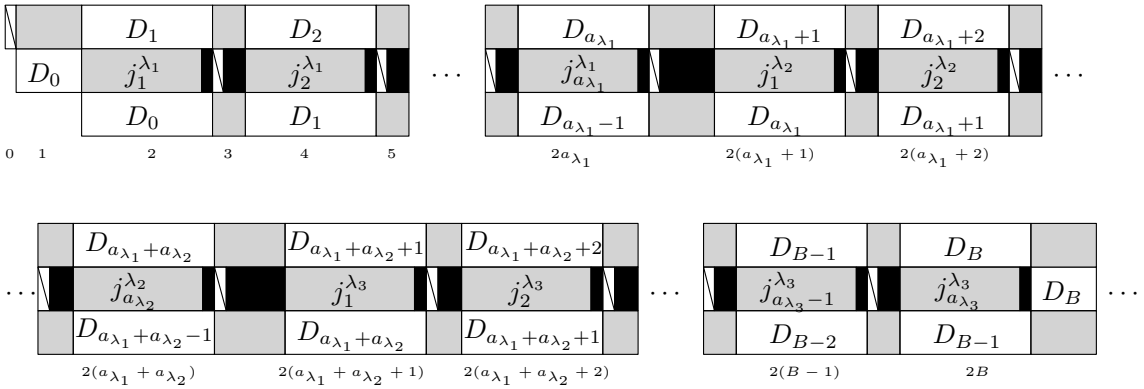


Figure 3.3: Insertion of job families $J^{\lambda_1}, J^{\lambda_2}, J^{\lambda_3}$ with $a_{\lambda_1}+a_{\lambda_2}+a_{\lambda_3} = B$ into the first B free slots. Job families are indicated in gray, idle times on machine M_2 are indicated in black. The numbering of the cycles is indicated below the respective cycles.

Since the cycle times sum up to

$$\sum_{t=0}^{2mB+2} c_t = \sum_{j\in\mathcal{J}} p_{1j} + S = mB\omega + \frac{mB(mB+1)}{2} + (3m+1)S + \sum_{i=1}^{3m} i(a_i - 1),$$

a schedule with the required makespan is found.

"$\Leftarrow$:" Let conversely $\sigma = (\sigma_0, \ldots, \sigma_{2mB})$ be a sequence with makespan

$$C^* \leq mB\omega + \frac{mB(mB+1)}{2} + (3m+1)S + \sum_{i=1}^{3m} i(a_i - 1).$$

We will show that the corresponding schedule is of the same structure as the one in the first part of the proof. The processing times of jobs D_i on machine M_1 sum up to $mB\omega + \frac{mB(mB+1)}{2}$, the processing times of all other jobs on machine M_1 sum up to $3mS + \sum_{i=1}^{3m} i(a_i - 1)$. Thus, there can be no idle time on machine M_1 as any idle time would result in a schedule with makespan larger than C^*.

At first we show that the dummy jobs D_i for $i = 0, \ldots, mB$ have to be scheduled in positions $2i$ in σ. If the jobs are not scheduled in these positions, there is at least one slot

in the schedule where a job of type D_i with $i < mB$ is processed on machine M_3 and no other job D_h with $h \neq i$ is processed on machine M_1 at the same time. As the processing time of any other job is at most S on machine M_1 in this cycle, this leads to an idle time of at least $\omega - S > 0$ time units on M_1 and hence to a schedule with makespan $C_{\max} > C^*$. Thus, these jobs have to be scheduled in positions $2i$. Because job D_0 is the only one with zero processing time on machine M_1, it has to be scheduled in the first position of σ as otherwise an idle time of at least ω would occur on machines M_2 and M_3 in the first cycle. Likewise, job D_{mB} has to be scheduled in the last position of σ, since otherwise this would lead to an idle time of at least ω on machines M_1 and M_2 in the last cycle. Thus, all jobs D_i have to be scheduled in the way described in the first part of the proof in ascending order D_i for $i = 0, \ldots, mB$ in positions $2i$.

Further, as this leads to processing times of length S on machine M_2 in positions $2iB+1$ for $i = 0, \ldots, m$, to avoid idle times on machine M_1 there has to be a job of type j_1^k for some k scheduled in positions i for $(i-1) \bmod 2B = 0$. This can be seen in Figure 3.3: Job D_0 is scheduled in cycle 0, leading to a processing time of S on machine M_2 in cycle 1. Therefore, to avoid an idle time on machine M_1, job $j_1^{\lambda_1}$ is scheduled in position 1, leading to a processing time of S on machine M_1 in cycle 1. Likewise, a job of type $j_{a_{\lambda_3}}^{\lambda_3}$ is scheduled in position $2B-1$, so the processing times on machines M_1 and M_2 are the same in cycle $2B+1$. Then, to avoid idle times on machines M_1 and M_3, starting with a job of type j_1^k in position i the whole job family J^k has to be scheduled in positions $i+2l$ for $l = 0, \ldots, a_i - 1$, as only in this case the processing time on machine M_1 is the same as the processing time on M_3 in the following cycles. Since this ends with a processing time of S on machine M_3 (and processing times on machine M_2 are at most S), another job family has to follow, as only the first job of a job family has a processing time of S on machine M_1. In Figure 3.3 this is depicted in cycles $2a_{\lambda_1} + 1$, $2(a_{\lambda_1} + a_{\lambda_2}) + 1$ and $2B+1$, respectively. Moreover, because of a processing time of length S on machine M_2 in positions $2iB + 1$ for $i = 0, \ldots, m$, the last job of a job family (i.e. a job of type $j_{a_k}^k$), has to be scheduled in position i for $(i+1) \bmod 2B = 0$. Only then, the processing times on machines M_2 and M_3 are the same and there are no idle times. In Figure 3.3 this occurs in cycle $2B-1$.

Due to the placement of all other jobs in the sequence and because the cardinality of each job family satisfies $B/4 < |J^i| < B/2$, for each $\lambda = 0, \ldots, m-1$ the subsequence $\sigma_{2\lambda B+2\mu+1}$ for $\mu = 0, \ldots, B-1$ has to consist of exactly three job families whose cardinalities sum up to B. This partition of job families leads to a solution of $(3P)$. In Figure 3.3, we show the schedule for one set $A_\lambda = \{\lambda_1, \lambda_2, \lambda_3\}$ with $a_{\lambda_1} + a_{\lambda_2} + a_{\lambda_3} = B$. Here, the jobs of the three job families $J^{\lambda_1}, J^{\lambda_2}$ and J^{λ_3} are scheduled in positions $\sigma_{2\mu+1}$ for $\mu = 0, \ldots, B-1$.

The above reduction is a pseudo-polynomial one since the number of jobs depends on the values a_i of the integers used in the 3-PARTITION problem. However, since 3-PARTITION is strongly $\mathcal{NP}$-hard, according to Garey and Johnson (1979) (page 101, Lemma 4.1) this is sufficient to show that the synchronous flow shop problem is also strongly $\mathcal{NP}$-hard. □

As in the classical flow shop, adding a further machine in a synchronous flow shop does not decrease the difficulty of the problem:

Theorem 3.2. *For each $m \in \mathbb{N}$ and any objective function f problem $Fm|synmv|f$ reduces polynomially to $F(m+1)|synmv|f$.*

Proof. Given an instance (I) of $Fm|\text{synmv}|f$ with jobs $j = 1, \ldots, n$ construct an instance (I') of $F(m+1)|\text{synmv}|f$: For each job j create a job j' with processing times $p'_{1j'} = 0$ and $p'_{ij'} = p_{i-1,j}$ for $i = 2, \ldots, m+1$. Obviously, this is a polynomial-time reduction and both instances have the same objective value. □

This implies that problem $Fm|\text{synmv}|C_{\max}$ is strongly $\mathcal{NP}$-hard for each fixed $m \geq 3$. Furthermore, $Fm|\text{synmv}|\sum C_j$ and $Fm|\text{synmv}|L_{\max}$ are strongly $\mathcal{NP}$-hard for each fixed $m \geq 2$.

3.3 Synchronous flow shop with dominating machines

As we showed in the preceding section, minimizing the makespan of a synchronous flow shop is $\mathcal{NP}$-hard for three or more machines. Therefore, unless $\mathcal{P} = \mathcal{NP}$, there exists no polynomial time algorithm and the problem becomes computationally intractable for larger instances. In this section, we will discuss special cases of the synchronous flow shop based on machine dominance and will show that some cases are polynomially solvable. We restrict ourselves to cases with one or two dominating machines as synchronous flow shop problems with three dominating machines are generalizations of three-machine synchronous flow shops and thus strongly $\mathcal{NP}$-hard as well for all considered objective functions.

3.3.1 One dominating machine

In this section we consider synchronous flow shop problems with a single dominating machine M_k. At first we show that if the number of machines is not fixed and the processing times on non-dominating machines are arbitrary, the synchronous flow shop problem with one dominating machine is $\mathcal{NP}$-hard for the objective functions $C_{\max}$ and $\sum C_j$.

Theorem 3.3. *Problem $F|synmv, dom(\mathcal{I})|C_{\max}$ is strongly $\mathcal{NP}$-hard for $|\mathcal{I}| = 1$ if the processing times on non-dominating machines are arbitrary.*

Proof. We prove this via a reduction from $F3|\text{synmv}|C_{\max}$ which we showed to be strongly $\mathcal{NP}$-hard in Theorem 3.1. Let $(SF3)$ be an instance of $F3|\text{synmv}|C_{\max}$ with n jobs and processing times p_{ij}. We construct the following instance $(SF1)$ of a synchronous flow shop problem with a single dominating machine. Let $\omega > \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} + 1$ be a large constant and set the number of machines to $n+3$ in (SF1). For each job $j = 1, \ldots, n$ of $(SF3)$ construct a job j' for $(SF1)$ with processing times

- $p'_{1j'} = \omega$,
- $p'_{ij'} = 0$ for $i = 2, \ldots, n$,

- $p'_{ij'} = p_{i-n,j}$ for $i = n+1, n+2, n+3$.

As can be easily seen, the first machine is dominating since the processing times for any job on machines $M_2, \ldots, M_{n+3}$ are smaller than the processing times $p'_{1j'}$ for any j'. We show that for each threshold value C^* the synchronous flow shop $(SF3)$ has a schedule with makespan $C_{\max} \leq C^*$ iff the synchronous flow shop $(SF1)$ has a schedule with makespan $C'_{\max} \leq n\omega + C^*$.

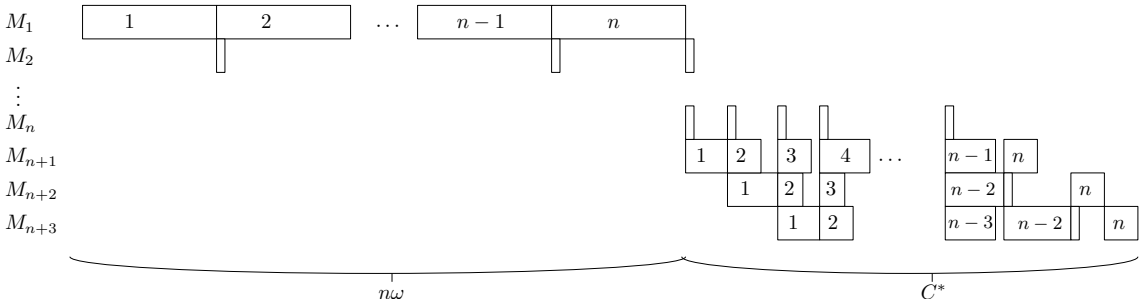


Figure 3.4: Schedule for $(SF1)$ with makespan $n\omega + C^*$ based on the sequence $\sigma = (1, \ldots, n)$ for $(SF3)$ with makespan C^*

"$\Rightarrow$": Let σ be a job sequence for $(SF3)$ with makespan $C_{\max} \leq C^*$. Define a sequence σ' for $(SF1)$ by setting $\sigma'_\lambda = j'$ if $\sigma_\lambda = j$. Since the first machine is dominating, the cycle times of the first n cycles sum up to $n\omega$. After cycle n there is no further job being processed on the first machine. Thus, the remaining cycle times are determined by the (arbitrary) processing times on the non-dominating machines. Due to the construction of the jobs, the cycle time of cycle $n+i$ in $(SF1)$ is the same as the cycle time of cycle i in $(SF3)$. Thus, the remaining cycle times add up to at most C^* and the schedule has the desired makespan. For illustration, Figure 3.4 shows a schedule for $(SF1)$ with makespan $n\omega + C^*$ based on the sequence $\sigma = (1, \ldots, n)$ for $(SF3)$ with makespan C^*.

"$\Leftarrow$": For a sequence σ' of $(SF1)$ a corresponding schedule for $(SF3)$ can be constructed in the same way as above. □

Theorem 3.4. *Problem $F|synmv, dom(\mathcal{I})|\sum C_j$ is strongly $\mathcal{NP}$-hard for $|\mathcal{I}| = 1$ if the processing times on the non-dominating machines are arbitrary.*

Proof. The proof is similar to the proof of Theorem 3.3 and we just sketch it here. Let $(SF2)$ be an instance of the strongly $\mathcal{NP}$-hard problem $F2|\text{synmv}|\sum C_j$ with n jobs. Construct an instance $(SF1)$ of a synchronous flow shop problem with a single dominating machine and objective function $\sum C_j$ in a similar way as in Theorem 3.3, using $n+2$ machines. Then, the synchronous flow shop $(SF2)$ has a schedule with $\sum C_j \leq C^*$ iff the synchronous flow shop $(SF1)$ has a schedule with $\sum C'_{j'} \leq n^2\omega + C^*$. □

In the following, we study the situation of job-independent processing times on the non-dominating machines, i.e. we assume $p_{ij} = p_i$ for all $i \notin \mathcal{I}$. As described in Section 2.3, in this situation w.l.o.g. we may assume $p_{ij} = 0$ for all $i \notin \mathcal{I}$. If a problem is already $\mathcal{NP}$-hard in this situation, the problem is also $\mathcal{NP}$-hard in the general situation. On the

other hand, if a problem is polynomially solvable for job-independent processing times on the non-dominating machines, it is also polynomially solvable for arbitrary processing times on the non-dominating machines and a fixed number m (the running time increases by the factor $\mathcal{O}(n^{m-1})$).

For any objective function f the problem $F|\text{synmv}, \text{dom}\,(k)\,, p_{ij}^{ndom} = 0|f$ with a single dominating machine M_k is closely related to the single-machine problem $1||f$. Any instance of the single-machine problem with jobs $j = 1, \ldots, n$ and processing times p_j can be transformed into a synchronous flow shop problem instance with one dominating machine and the same jobs by setting their processing times on M_k to p_j and on all other machines to zero. Obviously, the makespan of a single-machine sequence is equal to the makespan of the same sequence within the synchronous flow shop and does not depend on the sequence itself (it is equal to the sum of all processing times). Therefore, problem $F|\text{synmv}, \text{dom}\,(k)\,, p_{ij}^{ndom} = 0|C_{\max}$ is trivial for each index k and its optimal objective value can be calculated in $\mathcal{O}(n)$. Furthermore, as explained above, this implies that problem $Fm|\text{synmv}, \text{dom}\,(k)\,|C_{\max}$ can be solved in $\mathcal{O}(n^m)$ for each k.

Moreover, if we transform the single-machine problem into a synchronous flow shop where the last machine M_m is dominating and $p_{ij}^{ndom} = 0$, all completion times of the jobs in a synchronous flow shop schedule are the same as in a single-machine schedule. Therefore, problems with dominating machine M_m are equivalent to the corresponding single-machine problems. Since the problems $1||\sum T_j$ and $1||\sum w_j U_j$ are $\mathcal{NP}$-hard (cf. Du and Leung (1990), Lawler and Moore (1969)), the synchronous flow shop problems $F2|\text{synmv}, \text{dom}\,(2)\,, p_{ij}^{ndom} = 0|\sum T_j$ and $F2|\text{synmv}, \text{dom}\,(2)\,, p_{ij}^{ndom} = 0|\sum w_j U_j$ are $\mathcal{NP}$-hard as well.

However, if the dominating machine is not the last machine, determining the completion times of individual jobs becomes more difficult. If the dominating machine has index $k < m$, the completion time of each job depends on the sequence of the next $m-k$ jobs. In the following, we will discuss synchronous flow shop problems with a single dominating machine with zero processing time on the dominated machines where the objective functions depend on the individual completion times.

Minimizing total completion time

First, we study the problem of minimizing the total completion time $\sum C_j$ for one dominating machine. Let k be the index of the dominating machine and let $p_j := p_{kj}$ for simplicity. Problem $F|\text{synmv}, \text{dom}\,(k)\,, p_{ij}^{ndom} = 0|\sum C_j$ can be solved by scheduling the jobs according to non-increasing processing times on the dominating machine. This rule is known as the shortest processing time (SPT) rule and solves the single-machine problem $1||\sum C_j$ to optimality. As in the single-machine case, optimality can be proved by exchange arguments.

Theorem 3.5. *Problem $F|synmv, dom\,(k)\,, p_{ij}^{ndom} = 0|\sum C_j$ can be solved in $\mathcal{O}(n \log n)$ by scheduling the jobs according to the SPT rule for the dominating machine M_k.*

Proof. Let σ be an optimal sequence with completion times C_j and assume that σ does not satisfy the SPT rule for M_k. Then there are two jobs $\sigma_\lambda, \sigma_{\lambda+1}$ scheduled at consecutive

positions $\lambda, \lambda+1$ with $p_{\sigma_\lambda} > p_{\sigma_{\lambda+1}}$. Consider the sequence σ' where the jobs $\sigma_\lambda, \sigma_{\lambda+1}$ are interchanged. For σ' we have $C'_{\sigma_\lambda} = C_{\sigma_{\lambda+1}}$ and $C'_{\sigma_{\lambda+1}} = C_{\sigma_\lambda}$. Additionally, if $\lambda > m-k$ holds, the completion time of job $\sigma_{\lambda-(m-k)}$ is changed to

$$C'_{\sigma_{\lambda-(m-k)}} = C_{\sigma_{\lambda-(m-k)}} + p_{\sigma_{\lambda+1}} - p_{\sigma_\lambda} < C_{\sigma_{\lambda-(m-k)}}.$$

Since all other completion times remain the same, we have $\sum C'_j \leq \sum C_j$. Thus, by iteratively exchanging such jobs we obtain an optimal sequence satisfying the SPT rule. □

As discussed above, for the more general situation with arbitrary processing times on the non-dominating machines we get

Corollary 3.6. *Problem $Fm|synmv, dom(k)|\sum C_j$ can be solved in $\mathcal{O}(n^m \log n)$ time.*

When considering the total weighted completion time, the single-machine problem $1||\sum w_j C_j$ can also be solved in polynomial time, using Smith's rule (cf. Smith (1956)) that schedules the jobs in order of non-increasing ratios w_j/p_j. Unfortunately, this rule is not optimal for synchronous flow shops and the resulting schedules can even be arbitrarily bad. This can be seen in the following example for problem $F2|\text{synmv}, \text{dom}(1), p_{ij}^{ndom} = 0|\sum w_j C_j$ where ω is a large constant:

j	1	2	3
p_{1j}	1	ω	1
p_{2j}	0	0	0
w_j	ω	1	0

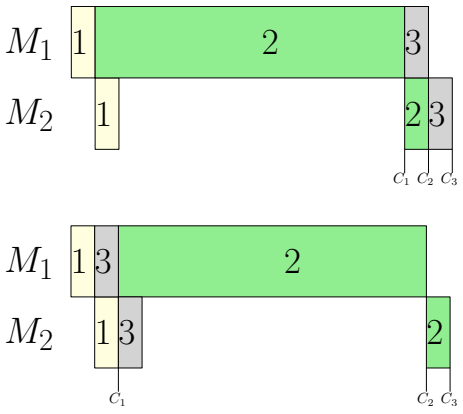


Figure 3.5: Schedule according to Smith's WSPT rule compared to an optimal schedule

Figure 3.5 shows the schedule according to Smith's Rule and the optimal solution, respectively. As can be seen, Smith's rule (scheduling the jobs according to their ratios w_j/p_j) sequences job 2 directly after job 1 which leads to a completion time of $\omega+1$ for the first job and thus a weighted completion time of $\omega(\omega+1)$. If job 3 is sequenced after

job 1 instead, this results in a weighted completion time of 2ω. Therefore, the relative error of Smith's rule for this instance is

$$\frac{\omega(\omega+1)+(\omega+2)-(2\omega+\omega+2)}{(2\omega+\omega+2)} = \frac{\omega^2-\omega}{3\omega+2}$$

which can be arbitrarily large.

Although we cannot use a similar rule to find an optimal schedule, we can show the following dominance rule: If for two jobs i, j the inequalities $p_i \leq p_j$ and $w_i \geq w_j$ hold, then there exists an optimal schedule in which job i is scheduled prior to job j. This result may reduce the solution space of permutations. The dominance rule can once again be proven by an exchange argument similar to the unweighted case. However, the complexity status of problem $F2|\text{synmv}, \text{dom}\,(1)\,, p_{ij}^{ndom} = 0|\sum w_j C_j$ remains open.

Problems involving job lateness

In the following, we study the problem of minimizing the maximum lateness $L_{\max}$ for a single dominating machine. Let again k be the index of the dominating machine and let $p_j := p_{kj}$ for simplicity. At first we consider problem $F|\text{synmv}, \text{dom}\,(k)\,, p_{ij}^{ndom} = 0|L_{\max}$, i.e. the situation that the processing times on the non-dominating machines are zero.

For the single-machine problem $1||L_{\max}$ Jackson's earliest due date (EDD) rule, i.e. sorting the jobs in non-decreasing order of their due dates, is a polynomial algorithm to minimize the maximum lateness. As with the WSPT rule for minimizing the total weighted completion time, the EDD rule is not optimal and can lead to arbitrarily bad schedules for problem $F2|\text{synmv}, \text{dom}\,(1)\,, p_{ij}^{ndom} = 0|L_{\max}$.

In the following, at first we consider the decision variant of $F|\text{synmv}, \text{dom}\,(k)\,, p_{ij}^{ndom} = 0|L_{\max}$ asking whether a feasible schedule with $L_{\max} \leq L$ for a given threshold value L exists. For this problem, Algorithm 3.1 determines a job sequence from the last to the first position. If there exists a feasible schedule, then also a feasible schedule without any idle times on M_k exists. Hence, from the sum of all processing times p_j we know the length of such a schedule, as it is sequence independent. Thus, we can calculate the completion time of the last job in the sequence and can determine which jobs can be scheduled last in the sequence such that their lateness is not larger than the specified maximum lateness L. If more than one job is feasible to be scheduled in the current position of the sequence in iteration λ, we choose a feasible job with largest processing time. The completion time of the job added in the next iteration $\lambda+1$ is calculated in line 11. Because we assume zero processing times on the non-dominating machines, in any schedule the last $m-k+1$ jobs all have completion time T. Therefore, nothing is subtracted from T in line 11 in the first $m-k$ iterations of the algorithm. For $\lambda > m-k$, the processing time of the job scheduled in position $n-\lambda+1+m-k$, which is the job scheduled $m-k+1$ positions after the job to be scheduled in iteration $\lambda+1$, is subtracted. As the job scheduled in position $n-\lambda+1+m-k$ was one with largest processing time of all feasible jobs, the completion time of the job to be scheduled in iteration $\lambda+1$ is as small as possible.

Input: number of machines m, index of dominating machine $k \leq m$
Input: jobs $j = 1, \ldots, n$ with processing times p_j on machine M_k, due dates d_j, threshold $L \in \mathbb{Z}$
Output: a job sequence with $L_{\max} \leq L$ if it exists, an empty sequence otherwise

```
1  T ← Σ_{j=1}^{n} p_j
2  σ ← sequence of jobs, initially empty
3  for λ = 1, ..., n do
4  |  feasibleJobs ← unscheduled jobs j with T − d_j ≤ L
5  |  if feasibleJobs = ∅ then
6  |  |  return ∅
7  |  end
8  |  else
9  |  |  Assign to σ_{n−λ+1} a feasible job with largest processing time
10 |  |  if λ > m − k then
11 |  |  |  T ← T − p_{σ_{n−λ+1+m−k}}
12 |  |  end
13 |  end
14 end
15 return σ
```

Algorithm 3.1: Checking whether a schedule with maximum lateness $\leq L$ exists

Theorem 3.7. *For problem $F|synmv, dom(k), p_{ij}^{ndom} = 0|L_{\max}$ and a threshold value $L \in \mathbb{Z}$, Algorithm 3.1 returns a feasible job sequence with $L_{\max} \leq L$ in $\mathcal{O}(n \log n)$ if such a sequence exists.*

Proof. If $L \neq 0$, we can transform each instance into an equivalent instance by setting all due dates to $d'_j = d_j - L$ and considering the threshold 0. Thus, w.l.o.g. we assume $L = 0$. Assume there exists a sequence of the jobs $\tilde{\sigma} = (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n)$ such that no job is completed after its due date. We show that in each iteration $\lambda = 1, \ldots, n$ of the algorithm, there is at least one feasible job to be scheduled, i.e. line 6 is never reached.

Let $l := m - k + 1$ and $T := \sum_{j=1}^{n} p_j$ be the sum of processing times on the dominating machine. Because we assume zero processing times on the non-dominating machines, in any schedule the last l jobs all have completion time T. If during the first l steps of the algorithm there is no more feasible job, less than l jobs have a due date of at least T, contradicting that there exists a feasible schedule of the jobs such that no job is completed after its due date.

Consider iteration $\lambda > l$ of Algorithm 3.1 and assume that there is no feasible job. The completion time of the job to be scheduled in iteration λ is given by

$$t_\lambda = T - \sum_{\mu=1}^{\lambda-l} p_{\sigma_{n-\mu+1}}$$

where $p_{\sigma_{n-\mu+1}}$ is the processing time of the job scheduled in iteration μ (i.e. the job scheduled in position $n-\mu+1$). Consider the job $\tilde{\sigma}_{n-\lambda+1}$ which is scheduled in this position in the feasible sequence $\tilde{\sigma}$ with completion time $\tilde{t}_\lambda$. Since this job is not feasible in iteration λ of the algorithm, this is either because the job has already been scheduled in a prior iteration of the algorithm or because $d_{\tilde{\sigma}_{n-\lambda+1}} < t_\lambda$ holds. Since Algorithm 3.1 always inserts a feasible job with largest processing time, t_λ is as small as possible and thus $t_\lambda \leq \tilde{t}_\lambda$, which contradicts $\tilde{t}_\lambda \leq d_{\tilde{\sigma}_{n-\lambda+1}} < t_\lambda$. If job $\tilde{\sigma}_{n-\lambda+1}$ has already been scheduled in Algorithm 3.1, then one of the jobs succeeding this one in $\tilde{\sigma}$ has not been scheduled by the algorithm. Because this job is infeasible in iteration λ, its due date must be greater than t_λ. However, this contradicts the assumption that the maximum lateness of $\tilde{\sigma}$ is at most zero.

Algorithm 3.1 can be implemented such that it runs in $\mathcal{O}(n \log n)$. Initially, we sort the jobs according to decreasing due dates. Furthermore, we maintain a heap containing all feasible unscheduled jobs j with $T - d_j \leq L$ of the current iteration. As key values in the heap we use the processing times p_j of the jobs. Then, in each iteration a feasible job with largest processing time can be determined in $\mathcal{O}(1)$. Afterwards, the heap has to be updated which can be done in $\mathcal{O}(\log n)$. Additionally, new jobs may have to be inserted into the heap for the new value T. This can efficiently be done by going through the initial list where the jobs are sorted according to decreasing due dates. Since in total we have n iterations, each job is inserted into the heap exactly once, and in each iteration at most n jobs are in the heap, all heap updates can be performed in $\mathcal{O}(n \log n)$. □

Using Algorithm 3.1 we can also solve the optimization problem $F|\text{synmv}, \text{dom}(k), p_{ij}^{ndom} = 0|L_{\max}$ and find the minimum maximum lateness for any instance by applying the algorithm for different values of L. One possibility is to use a binary search approach, starting with a lower bound of $-\max_j d_j$ and an upper bound of $\sum_j p_j - \min_j d_j$. Then, the number of times Algorithm 3.1 has to be executed depends on the numerical values of p_j and d_j and hence this approach does not yield a strongly polynomial algorithm. However, we can use Algorithm 3.1 in the strongly polynomial-time Algorithm 3.2 to find the minimum maximum lateness of any instance. Algorithm 3.2 starts with an upper bound on the maximum lateness and uses Algorithm 3.1 to construct a feasible schedule σ. In each iteration of the algorithm a new threshold value for the maximum lateness is calculated and tested by Algorithm 3.1 until Algorithm 3.1 no longer returns a feasible schedule.

Theorem 3.8. *For problem $F|synmv, dom(k), p_{ij}^{ndom} = 0|L_{\max}$ Algorithm 3.2 finds a sequence of jobs minimizing the maximum lateness $L_{\max}$ in $\mathcal{O}(n^3 \log n)$.*

Proof. First we show that the algorithm is correct. In the first iteration, Algorithm 3.1 is executed with a value of $L = \sum_j p_j - \min_j d_j$. As this is an upper bound for the maximum lateness of each instance and Algorithm 3.1 is correct (Theorem 3.7), we get a schedule with lateness $L_{\max} \leq L$ and reach line 11 in Algorithm 3.2. There the minimum deviation d of the lateness of the returned schedule from the current threshold L is calculated. The consequence is that this schedule yields a maximum lateness of $L_{\max} = L - d$ and therefore a schedule with maximum lateness $L_{\max} = L - d$ exists. In the next step of Algorithm 3.2

Input: number of machines m, index of dominating machine $k \leq m$
Input: jobs $j = 1, \ldots, n$ with processing times p_j on machine M_k, due dates d_j
Output: optimal $L_{\max}$-value
1 $L \leftarrow \sum_{j=1}^{n} p_j - \min_{j=1,\ldots,n} d_j$
2 $d \leftarrow 0$
3 $\mathcal{F} \leftarrow$ True
4 **while** $\mathcal{F}$ **do**
5 Execute algorithm 3.1 with threshold L to get a schedule σ with completion times C_j.
6 **if** *No schedule with* $L_{\max} \leq L$ *exists* **then**
7 $\mathcal{F} \leftarrow$ False
8 $L \leftarrow L + 1$
9 **end**
10 **else**
11 $d \leftarrow \min_j((d_j + L) - C_j)$
12 $L \leftarrow L - d - 1$
13 **end**
14 **end**
15 **return** L

Algorithm 3.2: Minimizing the maximum lateness

we thus try to find a schedule with maximum lateness $L_{\max} = L - d - 1$. We iteratively go on until no such schedule can be found for values L and d. Then, as a schedule with maximum lateness $L_{\max} = L - d$ exists, but a schedule with $L_{\max} = L - d - 1$ does not, we have found the minimum maximum lateness.

Next we show that the algorithm runs in $\mathcal{O}(n^3 \log n)$. Let σ be the feasible schedule for threshold value L returned by Algorithm 3.1 in line 5. Consider the job j with largest completion time C_j satisfying $(d_j + L) - C_j = d$, which we will call the "critical job". Let μ be the position of j in σ. Consider the execution of Algorithm 3.1 for a lateness value of $L - d - 1$ in the next iteration of Algorithm 3.2. Then, as job j has to be planned such that its completion time is $C_j \leq d_j + L - d - 1$, the schedule σ is no longer feasible and has to be changed. Algorithm 3.1 constructs the new schedule $\tilde{\sigma}$ in the following way: For all iterations $\lambda < n - \mu + 1$, the set of feasible jobs constructed in line 4 is a subset of the set of feasible jobs in the prior execution: Each job that was feasible to be scheduled in position $n - \lambda$ in σ is still feasible as the job is non-critical in σ. Also, in each iteration $\lambda < n - \mu + 1$ job $\sigma_{n-\lambda+1}$ is chosen in line 9, as there is no job with larger processing time in the set of feasible jobs. Thus, $\tilde{\sigma}_l = \sigma_l$ for $l > \mu$ holds. In iteration $n - \mu + 1$ the critical job j can not appear in the set of feasible jobs in line 4. Further, no job with processing time larger than p_j is in the set of feasible jobs, because otherwise it would have been scheduled in position μ in schedule σ already. Thus, either the set of feasible jobs is empty and no schedule with

lateness $L - d - 1$ can be found, or a feasible job with processing time smaller than p_j has to be scheduled in position μ. Therefore, for each job being critical, the schedule can be improved at most $\mathcal{O}(n)$ times resulting in at most $\mathcal{O}(n^2)$ iterations of Algorithm 3.2. Since Algorithm 3.1 runs in $\mathcal{O}(n \log n)$, the total running time is $\mathcal{O}(n^3 \log n)$. □

As discussed above, for the more general situation with arbitrary processing times on the non-dominating machines we get

Corollary 3.9. *Problem* $Fm|synmv, dom\,(k)\,|L_{\max}$ *can be solved in* $\mathcal{O}(n^{m+2} \log n)$ *time.*

To achieve this, we have to iterate over all $\mathcal{O}(n^{m-1})$ possibilities to fix the job sequences in the first $k-1$ and last $m-k$ cycles. Then, for all these possibilities, Algorithm 3.2 has to be employed, using slight modifications of Algorithm 3.1 in each case: The completion times of the last $m-k+1$ job have to be calculated individually based on the fixed jobs' processing times on the non-dominating machines. If the fixed jobs do not already lead to a lateness greater than the threshold, the non-fixed jobs once again can be scheduled iteratively, in each iteration choosing a feasible job with largest processing time. We only considered the case with zero processing times on non-dominating machines in the algorithms and proofs above to improve the readability. Nonetheless, the correctness of the algorithm and the time complexity for the case with arbitrary processing times on non-dominating machines can be proven in an analogous manner.

Another polynomially solvable single-machine problem involving due dates is the problem $1||\sum U_j$, which can be solved to optimality in polynomial time by Moore's Algorithm (cf. Moore (1968)). Unfortunately this algorithm does not find optimal solutions for synchronous flow shops with one dominating machine. Since we also did not find another polynomial-time algorithm, the complexity status of problem $F2|\text{synmv}, \text{dom}\,(1)\,, p_{ij}^{ndom} = 0|\sum U_j$ remains open.

3.3.2 Two dominating machines

In this section we consider synchronous flow shop problems with two dominating machines. In the case of an arbitrary number of machines where only two are dominating and the two dominating machines directly succeed each other, the makespan of the flow shop with synchronous movement is once more closely related to the makespan of the two-machine flow shop with either no-wait or blocking constraints. Ignoring sequence independent constants, job sequences for the two-machine no-wait flow shop and the synchronous flow shop have the same makespan. Hence, for each $1 \leq k < m$ problem $F|\text{synmv}, \text{dom}\,(k, k+1)\,, p_{ij}^{ndom} = 0|C_{\max}$ is equivalent to problem $F2|\text{no-wait}|C_{\max}$ and can be solved in $\mathcal{O}(n \log n)$. However, the individual completion times C_j for the jobs again depend on the succeeding jobs in the sequence and thus cannot be calculated easily, similar to the case with only one dominating machine.

An interesting question is whether it makes a difference if the two dominating machines directly succeed each other or there are non-dominating machines in between (e.g. in the case of $F3|\text{synmv}, \text{dom}\,(1,2)\,|f$ where M_1, M_2 are dominating and $F3|\text{synmv}, \text{dom}\,(1,3)\,|f$ where M_1, M_3 are dominating). While problem $F2|\text{synmv}, \text{dom}\,(1,2)\,|C_{\max}$ is equivalent

to $F2|\text{synmv}|C_{\max}$ and hence polynomially solvable as described in Section 3.2, it is not clear whether problem $F3|\text{synmv}, \text{dom}\,(1,3)\,|C_{\max}$ can also be solved in polynomial time. Intuition guides one into thinking that a problem where a non-dominating machine lies between two dominating machines is at least as difficult as a problem in which the two dominating machines are directly succeeding each other. More formally, one would assume that for $m \geq 3$ and two indices $k_1 < k_2 < m$ problem $Fm|\text{synmv}, \text{dom}\,(k_1, k_2+1)\,|f$ is at least as difficult as problem $Fm|\text{synmv}, \text{dom}\,(k_1, k_2)\,|f$ for any objective function f. Unfortunately we were not able to prove this intuitive result in the general case, but it can be proven for the objective function of minimizing the makespan. For the makespan objective function and job-independent processing times on non-dominating machines we can ignore all machines before the first dominating machine and after the second dominating machine. Further, for arbitrary processing times on the non-dominating machines we can once more fix the first and last jobs in the sequence. Thus, w.l.o.g. we can assume that the first and last machines are the dominating ones and it is sufficient to consider the relationship between problems $Fm|\text{synmv}, \text{dom}\,(1,m)\,|C_{\max}$ and $F(m+1)|\text{synmv}, \text{dom}\,(1,m+1)\,|C_{\max}$. For them we have

Theorem 3.10. *For each $m \in \mathbb{N}$ with $m \geq 2$ problem $Fm|\text{synmv}, \text{dom}\,(1,m)\,|C_{\max}$ reduces polynomially to $F(m+1)|\text{synmv}, \text{dom}\,(1,m+1)\,|C_{\max}$.*

Proof. Let (I) be an instance of $Fm|\text{synmv}, \text{dom}\,(1,m)\,|C_{\max}$ with jobs $j = 1, \ldots, n$, machines $M_1, \ldots, M_m$, $n \geq m+1$, and processing times p_{ij}. Construct an instance (I') of problem $F(m+1)|\text{synmv}, \text{dom}\,(1,m+1)\,|C_{\max}$ with machines $M'_1, \ldots, M'_{m+1}$ and processing times p'_{ij} in the following way:

- For each job j create a job j' with processing times $p'_{ij'} = p_{ij}$ for $i = 1, \ldots, m-1$, $p'_{mj'} = 0$, and $p'_{m+1,j'} = p_{mj}$.
- Let $P := \sum_{j=1}^{n} (p_{1j} + p_{2j}) + 1$. Create $k := \left\lceil \frac{n}{m-1} \right\rceil$ auxiliary jobs $a_1, \ldots, a_k$. For the first auxiliary job a_1 set $p'_{1,a_1} = 0$ and $p'_{m+1,a_1} = P$. For the last auxiliary job a_k set $p'_{1,a_k} = P$ and $p'_{m+1,a_k} = 0$, and for all auxiliary jobs a_l for $l = 2, \ldots, k-1$ set $p'_{1,a_l} = p'_{m+1,a_l} = P$. Additionally, set $p'_{i,a_l} = 0$ for $i = 2, \ldots, m$ and $l = 1, \ldots, k$.

Obviously, if machines M_1 and M_m are dominating for (I), then machines M'_1 and M'_{m+1} are dominating for (I'). Let $C \leq P - 1$. We will show that the instance (I) has a solution with makespan $C_{\max} \leq C$ iff the instance (I') has a solution with makespan $C'_{\max} \leq C + (k-1)P$.

"$\Rightarrow$": Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ be a sequence of jobs for (I) with makespan $C_{\sigma_n} \leq C$. Construct a sequence σ' for the instance (I') by

$$\sigma'_i = \begin{cases} a_{\lceil \frac{i}{m} \rceil} & \text{if } i \bmod m = 1 \\ \sigma_{i - \lceil \frac{i}{m} \rceil} & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, n+k$, i.e. inserting the dummy jobs as every m-th job (see Figure 3.6 for an example with $m = 2$ machines). The auxiliary jobs are scheduled in such a way that

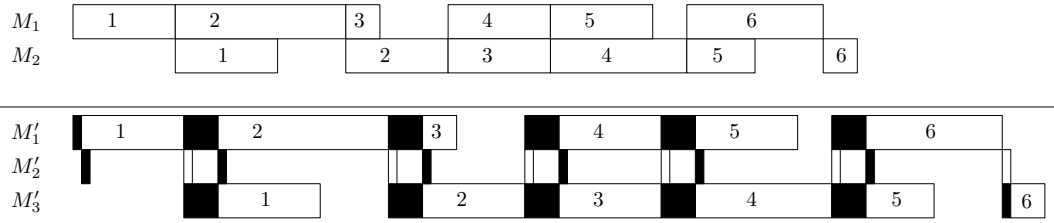


Figure 3.6: $F2|\text{synmv}, \text{dom}\,(1,2)\,|C_{\max}$ and $F3|\text{synmv}, \text{dom}\,(1,3)\,|C_{\max}$ with added auxiliary jobs. The auxiliary jobs are in black, their processing times are not according to scale to improve readability.

auxiliary job a_l is processed on machine M'_{m+1} when auxiliary job a_{l+1} is processed on machine M'_1. The auxiliary jobs contribute a total of $(k-1)P$ to the makespan $C'_{\max}$. The remaining jobs are scheduled in between the auxiliary jobs such that job σ_i is processed on machine M'_{m+1} when σ_{i+m} is processed on machine M'_1. Thus, the non-auxiliary jobs add C_{σ_n} to the makespan of (I'), which leads to the desired makespan of $C'_{\max} \leq C + (k-1)P$.

"$\Leftarrow$": Let $\sigma' = \left(\sigma'_1, \sigma'_2, \ldots, \sigma'_{n+k}\right)$ be a sequence of jobs for (I') with makespan $C'_{\sigma_{n+k}} \leq C + (k-1)P$. Note that each auxiliary job has processing time P on at least one machine. Due to $C \leq P$ this means that the auxiliary jobs have to be scheduled in such a way that for each $l_1 = 1, \ldots, k-1$, auxiliary job a_{l_1} is processed on machine M'_{m+1} when another auxiliary job a_{l_2} for some $l_2 \in \{2, \ldots, k\}$ is processed on machine M'_1. Further, auxiliary job a_1 has to be scheduled as the first of the auxiliary jobs in the first m positions of the schedule and job a_k has to be scheduled as the last auxiliary job to allow for a processing time of $(k-1)P$ for the auxiliary jobs. For all jobs σ'_i, the job is processed on the dominating machine M'_{m+1} when job σ'_{i+m} is processed on machine M'_1. Let λ be the position of the first auxiliary job a_1 and let w.l.o.g. a_l be scheduled in position $\lambda + m(l-1)$ for $l = 2, \ldots, k$. Construct a sequence σ by

$$\sigma_\mu = \begin{cases} \sigma'_{\mu + \left\lceil \frac{\mu - \lambda + 1}{m} \right\rceil} & \text{for } \mu \geq \lambda \\ \sigma'_\mu & \text{otherwise,} \end{cases}$$

for $\mu = 1, \ldots, n$, i.e. removing all auxiliary jobs. Then this sequence results in a makespan of $C_{\sigma_n} \leq C$ for instance (I) with the same arguments as above. □

Theorem 3.10 shows that adding a non-dominating machine between two dominating machines does not decrease the difficulty of the problem. Thus, if we could show $\mathcal{NP}$-hardness for $F3|\text{synmv}, \text{dom}\,(1,3)\,|C_{\max}$, this would imply $\mathcal{NP}$-hardness for $Fm|\text{synmv}, \text{dom}\,(k_1, k_2)\,, p_{ij}^{ndom} = 0|C_{\max}$ for all $1 \leq k_1 < k_2 - 1 < m$, i.e. in all situations where the two dominating machines are not directly succeeding. While the problem is open for a fixed number of machines, we can prove that problems with two dominating machines are $\mathcal{NP}$ hard if the number of machines and the positions of the dominating machines are part of the input:

Theorem 3.11. *Problem* $F|synmv, dom(\mathcal{I}), p_{ij}^{ndom} = 0|C_{\max}$ *is strongly* $\mathcal{NP}$*-hard for* $|\mathcal{I}| = 2$.

Proof. This can be proven via a pseudo-polynomial reduction from 3-PARTITION in a similar way as in Theorem 3.1. We construct the following instance (SF) of the synchronous flow shop problem with $m+1$ machines and mB jobs. For each $i = 1, \ldots, 3m$ introduce the following jobs:

- One job j_1^i with processing times $p_{1,j_1^i} = 0$ and $p_{m+1,j_1^i} = i$
- For $l = 2, \ldots, a_i - 1$ jobs j_l^i with processing times $p_{1,j_l^i} = p_{m+1,j_l^i} = i$
- One job $j_{a_i}^i$ with processing times $p_{1,j_{a_i}^i} = i$ and $p_{m+1,j_{a_i}^i} = 0$.

All other processing times are set to zero, i.e. the machines M_1 and M_{m+1} are dominating. This results in a_i jobs for each element i. As in the proof of Theorem 3.1 we will call the set $J^i = \left\{ j_l^i \mid l = 1, \ldots, a_i \right\}$ the "job family" of element i. Because we again assume $a_i \geq 2$ for all $i = 1, \ldots, 3m$, at least two jobs j_1^i and $j_{a_i}^i$ are created for each element i. We show that there exists a partition of $(3P)$ into m sets with sum value B each, iff the synchronous flow shop (SF) has a schedule with makespan $C_{\max} \leq \sum_{i=1}^{3m} i(a_i - 1)$.

"$\Rightarrow$:" Let $A_0, \ldots, A_{m-1}$ be a partition into m sets of sum value B each and assume that $A_\lambda = \{\lambda_1, \lambda_2, \lambda_3\}$. Then we construct a sequence $\sigma = (\sigma_0, \ldots, \sigma_{mB-1})$ in the following way: for $\lambda = 0, \ldots, m-1$ insert the jobs corresponding to the set $A_\lambda = \{\lambda_1, \lambda_2, \lambda_3\}$ with $a_{\lambda_1} + a_{\lambda_2} + a_{\lambda_3} = B$ at the positions $\lambda + \mu m$ for $\mu = 0, \ldots, B-1$ such that job family J^{λ_1} is processed in the first a_{λ_1}, job family J^{λ_2} in the next a_{λ_2} and job family J^{λ_3} in the last a_{λ_3} of these slots. In this sequence there are no idle times on machines M_1 and M_{m+1} and the cycle times sum up to $\sum_{i=1}^{3m} i(a_i - 1)$. Thus, a schedule with the required makespan is found.

"$\Leftarrow$:" Let conversely σ be a sequence with makespan $C^* \leq \sum_{i=1}^{3m} i(a_i - 1)$. We will show that the schedule is of the same structure as the one in the first part of the proof. Because the processing times of all jobs on machines M_1 and M_{m+1} sum up to $\sum_{i=1}^{3m} i(a_i - 1)$, there may be no idle time on those machines as any idle time would result in a schedule with makespan larger than C^*. Thus, the first m jobs have to be of type j_1^k for some k and the last m jobs have to be of type $j_{a_k}^k$. To avoid idle times on machine M_1 starting with a job of type j_1^k in position λ the whole job family J^k has to be scheduled in positions $\lambda + \mu m$ for $\mu = 0, \ldots, a_k - 1$. Additionally, if $\lambda + (a_k - 1)m < n - m$, another job family has to follow as the last job of the family has a processing time of 0 on machine M_{m+1}.

Because the cardinality of each job family satisfies $B/4 < |J^i| < B/2$, for each $\lambda = 0, \ldots, m-1$ the subsequence $\sigma_{\lambda+\mu m}$ for $\mu = 0, \ldots, B-1$ has to consist of exactly three job families whose cardinalities sum up to B. This partition of job families leads to a solution of $(3P)$. □

As discussed in Section 3.2, for the problem of minimizing the maximum lateness or the total completion time, the results of Röck (1984a) can be used to prove $\mathcal{NP}$-hardness for two-machine synchronous flow shops. The proof for minimizing the maximum lateness can be easily adapted to show $\mathcal{NP}$-hardness for any position of two dominating machines. In the following, we include a generalized version of the proof of Röck.

The proof uses a reduction from the $\mathcal{NP}$-complete problem Hamiltonian path (cf. Garey et al. (1976)). Given a graph $G = (V, E)$, is there a path through all vertices in V that visits every vertex exactly once? The idea of the reduction relies on the following observation: Let $G^- = (V^-, E^-)$ with $V^- = \{2, \ldots, n\}$ be a connected graph with degree $d(v) \geq 1$ for all $v \in V^-$. Define a graph $G = (V, E)$ with $V = V^- \cup \{1\}$ and $E = E^- \cup \{(1, v) | v \in V^-\}$. Then, G^- contains a Hamiltonian path if and only if G contains a Hamiltonian cycle, i.e. a cycle that visits each vertex in V exactly once. Transform G into a directed graph $\vec{G} = (V, A_E)$ where each edge $e = \{v, w\} \in E$ is represented by two arcs (v, w) and (w, v) in A_E. Because the in-degree of all vertices in $\vec{G}$ is equal to its out-degree, $\vec{G}$ always contains an Eulerian circuit, i.e. a circuit of length $|A_E|$ which includes every edge in A_E. Then, if G contains a Hamiltonian cycle, $\vec{G}$ also contains a directed Hamiltonian circuit and therefore it also contains a directed Hamiltonian path h which ends in 1. Further, there exists an Eulerian circuit in $\vec{G}$ that starts and ends in 1 and where the last n vertices constitute h: Let v_0 be the first vertex of h. Removing all arcs in h leads to a connected directed graph where for each vertex except for v_0 and 1 its in-degree is equal to its out-degree, vertex v_0 has (in-degree)$-$(out-degree) $= 1$ and vertex 1 has (out-degree)$-$(in-degree) $= 1$. Thus, the reduced graph contains an Eulerian path starting in 1 and ending in v_0. Adding h again leads to the desired circuit. Then, the first $n - 1$ vertices of h constitute a Hamiltonian path in G^-.

We will encode the arc and vertex set of the directed graph into a flow shop problem and show that a schedule without any late jobs exists if and only if there exists an Eulerian circuit in $\vec{G}$ where the last n arcs constitute a directed Hamiltonian path that ends in vertex 1.

Theorem 3.12. *The problem $Fm|synmv, dom(\mathcal{I}), p_{ij}^{ndom} = 0|L_{\max}$ is strongly $\mathcal{NP}$-hard for each fixed $m \geq 2$ and each set $\mathcal{I}$ with $|\mathcal{I}| = 2$.*

Proof. We prove the theorem by a reduction from Hamiltonian path. Let $G^- = (V^-, E^-)$ with $V = \{2, \ldots, n\}$ be a connected graph with degree $d(v) \geq 1$ for all vertices $v \in V$. Let $G = (V, E)$ be a graph with $V = V^- \cup \{1\}$ and $E = E^- \cup \{(1, v) | v \in V^-\}$. Further, let $1 \leq k_1 < k_2 \leq m$. We construct an instance (SF) of $F|\text{synmv}, \text{dom}(k_1, k_2)|L_{\max}$ with a set $\mathcal{J}$ consisting of $4(k_2 - k_1)|E|$ jobs. In the following we use the notation $(p_{k_1,j}, p_{k_2,j})$ to denote job j's processing times on machine M_{k_1} and machine M_{k_2}, respectively. Set processing time $p_{ij} = 0$ for all $j \in \mathcal{J}$ and $i \notin \{k_1, k_2\}$ and let $K := 2(n + 1)$. We create the following jobs:

- For each vertex $v \in V$ jobs VE_v^λ with processing times $(K - v, v)$ for $\lambda = 1, \ldots, d(v)$.
- For each edge $\{v, w\} \in E$ one job AR_{vw} with processing times $(v, K - w)$ and one job AR_{wv} with processing times $(w, K - v)$.

- For each $i = 1, \ldots, (k_2 - k_1 - 1)$ jobs D_0^i with processing times $(1, K)$ and jobs $D_{4|E|-1}^i$ with processing times $(K, 1)$.
- For $j = 1, \ldots, 4|E| - 2$ and $i = 1, \ldots, (k_2 - k_1 - 1)$ jobs D_j^i with processing times (K, K).

As due dates for jobs $\mathrm{VE}_v^{d(v)}$ and all jobs of type AR and D we define

$$\begin{aligned} d_2 &:= 2|E| \cdot K + (4|E| - 1) \cdot (k_2 - k_1 - 1) \cdot K + (k_2 - k_1) \\ &= \sum_{j \in \mathcal{J}} p_{k_1,j} + (k_2 - k_1) \end{aligned}$$

which is the sum of processing times of all jobs on machine M_{k_1} (and machine M_{k_2}, respectively) increased by the distance of the two dominating machines. For jobs VE_v^{λ} for $v \in V$ and $\lambda < d(v)$ we choose the due date d_1 as:

$$d_1 := d_2 - nK - 2n\,(k_2 - k_1 - 1)\,K + \left(m - k_2 - \left\lfloor \frac{m - k_2}{2(k_2 - k_1)} \right\rfloor\right) K$$

Then, there exists a schedule for (SF) with $L_{\max} \leq 0$ iff G^- contains a Hamiltonian path.

"$\Leftarrow$": Consider the directed graph $\vec{G} = (V, A_E)$ where each edge $\{v, w\} \in E$ is represented by two arcs, $(v, w), (w, v) \in A_E$. If G^- has a Hamiltonian path h^-, then G has a Hamiltonian cycle h and thus $\vec{G}$ has two arc-disjoint directed Hamiltonian circuit h_1 and h_2, corresponding to the two orientations in which h can be traversed on G. Therefore, based on the Hamiltonian cycle h we can find an Eulerian circuit e on $\vec{G}$ which starts and ends at vertex 1, and its last $n + 1$ vertices constitute h_2. Construct a sequence $\sigma = (\sigma_0, \ldots, \sigma_{4(k_2-k_1)|E|-1})$ in the following way:

- For positions $i \in \{0, \ldots, 4(k_2 - k_1)|E| - 1\}$ with $i \mod (k_2 - k_1) = 0$ sequence the jobs according to the arc-vertex sequence of the Eulerian circuit e, starting with the first arc of e, and using job VE_v^{λ} for the λ-th occurrence of vertex v in the circuit. Then, whenever a job representing an arc connecting vertices v_1 and v_2 in the tour is processed on machine M_{k_2}, a job representing vertex v_2 is processed on machine M_{k_1}. Whenever a job representing vertex v_1 is processed on machine M_{k_2}, a job representing an outgoing arc from vertex v_1 is processed on machine M_{k_2} with exception for the final visit of vertex 1 in the tour.
- For the remaining positions set $\sigma_i = D_l^k$ with $k = i \mod (k_2 - k_1)$ and $l = \lfloor i/(k_2 - k_1) \rfloor$.

Figure 3.7 shows an extract of a schedule representing an Eulerian cycle as a schedule for $m = 3, k_1 = 1$ and $k_2 = 3$.

The resulting schedule has no idle times on machines M_{k_1} except in the $(k_2 - k_1)$ cycles at the end of the schedule when the processing time on machine M_{k_2} is 1 in each cycle. Thus, the makespan is equal to d_2 and no job is finished later than d_2. Further, the VE jobs

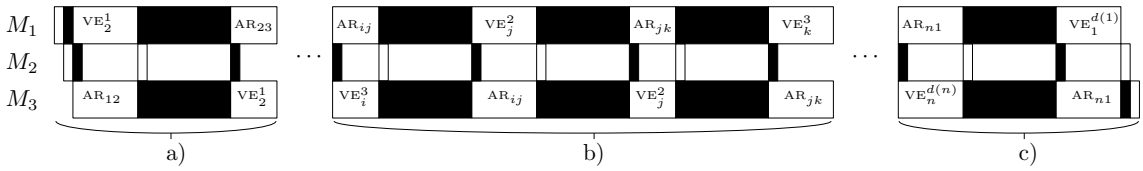


Figure 3.7: Representation of an Eulerian cycle as a schedule for $m = 3, k_1 = 1$ and $k_2 = 3$. The black jobs indicate the dummy jobs D. Shown are:

a) The first part of the schedule. The first job to be scheduled represents the first arc of e, connecting vertices 1 and 2. When this job is processed on machine M_3, the first VE job representing vertex 2 is processed on machine M_1.

b) An extract showing jobs that represent the arcs visiting vertex i for the third time, vertex j for the second time and vertex k for the third time.

c) The end of the schedule.

which appear in the last n pairs of $\mathrm{AR}-\mathrm{VE}$ jobs are $\mathrm{VE}_v^{d(v)}, v \in V$. The last job with due date d_1 that is completed in this schedule is a job of type VE_1^{λ} with $\lambda < d(1)$ which happens in position $p := (k_2 - k_1)(4|E| - 2n - 1)$. The job is completed $m - 1$ cycles later and the times of cycles c_i for $i = 0, \ldots, (k_2 - k_1)(4|E| - 2n - 1) + m$ are determined by the dummy jobs that are scheduled prior to position p (adding up to $(4|E| - 2n - 1)(k_2 - k_1 - 1)K$), the AR and VE jobs that are scheduled prior to p (adding up to $(2|E| - n)K$) and the next $m - k_2$ jobs that are scheduled after position p. Because this sums up to at most $\left(m - k_2 - \left\lfloor \frac{m-k_2}{2(k_2-k_1)} \right\rfloor\right) K$, the job is completed no later than its due date. Thus, no job in the schedule is late. Figure 3.8 shows the completion time of the last job with due date d_1 for $m = 6, k_1 = 1$ and $k_2 = 3$. In this case, the last term of the due date results in $3K$ and therefore the job is on time because the processing times of the jobs $\mathrm{AR}_{12}, \mathrm{VE}_2^{d(2)}$ and the dummy jobs are at most K.

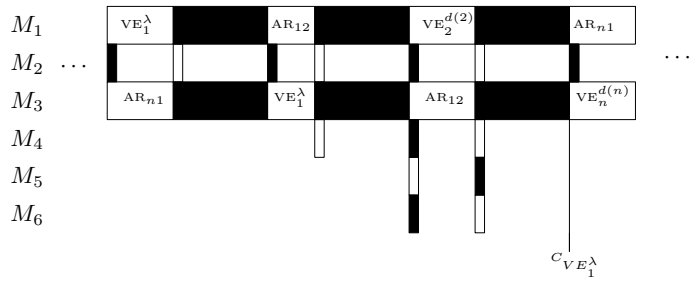


Figure 3.8: Completion time of the last job with due date d_1 for $m = 6, k_1 = 1$ and $k_2 = 3$

"⇒": Let $\sigma = \left(\sigma_0, \ldots, \sigma_{4(k_2-k_1)|E|-1}\right)$ be a sequence of jobs with $L_{\max} \leq 0$. Because the processing times on both machines, M_{k_1} and M_{k_2} sum up to $d_2-(k_2-k_1)$ there can not be any idle times on those two machines when there is a job processed on both machines if no job is to be completed later than d_2. Further, the first $(k_2 - k_1)$ jobs in the sequence need to have processing time 1 on machine M_{k_1} while the last (k_2-k_1) jobs in the sequence need to have processing time 1 on machine M_{k_2}. There are $4|E| \cdot (k_2 - k_1)$ jobs in total and $4|E| \cdot (k_2 - k_1 - 1)$ jobs have a processing time of K on at least one of the machines, while the other $4|E|$ jobs have a smaller processing time on each of the machines. Thus, to avoid idle times, the jobs of type D need to be scheduled in the sequence described in the first part of this proof. This leaves the AR and VE jobs to be scheduled in positions i for $i \bmod (k_2 - k_1) = k$ for some $k \in [0, \ldots, k_2 - k_1 - 1]$. By construction of the processing times, this implies that this results in an alternating sequence of AR and VE jobs, where each AR job has to represent an arc that connects the two VE jobs. Thus, the index sequence of the arc-vertex sequence represents an Eulerian circuit on the directed graph $\vec{G}$ which begins with an arc that leaves vertex 1 and ends in vertex 1. Since furthermore all jobs $\mathrm{VE}_v^{\lambda}, v \in V, \lambda < d(v)$ finish not later than d_1, the last n vertices constitute a directed Hamiltonian path on $\vec{G}$ that ends in 1. Consequently, G^- must have a Hamiltonian path as well. □

For minimizing the total completion time, the proof of Röck (1984a) can only be transfered to synchronous flow shop problems with two succeeding dominating machines. However, it would be a big surprise if the problem would in fact become easy (i.e. solvable in polynomial time) if non-dominating machines are introduced in between.

3.4 Extensions

In this section we consider the complexity of some extensions of flow shops with synchronous movement.

3.4.1 Idle jobs

As discussed in Section 2.4.1, $(n-1)(m-1)$ is an upper bound for the number of idle jobs for which an improvement can be achieved, or even $(n-1)(m-2)$ when minimizing the makespan. Therefore, for already polynomially solvable cases, we could extend the instance with the maximal required number of idle jobs and still retain an algorithm polynomial in the number of jobs and machines. However, in the case of synchronous flow shops with dominating machines in which the processing times on dominated machines are not equal to zero, introduction of idle jobs with a processing time of zero destroys the machine dominance and thus might increase the complexity of the problem.

An interesting effect occurs for problems with two non-consecutive dominating machines and zero processing times on dominated machines. While the complexity of this problem is still open for a fixed number of machines if no idle jobs are present, the introduction of idle jobs allows for a polynomial time algorithm which determines an optimal sequence for problem $Fm|\text{synmv,dom}\,(1,m)\,,\text{idle}, p_{ij}^{ndom} = 0|C_{\max}$ (see Algorithm 3.3).

Input: jobs $j = 1, \ldots, n$ with processing times p_{1j}, p_{mj}
Output: a sequence minimizing $C_{\max}$

```
J ← ∅
σ ← sequence of mn jobs, initially empty
for j = 1, ..., n do
    Add a job j' to J with processing times p_{1j'} = p_{1j}, p_{2j'} = p_{mj}
end
σ' ← optimal sequence for J via algorithm of Gilmore and Gomory
for j = 1, ..., n do
    σ_{(m-1)(j-1)+1} ← σ'_j
    for k = 2, ..., m − 1 do
        σ_{(m-1)(j-1)+k} ← idle job
    end
end
return σ
```

Algorithm 3.3: Algorithm for $Fm|\text{synmv,dom}\,(1,m)\,,\text{idle},p_{ij}^{ndom}=0|C_{\max}$

The idea is to transform an instance of this problem into an instance of $F2|\text{synmv}|C_{\max}$ and to solve the transformed instance with the algorithm of Gilmore and Gomory. Afterwards, the schedule is constructed in such a way that in the resulting schedule there is exactly one subsequence containing actual jobs (the optimal sequence for the two-machine case) and all other subsequences only contain idle jobs.

Theorem 3.13. *For problem $Fm|synmv,dom\,(1,m)\,,idle,p_{ij}^{ndom}=0|C_{\max}$, Algorithm 3.3 finds a sequence of jobs minimizing the makespan in $\mathcal{O}\,(mn+n\log n)$.*

Proof. The algorithm consists of the execution of the algorithm of Gilmore and Gomory for n jobs and constructing a schedule of size mn, thus the runtime is clear.

Let C be the makespan for the sequence σ as determined by Algorithm 3.3. As discussed in Section 2.3.2, C is the sum of the makespans of the subsequences $\sigma^\lambda = \left(\sigma_1^\lambda, \ldots, \sigma_n^\lambda\right)$ with $\sigma_h^\lambda = \sigma_{\lambda+(h-1)(m-1)}$ for $h = 1, \ldots, n$ and $\lambda = 1, \ldots, m-1$. As all subsequences except for σ^1 contain only idle jobs, their individual makespans sum up to zero and the makespan of subsequence σ^1 is C. Further, the jobs in σ^1 can not be sequenced in such a way that leads to a lower makespan, as C is the optimal makespan of the corresponding two-machine problem as determined by the algorithm of Gilmore and Gomory and the upper bound on idle jobs that can lead to an improvement of the makespan in the two-machine problem is 0. Thus, we can not achieve a better makespan than C with any sequence in which only a single subsequence contains non-idle jobs.

In the following, we will show that no better makespan can be achieved if more than one subsequence contains non-idle jobs. Let $\bar{\sigma}$ be such a sequence with makespan $\bar{C}$ and let w.l.o.g. the first $k \leq m-1$ subsequences contain non-idle jobs. Let $\bar{C}^\lambda$ be the makespan of the subsequence $\bar{\sigma}^\lambda = \left(\bar{\sigma}_1^\lambda, \ldots, \bar{\sigma}_n^\lambda\right)$ with $\bar{\sigma}_h^\lambda = \bar{\sigma}_{\lambda+(h-1)(m-1)}$ for $h = 1, \ldots, n$ and

$\lambda = 1, \ldots, m-1$. Construct a new sequence $\hat{\sigma}$ of length kmn and let the first subsequence $\hat{\sigma}^1$ be the concatenation of the subsequences $\bar{\sigma}^1, \ldots, \bar{\sigma}^k$, i.e.

$$\hat{\sigma}^1 = \left(\bar{\sigma}_1^1, \ldots, \bar{\sigma}_n^1, \bar{\sigma}_1^2, \ldots, \bar{\sigma}_n^2, \ldots, \bar{\sigma}_1^k, \ldots, \bar{\sigma}_n^k\right)$$

and let all other subsequences contain only idle jobs. Then,the makespan $\hat{C}$ of this schedule is at most $\hat{C} \leq \bar{C} = \sum_{\lambda=1}^{k} \bar{C}^{\lambda}$. Again, since C is the optimal makespan of the corresponding two-machine problem and the introduction of idle jobs can not lead to an improvement therein, it follows that $C \leq \hat{C} \leq \bar{C}$. Therefore, no better makespan can be achieved than the one obtained by Algorithm 3.3 if more than one subsequence contains non-idle jobs. □

3.4.2 Job splitting

In the following we will show that for both notions of splitting, arbitrary as well as restricted splitting, the synchronous flow shop problem becomes $\mathcal{NP}$-hard even for two machines.

Theorem 3.14. *$F2|synmv,split|C_{\max}$ is $\mathcal{NP}$-hard.*

Proof. We show this via a reduction from equal-size partition. Let (P) be an instance of partition with integers $a_1, \ldots, a_{2n}$ and $2M = \sum_{j=1}^{2n} a_j$. We are interested in finding two disjoint sets $A = \{\lambda_1, \ldots, \lambda_n\}, B = \{\mu_1, \ldots, \mu_n\}$ of equal size with $A \cup B = \{1, \ldots, 2n\}$, such that $\sum_{j=1}^{n} a_{\lambda_j} = \sum_{j=1}^{n} a_{\mu_j} = M$. We demand of the integers that there is a sufficiently large constant $\omega \in \mathbb{N}$ with :

- $a_i \geq \omega$ for all $i = 1, \ldots, 2n$
- $|a_i - a_j| \leq \frac{\omega}{n}$ for all $i, j = 1, \ldots, 2n$.

This can be achieved by adding ω to all integers which does not change the complexity of the partition problem. Adding this constant also leads to the condition that $a_i + a_j \geq a_k$ holds for all $i, j, k = 1, \ldots, 2n$.

We will now construct an instance (F) of $F2|$synmv,split$|C_{\max}$ with jobs $j = 1, \ldots, 2n$, processing times $p_j = a_j$ and no mandatory parts on any of the two machines. We will show that (P) has a solution iff there exists a schedule of the jobs in (F) with makespan $C_{\max} \leq C^* = \frac{1}{2} \sum_{j=1}^{2n} p_j = M$.

"$\Leftarrow$": Let $\sigma = (\sigma_1, \ldots, \sigma_{2n})$ be a schedule with the desired makespan C^* and let w.l.o.g. be $\sigma_j = j$ for all $j = 1, \ldots, 2n$. Because $\sum_{j=1}^{2n} p_j = 2C^*$, there are no idle times on either machine M_1 and M_2. Let

$$A = \{1 \leq \lambda \leq 2n | \lambda \bmod 2 = 1\}$$
$$B = \{1 \leq \mu \leq 2n | \mu \bmod 2 = 0\}$$

be two sets of equal size. In each cycle there is a job λ for $\lambda \in A$ processed on exactly one machine and as there are no idle times, it holds that

$$\sum_{\lambda \in A} p_\lambda = M.$$

Thus, also

$$\sum_{\lambda \in A} a_\lambda = \sum_{\mu \in B} a_\mu = M$$

and therefore we have a solution to (P).

$\Rightarrow$ Let $A = \{\lambda_1, \ldots, \lambda_n\}$ and $B = \{\mu_1, \ldots, \mu_n\}$ with $\sum_{i=1}^{n} a_{\lambda_i} = \sum_{i=1}^{n} a_{\mu_i} = M$ be the two disjoint equal-sized sets that represent a solution of the partition problem. Let the elements $\lambda_i \in A$ be ordered such that $a_{\lambda_i} \leq a_{\lambda_j}$ for $i \leq j$ and the elements $\mu_i \in B$ be ordered such that $a_{\mu_i} \geq a_{\mu_j}$ for $i \leq j$. We construct a schedule $\sigma = (\sigma_1, \ldots, \sigma_{2n})$ with

$$\sigma_i = \begin{cases} \lambda_{(i+1)/2} & \text{for } i \bmod 2 = 1 \\ \mu_{i/2} & \text{for } i \bmod 2 = 0 \end{cases} \quad (i = 1, \ldots, 2n).$$

Figure 3.9 shows the corresponding schedule.

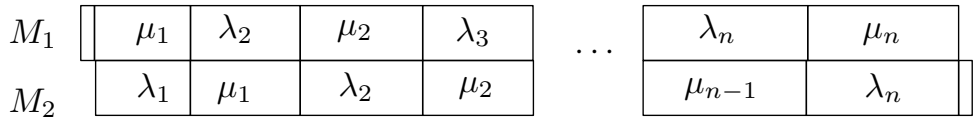


Figure 3.9: Schedule of the flow shop representing a solution of the partition problem

We need to show that we can split the jobs in this schedule in such a way that there are no idle times on any of the two machines. Then, as the processing times of all jobs sum up $2M$, there has to be a processing time of M on each machine. From this, it follows that the schedule has the desired makespan.

The jobs are split as follows: The first job λ_1 is processed for zero time units on machine M_1 and for p_{λ_1} time units on machine M_2. All succeeding jobs σ_i are processed on machine M_1 for the same processing time as job σ_{i-1} is processed on machine M_2 in the same cycle and split at this point. Thus, if the combined processing time of job σ_i is at least as large as the time for which job σ_{i-1} is processed on machine M_2, there are no idle times on any of the machines. In the following, we show that this requirement holds for all $i = 2, \ldots, 2n$.

If job λ_1 is processed for zero time units on machine M_1 and for p_{λ_1} time units on machine M_2, the succeeding job μ_1 has to be processed on machine M_1 for p_{λ_1} time units and thus is processed for $p_{\mu_1} - p_{\lambda_1}$ time units on machine M_2. If we number the cycles $c_0, \ldots, c_{2n}$ (cycle c_i being the cycle in which job σ_i is processed on machine M_2), it follows that job σ_i is processed on machine M_2 for the following amount of time:

Cycle $c_1 : p_{\lambda_1}$
Cycle $c_2 : p_{\mu_1} - p_{\lambda_1}$
Cycle $c_3 : p_{\lambda_2} + p_{\lambda_1} - p_{\mu_1}$

$$\text{Cycle } c_i : \sum_{l=1}^{i/2} (p_{\mu_l} - p_{\lambda_l}) \text{ for } i \bmod 2 = 0$$

$$\text{Cycle } c_i : \sum_{l=1}^{(i-1)/2} (p_{\lambda_l} - p_{\mu_l}) + p_{\lambda_{(i+1)/2}} \text{ for } i \bmod 2 = 1$$

A negative value for cycle c_i would indicate that the processing time of job σ_{i+1} is smaller than the amount of time job σ_i is processed on machine M_2. In this case, an idle time would occur on machine M_1, leading to a schedule with makespan larger than C^*. Thus, in the following we will show that all values are non-negative.

For each c_i with $i \bmod 2 = 1$ we get:

$$\left| \sum_{l=1}^{(i-1)/2} (p_{\lambda_l} - p_{\mu_l}) \right| \leq \sum_{l=1}^{(i-1)/2} | (p_{\lambda_l} - p_{\mu_l}) | \leq \sum_{l=1}^{(i-1)/2} \frac{\omega}{n} \leq \omega \leq p_{\lambda_{(i+1)/2}}$$

and thus $\sum_{l=1}^{(i-1)/2} (p_{\lambda_l} - p_{\mu_l}) + p_{\lambda_{(i+1)/2}} \geq 0$.

For each c_i with $i \bmod 2 = 0$ assume that $\sum_{l=1}^{k} p_{\mu_l} < \sum_{l=1}^{k} p_{\lambda_l}$ for some $1 \leq k \leq n$: Because of the ordering of the numbers within the sets A and B we get

$$kp_{\lambda_k} \geq \sum_{l=1}^{k} p_{\lambda_l} > \sum_{l=1}^{k} p_{\mu_l} \geq kp_{\mu_k}$$

and thus $p_{\lambda_k} > p_{\mu_k}$. However this leads to

$$\sum_{l=k+1}^{n} p_{\lambda_l} \geq (n-k-1)p_{\lambda_k} > (n-k-1)p_{\mu_k} \geq \sum_{l=k+1}^{n} p_{\mu_l}$$

contradicting $\sum_{l=1}^{n} p_{\lambda_l} = \sum_{l=1}^{n} p_{\mu_l}$. Therefore, $\sum_{l=1}^{k} p_{\mu_l} \geq \sum_{l=1}^{k} p_{\lambda_l}$ and thus $\sum_{l=1}^{i/2} (p_{\mu_l} - p_{\lambda_l}) \geq 0$ for all $i \bmod 2 = 0$. In the last cycle, the last job is processed for

$$\sum_{l=1}^{n} (p_{\mu_l} - p_{\lambda_l}) = \sum_{i=1}^{n} a_{\lambda_i} - \sum_{i=1}^{n} a_{\mu_i} = 0$$

time units on machine M_2 leading to no idle times on machine M_1 in this cycle. As all values are non-negative, there are no idle times on any of the two machines and the schedule yields the desired makespan.

□

In the case of restricted splitting with mandatory parts, the problem is even strongly $\mathcal{NP}$-hard.

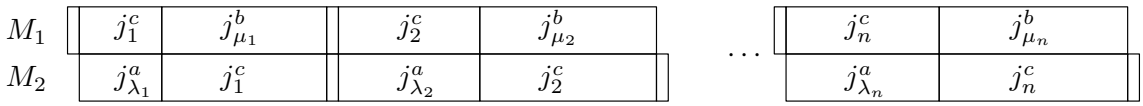


Figure 3.10: Schedule of the flow shop representing a solution of the numerical matching with target sums

Theorem 3.15. *$F2|synmv,r\text{-}split|C_{\max}$ is strongly $\mathcal{NP}$-hard.*

Proof. We show this via a reduction from numerical matching with target sums: Given two disjoint multisets of integers $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ as well as a target multiset of integers $C = \{c_1, \ldots, c_n\}$, can the integers in A and B be partitioned into n disjoint sets D_i, each containing exactly one element from A and B such that $c_i = a_{\lambda_i} + b_{\mu_i}$ with $a_{\lambda_i}, b_{\mu_i} \in D_i$?

Let (T) be an instance of numerical matching with target sums with integer sets A, B, C. Construct an instance of the flow shop problem (F) in the following way: Let $\omega \geq \sum_{i=1}^{n} c_i + 1$ and create the following jobs:

- For each $a_i \in A$ create a job j_i^a with processing time $p_{j_i^a} = a_i + \omega$ and mandatory processing times $\underline{p}_{2,j_i^a} = \overline{p}_{2,j_i^a} = p_{j_i^a}$.
- For each $b_i \in B$ create a job j_i^b with processing time $p_{j_i^b} = b_i + 2\omega$ and mandatory processing times $\underline{p}_{1,j_i^b} = \overline{p}_{1,j_i^b} = p_{j_i^b}$.
- For each $c_i \in C$ create a job j_i^c with processing time $p_{j_i^c} = c_i + 3\omega$ and mandatory processing times $\underline{p}_{1,j_i^c} = \underline{p}_{2,j_i^c} = 0$.

The mandatory parts are chosen in such a way that all jobs of type j^a have to be processed fully on machine M_2, all jobs of type j^b have to be processed fully on machine M_1 and all jobs of type j^c may be split arbitrarily.

Then, the instance (T) allows for a numerical matching with target sums iff the instance (F) of the flow shop problem can be scheduled with a makespan of

$$C_{\max} \leq C^* = 3n\omega + \sum_{i=1}^{n} c_i.$$

"⇒": Let $c_i = a_{\lambda_i} + b_{\mu_i}$ for all $i = 1, \ldots, n$ be a matching for (T). Then the following schedule results in the desired makespan: For each i let job $j_{\lambda_i}^a$ be directly followed by job j_i^c which is processed for $a_i + \omega$ time periods on machine M_1 and for $b_i + 2\omega$ time slots on machine M_2. Afterwards job $j_{\mu_i}^b$ is processed, followed by the next job $j_{\lambda_{i+1}}^a$ (see Figure 3.10). This way, there are no idle times on either of the two machines and the desired makespan is reached as the schedule results in a makespan of $C_{\max} = \sum_{i=1}^{n} p_{j_i^c} = 3n\omega + \sum_{i=1}^{n} c_i$.

"⇐": Let σ be a schedule of (F) with makespan $C_{\max} \leq C^*$. Let P be the sum of processing times of jobs of type j^c on machine M_1 in this schedule. Then, the sum of processing times on machine M_1 adds up to

$$\sum_{i=1}^{n} p_{j_i^b} + P = 2n\omega + \sum_{i=1}^{n} b_i + P$$

and the sum of processing times on machine M_2 adds up to

$$\sum_{i=1}^{n} p_{j_i^a} + \sum_{i=1}^{n} p_{j_i^c} - P = n\omega + \sum_{i=1}^{n} a_i + 3n\omega + \sum_{i=1}^{n} c_i - P.$$

Because of the desired makespan, the sum of processing times on both machines must be at most C^* and thus

$$P = n\omega + \sum_{i=1}^{n} a_i = \sum_{i=1}^{n} p_{j_i^a}$$

has to hold. Further, there may be no idle times on both machines as any idle time would result in a makespan larger than C^*. For any two h, i, job j_h^a may be followed neither by job j_i^a nor job j_i^b as this would lead to an idle time of $\omega + a_h > 0$ on machine M_1 in the former and an idle time of $\omega + b_i - a_h > 0$ on machine M_2 in the latter case. Therefore, a job of type j^a has to be succeeded by a job of type j^c. If for some h, i, job j_i^c succeeds job j_h^a, to avoid idle times, job j_i^c has to be split in a way that its processing time on machine M_2 is $3\omega + c_i - (\omega + a_h) = 2\omega + (c_i - a_j)$. Therefore, to avoid idle times on machine M_1, a matching job of type j^b has to succeed job j_h^c. Iteratively, it follows that the schedule has to be as described in the first part of the proof and as depicted in Figure 3.10. The corresponding matching can then be read from the schedule, each target value c_i represented by the job j_i^c is reached by adding the integers a_{λ_i} and b_{μ_i} which are represented by the jobs $j_{\lambda_i}^a$ and $j_{\mu_i}^b$ that are processed at the same time as job j_i^c. □

3.4.3 Resources

Introducing pallet-like resources as described in Section 2.4.3, the problem becomes $\mathcal{NP}$-hard even for the problem $F2|\text{synmv}, \text{dom}(1), \text{jres} \cdot 11, p_{ij} = 1|C_{\max}$. For the following proof we will assume that each job requires pallet-like resources which need to be present for all operations of the job and are free again as soon as the job is completed.

Theorem 3.16. *The problem $F2|synmv, dom(1), jres \cdot 11, p_{ij} = 1|C_{\max}$ is strongly $\mathcal{NP}$-hard.*

Proof. We show this by reduction from Hamiltonian path (see Garey and Johnson (1979)): Given a graph $G = (V, E)$, is there a path through all vertices V that visits every vertex exactly once?

We construct an instance (SF) of $F2|\text{synmv}, \text{dom}(1), \text{jres} \cdot 11|C_{\max}$ for a given graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ in the following way: For each vertex $v \in V$ add a job v

with processing times $p_{1v} = p_{2v} = 1$. For each pair of vertices $u, v \in V$ with $u < v$ that are not connected by an edge $\{u, v\} \in E$, add a resource r_{uv} and let both jobs u and v require resource r_{uv}, creating $\frac{n(n-1)}{2}$ resources. As a result, two jobs u and v can be processed concurrently on machines M_1 and M_2 if and only if their respective vertices are connected by an edge $\{u, v\} \in E$. Then, there exists a schedule σ with makespan $n+1$ in (SF) iff G contains a Hamiltonian path.

"$\Leftarrow$": Let $P = (v_1, v_2, \ldots, v_n)$ be a Hamiltonian path in G. Then the vertices v_i, v_{i+1} are connected by an edge and thus the schedule $\sigma = (\sigma_1, \ldots, \sigma_n)$ with $\sigma_i = v_i$ has a makespan of $n+1$ in (SF).

"$\Rightarrow$": Conversely, let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be a schedule with makespan $n+1$ in (SF). As two jobs can only be scheduled directly succeeding each other if their corresponding vertices in G are connected by an edge, this leads to a path of length n in G. As every job corresponds to a distinct vertex, this path is Hamiltonian. □

In most cases, the presence of resources also involves changeovers. We will discuss this extension in the next subsection.

3.4.4 Changeovers

The following theorem shows that in the presence of job families and changeovers the synchronous flow shop on two machines is $\mathcal{NP}$-hard even if changeover times are constant.

Theorem 3.17. *Problem* $F2|synmv, circ\text{-}s_{fg} = s|C_{\max}$ *is strongly* $\mathcal{NP}$*-hard*

Proof. We prove this via a pseudo-polynomial reduction from numerical matching with target sums. Let (T) be an instance of numerical matching with target sums with multisets of integers $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$ and a multiset of target integers $C = \{c_1, \ldots, c_n\}$. W.l.o.g. we assume $a_i, b_i \geq 2$ for all i. Let $\chi := \left(\sum_{j=1}^{n} c_j\right)$.

We construct the following instance (SF) of the synchronous flow shop problem with a set $\mathcal{J}$ consisting of $2\chi + 1$ jobs in $3n + 1$ job families. Let α, β, γ and s be large constants with $1 << \alpha < \beta < \gamma << s$ and create the following jobs:

- For c_1 one job family J_1^C with one job j_{11}^C with processing times $(0, \alpha)$, one job $j_{1c_1}^C$ with processing times $(2, \beta)$ and $c_1 - 2$ jobs j_{1k}^B with processing times $(2, 1)$ for $k = 2, \ldots, c_1 - 1$.
- For c_i with $i = 2, \ldots, n$ one job family J_i^C, each with one job j_{i1}^C with processing times (γ, α), one job $j_{ic_i}^C$ with processing times $(2, \beta)$ and $c_i - 2$ jobs j_{1k}^C with processing times $(2, 1)$ for $k = 2, \ldots, c_i - 1$.
- For a_i with $i = 1, \ldots, n$ one job family J_i^A, each with one job j_{i1}^A with processing times $(\alpha, 2)$ and $a_i - 1$ jobs j_{ik}^A with processing times $(1, 2)$ for $k = 2, \ldots, a_i$.
- For b_i with $i = 1, \ldots, n$ one job family J_i^B, each with one job $j_{ib_i}^B$ with processing times (β, γ) and $b_i - 1$ jobs j_{ik}^B with processing times $(1, 2)$ for $k = 1, \ldots, b_i - 1$.
- One dummy job D, being its own job family with processing times $(\gamma, 0)$.

We show that there is a solution of (T) iff the synchronous flow shop (SF) has a schedule with makespan

$$\begin{aligned} C_{\max} &\leq (3n-1)s + n(\alpha+\beta+\gamma) + 3\chi - 4n \\ &= (3n-1)s + \sum_{j=1}^{n} p_{1j} \end{aligned}$$

"⇒": Let $c_i = a_{\lambda_i} + b_{\mu_i}$ for all $i = 1, \ldots, n$ be a numerical matching for (T). Then we construct a sequence $\sigma = (\sigma_1, \ldots, \sigma_{2\chi+1})$ for (SF) by:

- For $i = 1, \ldots, n$ schedule job j^C_{ik} at position $2\left(\sum_{j=1}^{i-1} c_j + k\right) - 1$ for $k = 1, \ldots, c_i$.
- For $i = 1, \ldots, n$ schedule job $j^A_{\lambda_i k}$ at position $2\left(\sum_{j=1}^{i-1} c_j + k\right)$ for $k = 1, \ldots, a_{\lambda_i}$.
- For $i = 1, \ldots, n$ schedule job $j^B_{\mu_i k}$ at position $2\left(\sum_{j=1}^{i-1} c_j + a_{\lambda_i} + k\right)$ for $k = 1, \ldots, b_{\lambda_i}$.
- Schedule job D at position $2\chi + 1$.

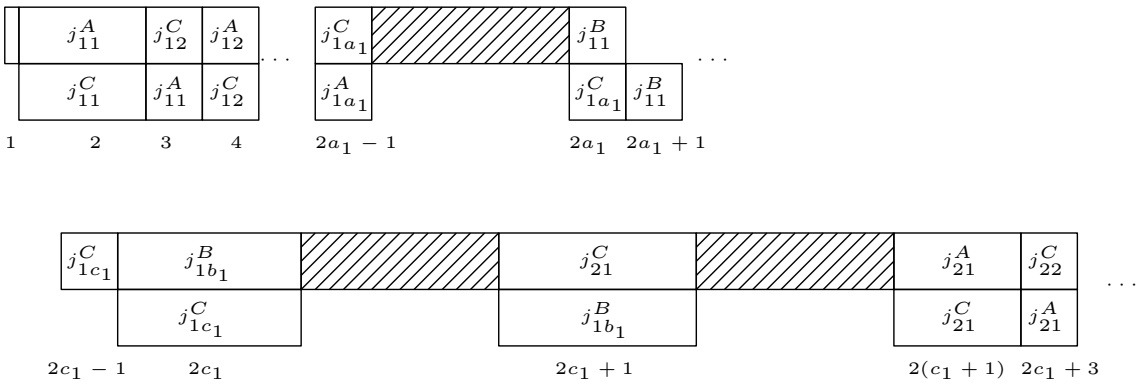


Figure 3.11: Schedule of jobs representing the first numerical matching with $c_1 = a_1 + b_1$. The numbering of the cycles is indicated below the respective cycles. Changeovers are represented by the striped boxes. There is a changeover between the last job of family J^A_1 and the first job of family J^B_1, between the last job of family J^C_1 and the first job of family J^C_2 and between the last job of family J^B_1 and the first job of family J^A_2.

Figure 3.11 shows an extract of a corresponding schedule depicting the first numerical matching with $c_1 = a_1 + b_1$. Note, that in this schedule there is no idle time on either machine M_1 or M_2. Thus, the processing times add up to $\sum_j p_{1j} = n(\alpha+\beta+\gamma)+3\chi-4n$. Further, all job families are scheduled in a way that as soon as a job of a job family f is scheduled in position i, all other jobs of the family are sequenced immediately afterwards

in positions $i + 2l$ for $l = 1, \ldots, |f| - 1$, leading to exactly one changeover needed for each job family. Thus, the time needed for changeovers adds up to $(3n - 1)s$ and the desired makespan is achieved.

"$\Leftarrow$": Let conversely $\sigma = (\sigma_1, \ldots, \sigma_{2\chi+1})$ be a sequence with makespan

$$C^* \leq (3n - 1)s + n(\alpha + \beta + \gamma) + 3\chi - 4n.$$

We will show that the corresponding schedule is of the same structure as the one in the first part of the proof.

First, we can assure that as soon as a job of a job family f is scheduled in position i, all other jobs of the family are sequenced immediately afterwards in positions $i + 2l$ for $l = 1, \ldots, |f| - 1$ as this results in the least amount of changeovers. If only one job family would be split, i.e. an additional changeover to a job family was necessary, this would immediately result in at least $3n$ changeovers and to a makespan of $C_{\max} \geq 3ns > C^*$. If there is only one changeover necessary for each job family, the time required for changeovers sums up to $(3n - 1)s$.

Further, to assure a processing time of at most $n(\alpha+\beta+\gamma)+3\chi-4n = \sum_j p_{1j} = \sum_j p_{2j}$, there may be no idle times on machines M_1 and M_2. Thus, the first job to be processed must be a job with processing time 0 on machine M_1 and the last job to be processed must have a processing time of 0 on machine M_2. Thus, the first job to be processed must be job j^C_{11} and the last job to be processed must be job D.

As job j^C_{11} has to be processed in the first position, the whole job family j^C_1 has to be processed in the first c_1 odd positions $1 + 2l$ for $l = 0, \ldots, c_1 - 1$. Further, as the job j^C_{11} has a processing time of α on the second machine, a job of type j^A_{h1} has to be processed in the second position. Then, all jobs of the job family J^A_h have to be processed in the first a_h even cycles. Because of job j^C_{c1} with processing time of β on machine M_2, a job family of type j^B_i for some i has to follow the job family j^A_h in the next b_i even cycles to avoid idle times. Also, job $j^B_{ib_i}$ with processing time β on machine M_1 has to be succeeded either by the dummy job D or another job family of type J^C because of its processing time of γ on machine M_2. As discussed above, the dummy job has to be scheduled at the last position in the sequence and thus for all other positions the succeeding job cannot be the dummy job D but has to be a job of type j^C with a processing time of γ on machine M_1. Iteratively, all job families of type J^C have to be scheduled in the odd cycles while all job families of type J^A and J^B have to be scheduled alternating in the even cycles. The last cycle is occupied by the dummy job D. Further, consider the first job family J^C_1 which is scheduled in positions $2l + 1$ for $l = 1, \ldots, c_1$. To avoid idle times, the job $j^C_{1c_1}$, job $j^B_{ib_i}$, a job of type j^C_{k1} for some k and a job of type j^A_{o1} for some o have to be scheduled in direct succession, see Figure 3.11. In this case, the size of the job family j^C_1 is equal to the sum of the sizes of job families j^B_i and j^A_h, i.e. $c_1 = a_h + b_i$. Iteratively, the same holds true for all succeeding job families J^C. Thus, a solution to (T) can be read from the schedule. $\square$

3.4.5 Order scheduling

Obviously, order scheduling problems are at least as difficult as their corresponding counterparts with individual jobs. As discussed in Section 2.4.5, the presence of orders makes no difference when considering problems with the objective functions $C_{\max}$ or $L_{\max}$. Thus, the corresponding complexity results obtained in this chapter also apply to these cases. The polynomial time algorithm 3.2 for the problem $F|\text{synmv}, \text{dom}\left(\mathcal{I}\right), p_{ij}^{ndom} = 0|L_{\max}$ with $|\mathcal{I}| = 1$ can be used for order scheduling as well, by just splitting the orders into individual jobs. For the problem $F|\text{synmv}, \text{dom}\left(\mathcal{I}\right), p_{ij}^{ndom} = 0|\sum C_j$ with $|\mathcal{I}| = 1$ the orders can be sorted in SPT order like in the case for individual jobs to achieve an optimal schedule. This can once again be shown via an exchange argument, similar to Theorem 3.5. While these algorithms remain polynomial in the number of jobs, they are not polynomial in the input, as for each order only its corresponding product type and volume has to be specified.

3.5 Summary

In this chapter we derived some complexity results for flow shops with synchronous movement, which are summarized in Table 3.1. In the first part of the table the general situation (i.e. without machine dominance) is considered. Afterwards, results for sets $\mathcal{I}$ of dominating machines with cardinalities $|\mathcal{I}| = 1$ and $|\mathcal{I}| = 2$ are shown. The third part shows results for the extensions discussed in this thesis.

$F2\|\text{synmv}\|C_{\max}$	$\mathcal{O}(n \log n)$	Gilmore and Gomory (1964)
$F2\|\text{synmv}\|\sum C_j$	str. $\mathcal{NP}$-hard	Röck (1984a)
$F2\|\text{synmv}\|L_{\max}$	str. $\mathcal{NP}$-hard	Röck (1984a)
$F3\|\text{synmv}\|C_{\max}$	str. $\mathcal{NP}$-hard	Theorem 3.1
$\|\mathcal{I}\| = 1$:		
$F\|\text{synmv}, \text{dom}\,(\mathcal{I})\,\|C_{\max}$	str. $\mathcal{NP}$-hard	Theorem 3.3
$F\|\text{synmv}, \text{dom}\,(\mathcal{I})\,, p_{ij}^{ndom} = 0\|C_{\max}$	$\mathcal{O}(n)$	Section 3.3.1
$Fm\|\text{synmv}, \text{dom}\,(\mathcal{I})\,\|C_{\max}$	$\mathcal{O}(n^m)$	Section 3.3.1
$F\|\text{synmv}, \text{dom}\,(\mathcal{I})\,\|\sum C_j$	str. $\mathcal{NP}$-hard	Theorem 3.4
$F\|\text{synmv}, \text{dom}\,(\mathcal{I})\,, p_{ij}^{ndom} = 0\|\sum C_j$	$\mathcal{O}(n \log n)$	Theorem 3.5
$Fm\|\text{synmv}, \text{dom}\,(\mathcal{I})\,\|\sum C_j$	$\mathcal{O}(n^m \log n)$	Corollary 3.6
$F\|\text{synmv}, \text{dom}\,(\mathcal{I})\,, p_{ij}^{ndom} = 0\|L_{\max}$	$\mathcal{O}(n^3 \log n)$	Theorem 3.8
$Fm\|\text{synmv}, \text{dom}\,(\mathcal{I})\,\|L_{\max}$	$\mathcal{O}(n^{m+2} \log n)$	Corollary 3.9
$F2\|\text{synmv}, \text{dom}\,(\mathcal{I})\,, p_{ij}^{ndom} = 0\|\sum w_j U_j$	$\mathcal{NP}$-hard	Lawler and Moore (1969)
$F2\|\text{synmv}, \text{dom}\,(\mathcal{I})\,, p_{ij}^{ndom} = 0\|\sum T_j$	$\mathcal{NP}$-hard	Du and Leung (1990)
$\|\mathcal{I}\| = 2$:		
$F\|\text{synmv}, \text{dom}\,(\mathcal{I})\,, p_{ij}^{ndom} = 0\|C_{\max}$	str. $\mathcal{NP}$-hard	Theorem 3.11
$Fm\|\text{synmv}, \text{dom}\,(\mathcal{I})\,, p_{ij}^{ndom} = 0\|L_{\max}$	$\forall m$ str. $\mathcal{NP}$-hard	Röck (1984a), Theorem 3.12
$F\|\text{synmv}, \text{dom}\,(k, k+1)\,, p_{ij}^{ndom} = 0\|C_{\max}$	$\mathcal{O}(n \log n)$	Gilmore and Gomory (1964)
$Fm\|\text{synmv}, \text{dom}\,(k, k+1)\,\|C_{\max}$	$\mathcal{O}(n^{m-1} \log n)$	Gilmore and Gomory (1964)
$Fm\|\text{synmv}, \text{dom}\,(k, k+1)\,, p_{ij}^{ndom} = 0\|\sum C_j$	$\forall m$ str. $\mathcal{NP}$-hard	Röck (1984a)
With idle jobs, $\|\mathcal{I}\| = 2$:		
$Fm\|\text{synmv,dom}\,(\mathcal{I})\,, \text{idle}, p_{ij}^{ndom} = 0\|C_{\max}$	$\mathcal{O}(nm + n \log n)$	Theorem 3.13
With job splitting:		
$F2\|\text{synmv,split}\|C_{\max}$	$\mathcal{NP}$-hard	Theorem 3.14
$F2\|\text{synmv,r-split}\|C_{\max}$	str. $\mathcal{NP}$-hard	Theorem 3.15
With resources:		
$F2\|\text{synmv}, \text{dom}\,(1)\,, \text{jres} \cdot 11, p_{ij} = 1\|C_{\max}$	str. $\mathcal{NP}$-hard	Theorem 3.16
With changeovers:		
$F2\|\text{synmv,circ-}s_{fg} = s\|C_{\max}$	str. $\mathcal{NP}$-hard	Theorem 3.17

Table 3.1: Complexity results for flow shop problems with synchronous movement

Chapter 4

Exact methods

In the previous chapter we already discussed some algorithms to optimally solve special cases of synchronous flow shop problems. In Section 3.3.1 we described polynomial time algorithms to minimize the total completion time as well as the maximum lateness in synchronous flow shops with a single dominating machine. Further, the algorithm of Gilmore and Gomory (1964) can be applied to solve synchronous flow shops with two adjacent dominating machines to optimality in polynomial time. For reasons of self-containment we will recite the algorithm of Gilmore and Gomory (1964) in this chapter. Later, concepts of this algorithm will be used for obtaining lower bounds as well as heuristic algorithms.

Unfortunately, for the general case, finding optimal solutions for the synchronous flow shop is an $\mathcal{NP}$-hard problem for all of the objective functions contemplated in this thesis. Thus, unless $\mathcal{P} = \mathcal{NP}$, we can not expect to derive a polynomial time algorithm for these cases. In this chapter we will discuss two classes of exact methods for obtaining optimal solutions for hard optimization problems. First, we will show how to model synchronous flow shop problems as mixed integer linear programs (MILP). While this does not reduce the complexity of the problem, there exist very efficient solvers to derive optimal solutions for MILPs. Another class of algorithms consists of branch and bound methods which are based on removing regions from the solution space that can not contain optimal solutions and thus speed up the solution procedure by concentrating on the non-removed regions. In this chapter we will discuss lower bounds for instances of synchronous flow shop problems that can be used in such a branch and bound scheme. The evaluation of the exact algorithms described here can be found in Chapter 7.

The remainder of this chapter is structured as follows. In Section 4.1, we will lay out the algorithm of Gilmore and Gomory (1964) for the polynomially solvable special case $F2|\text{synmv}|C_{\max}$. In Section 4.2 synchronous flow shop problems will be modeled as mixed integer linear programs. Formulations will be presented for all of the objective functions, special cases and extensions that are discussed in Section 2.4. In Section 4.3, we will discuss branch and bound algorithms that make use of lower bounds which are presented in Section 4.4.

4.1 Gilmore and Gomory's algorithm for $F2|\text{synmv}|C_{\max}$

In this section, we give a short recapitulation of the algorithm of Gilmore and Gomory (1964) that can be used to solve the problem $F2|\text{no-wait}|C_{\max}$. As discussed in Chapter 3 it can thus be applied to obtain an optimal solution for $Fm|\text{synmv}, \text{dom}\,(i, i+1)\,|C_{\max}$ and $F|\text{synmv}, \text{dom}\,(i, i+1)\,, p_{ij}^{ndom} = 0|C_{\max}$ in polynomial time. In the following, we will describe the algorithm for the case $F2|\text{synmv}|C_{\max}$. The correctness of the algorithm and all intermediate results are proven in Gilmore and Gomory (1964). As the proof is rather long, we will omit it in this thesis and only state the algorithm.

To improve readability, we define $a_j := p_{1j}$ and $b_j := p_{2j}$ as the processing times of job j on the two machines. For two consecutive jobs i, j the cycle time of the cycle in which these jobs are processed on machine M_1 and M_2, respectively can be calculated by

$$c_{ij} = \max\left(b_i, a_j\right).$$

If we introduce another job 0 with $a_0 = b_0 = 0$, the costs $c_{0j} = \max\left(0, a_j\right)$ and $c_{j0} = \max\left(b_j, 0\right)$ are equal to the cycle time of the first or last cycle if job j is processed in the first or the last position of the schedule. The task to find a schedule which minimizes the sum of cycle times and thus the makespan can now be regarded as a traveling salesman problem containing nodes $0, \ldots, n$ representing the jobs with distances c_{ij} defined by the cycle times that occur if job j is succeeding job i. An optimal tour with these distances is equivalent to a schedule with optimal makespan: Node 0 represents the start and end of such a schedule and the jobs are scheduled in the sequence given by the tour.

In general, the traveling salesman problem is $\mathcal{NP}$-hard (see e.g. Garey and Johnson (1979)). However, with the special distance metric defined above, the problem can be solved in polynomial time. The basic idea of the algorithm is to assign to each node its successor in the tour by finding an optimal matching of b- to a-values. However, in general, this leads to several disjoint circuits which then need to be joined into a single optimal tour.

In the following, we describe Algorithm 4.1 by Gilmore and Gomory in detail. We denote with ϕ_j the successor of job j, i.e. the job we want to schedule after job j. At first, sort the jobs in non-decreasing order of b_j. Then, the initial successor for each job is determined by a bipartite matching of b- to a-values. With the special metric this can be achieved by sorting the a-values and setting $\phi_k = l$ with l being the job with the k-th smallest a-value (line 5). This can be modeled as a graph G consisting of nodes $0, \ldots, n$ and edges between nodes i, j if and only if $\phi_i = j$. In general, the graph consists of multiple disjoint small circuits. Figure 4.1 shows an exemplary matching for 8 jobs which results in an initial permutation $(4, 8, 1, 3, 6, 5, 7, 2)$, leading to the disjoint small circuits depicted in Figure 4.2.

Let $C_1, \ldots, C_m$ be the connected components of G, all of which are directed circuits. We can connect any two components C_i and C_j into a single circuit by interchanging two edges in the graph G: Let i_1, i_2 be two consecutive nodes in C_i and j_1, j_2 be two consecutive nodes in C_j. An interchange R_{i_1,j_1} consists of removing the directed edges

Input: set of jobs J with processing times $a_j = p_{1j}, b_j = p_{2j}$
Output: sequence which minimizes $F2|\text{synmv}|C_{\max}$
1 Add job 0 with processing times $a_0 = b_0 = 0$ to J
2 Sort the jobs non-decreasingly in b_j
3 $A \leftarrow$ list of jobs $j = 0, \ldots, n$, sorted non-decreasingly in a_j
4 $\phi \leftarrow$ list of size $n + 1$
5 $\phi_j \leftarrow A_j$ for all $j = 0, \ldots, n$
6 $G = (V, E)$ with $V = \{0, \ldots, n\}$ and $E = \{(j, \phi_j) | j = 0, \ldots, n\}$
7 $c_{ij} \leftarrow \max(b_i, a_j)$ for all $i, j \in V$
8 $r_{j,j+1} \leftarrow -\left(c_{j,\phi_j} + c_{j+1,\phi j+1}\right) + c_{j,\phi_{j+1}} + c_{j+1,\phi_j}$ for all $j < n$
9 $R \leftarrow \emptyset$ set of interchanges
10 **while** G *consists of more than one component* **do**
11 Find smallest value $r_{j,j+1}$ such that j and $j+1$ are in distinct components
12 Add $(j, j+1)$ to R
13 Add $(j, j+1)$ to E to join the components
14 **end**
15 Remove all edges $(j, j+1) \in R$ from E
16 $S_1 \leftarrow$ list of $(j, j+1) \in R$ with $a_{\phi_j} \geq b_j$, decreasing in j
17 $S_2 \leftarrow$ list of $(j, j+1) \in R$ with $a_{\phi_j} < b_j$, increasing in j
18 Make interchanges $(j, j+1) \in S_1$, then interchanges $(j, j+1) \in S_2$
19 $\sigma \leftarrow$ sequence with length n
20 $\sigma_1 \leftarrow \phi_0$
21 **for** $i = 2, \ldots, n$ **do**
22 $\sigma_i \leftarrow \phi_{\sigma_{i-1}}$
23 **end**
24 **return** σ

Algorithm 4.1: Algorithm of Gilmore and Gomory for $F2|\text{synmv}|C_{\max}$

(i_1, i_2) and (j_1, j_2) and adding the directed edges (i_1, j_2) and (j_1, i_2), thus creating a single circuit. The costs of such an interchange can be calculated by

$$c(R_{i_1,j_1}) = -(c_{i_1 i_2} + c_{j_1 j_2}) + c_{i_1 j_2} + c_{j_1 i_2}$$

To join the components, a variation of the algorithm of Kruskal (1956) can be applied. In its original version, Kruskal's algorithm is used to find a minimal spanning tree. Starting with an empty graph, the algorithm iteratively adds an edge with minimum cost which does not create a cycle. Similarly, in lines 11-13 of Algorithm 4.1 the disjoint components are joined by greedily applying an interchange with minimum cost which connects two disjoint components of the graph G. Gilmore and Gomory were able to show that only interchanges $R_{i,i+1}$ of two jobs which directly succeed each other in the sorted sequence of jobs need to be considered for this. Whenever an interchange $R_{j,j+1}$ is chosen to be made, we connect the corresponding components with a directed edge $(j, j+1)$. After we

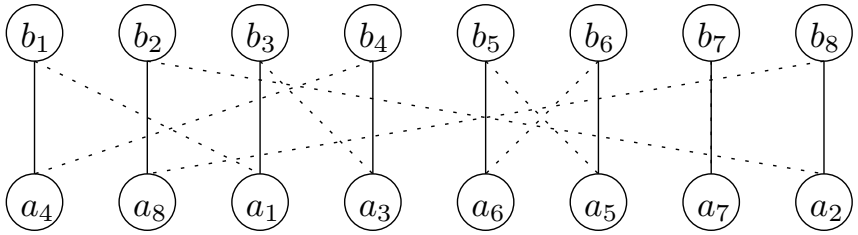


Figure 4.1: Exemplary matching to find an initial permutation for 8 nodes. The dotted lines connect the a_j- and b_j-values of the same job. Regular lines connect b_k to the k-th smallest a-value.

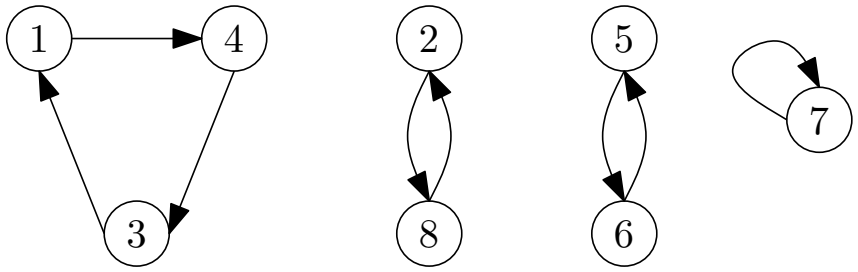


Figure 4.2: Disconnected circuits resulting from the mapping in Figure 4.1.

reached a single component and saved the interchanges required for this, we again remove these edges from the graph.

The next step is to join the individual components in G to form a single circuit using the minimal interchanges determined before. Note that two interchanges might interfere with each other and thus the total cost of the resulting tour might differ depending on the sequence in which the interchanges are performed. Gilmore and Gomory were able to prove that an optimal series of interchanges can be obtained by sorting them into two groups S_1 and S_2. Those interchanges $R_{j,j+1}$ for which $a_{\phi_j} \geq b_j$ are assigned to group S_1, the others to group S_2, respectively. In lines 16 and 17 the interchanges in group S_1 are sorted in order of decreasing index j and the interchanges in group S_2 are sorted in order of increasing index j.

A single circuit is then obtained by first executing all interchanges in group S_1 and then executing the interchanges S_2 in the defined order. Figure 4.3 depicts a single tour of our example after making the interchanges $R_{7,8}, R_{6,7}$ and $R_{2,3}$ in this order. Gilmore and Gomory showed that the resulting tour is an optimal solution to the TSP with the special metric. Therefore, the schedule $\sigma = (\sigma_1, \ldots, \sigma_n)$ with $\sigma_1 = \phi_0, \sigma_i = \phi_{\sigma_{i-1}}$ for $i = 2, \ldots, n$ is an optimal solution of the synchronous flow shop problem.

We can also use Algorithm 4.1 if we are given a starting sequence $\sigma = (\sigma_1, \ldots, \sigma_l)$ of jobs that are fixed in the beginning of the schedule and we want to determine an optimal sequence σ' with $\sigma'_i = \sigma_i$ for $i = 1, \ldots, l$. In this case, the cycle time of the cycle in which

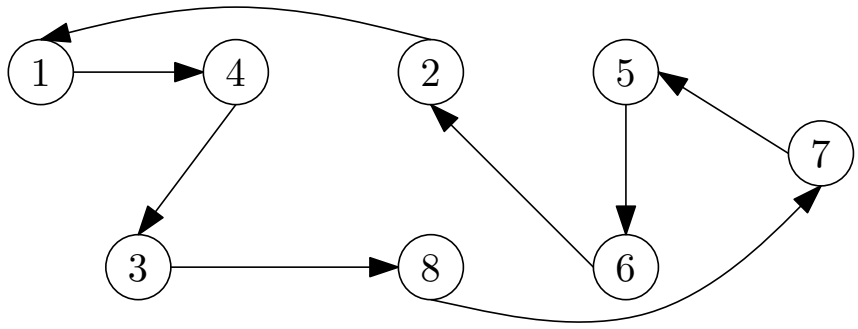


Figure 4.3: Resulting tour from Figure 4.2 after making interchanges $R_{7,8}, R_{6,7}$ and $R_{2,3}$.

job σ'_{l+1} is processed on machine M_1 is determined by $\max\left(b_{\sigma'_l}, a_{\sigma'_{l+1}}\right)$. By changing the attributes of job 0 in line 1 of Algorithm 4.1 such that $a_0 = 0$ and $b_0 = p_{\sigma'_l 2}$ we can again model the problem of minimizing the sum of cycle times of the jobs $J' = J \backslash \{\sigma_i | i = 1, \ldots, l\}$ as a traveling salesman problem through nodes $\{0\} \cup J'$. The node that succeeds 0 in the tour relates to the job σ'_{l+1} that is to be scheduled in position $l+1$ of the schedule and the cycle time of the $(l+1)$-st cycle is equal to the cost $c_{0,\sigma'_{l+1}}$. An optimal makespan for a schedule with this fixed starting sequence is then equal to the completion time of job σ_{l-1} in the fixed starting sequence σ plus the makespan determined for the jobs $\{0\} \cup J'$ by Algorithm 4.1.

4.2 Mixed integer linear programming

In this section we will discuss how to model synchronous flow shop problems as mixed integer linear programs. At first, in 4.2.1 we will describe the basic model, examining all objective functions considered in this thesis. An alternative formulation for the special case of minimizing the makespan of synchronous flow shops with two dominating machines will be presented in Section 4.2.2. Finally, extensions discussed in Section 2.4 are modeled in Section 4.2.3.

4.2.1 The basic model

Karabati and Sayin (2003) presented a MILP model to minimize the makespan in synchronous flow shops. We will recite this model in the following and show how the model can be extended to allow the handling of further objective functions. For each of the first n cycles $t = 1, \ldots, n$ we define a binary variable $x_{jt} \in \{0, 1\}$ which is set to 1 iff job j starts its processing on machine M_1 in cycle t. Further, for all cycles $t = 1, \ldots, n+m-1$ variables $c_t \in \mathbb{R}_0^+$ denotes their lengths. Because all processing times are assumed to be integer and the sum of cycle times is to be minimized, the variable c_t does not need to be defined as integer. The following MILP minimizes the makespan of a synchronous flow shop by minimizing the sum of all cycle times:

$$\min \quad \sum_{t=1}^{n+m-1} c_t \tag{4.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{j\tau} p_{(t-\tau+1),j} \leq c_t \qquad t = 1, \ldots, n+m-1;$$

$$\tau = \max(t-m+1, 1), \ldots, \min(n, t) \tag{4.2}$$

$$\sum_{t=1}^{n} x_{jt} = 1 \qquad j = 1, \ldots, n \tag{4.3}$$

$$\sum_{j=1}^{n} x_{jt} = 1 \qquad t = 1, \ldots, n \tag{4.4}$$

$$c_t \in \mathbb{R}_0^+ \qquad t = 1, \ldots, n+m-1 \tag{4.5}$$

$$x_{jt} \in \{0, 1\} \quad j, t = 1, \ldots, n \tag{4.6}$$

In constraint (4.2) the cycle time of each cycle is determined by the maximum of the processing times within this cycle. Constraints (4.3) and (4.4) ensure that each job starts processing on machine M_1 in exactly one of the first n cycles and that in each of those cycles exactly one job starts. Finally, constraints (4.5) and (4.6) define the domains of the respective variables.

Further objective functions

To model further objective functions it is not sufficient to only have information about the length of the individual cycles. Additional auxiliary variables are needed which describe the individual completion times of the jobs. For each job j, we introduce an auxiliary variable $C_j \in \mathbb{R}_0^+$ indicating its completion time. Again, because of the integrality of all processing times, we do not need to require the domain of the variable to be integer. As each job is completed $m-1$ cycles after its start, we use the following constraint to determine the completion time:

$$\mathrm{C}_j \geq \sum_{\tau=1}^{t+m-1} c_\tau - (1 - x_{jt}) \cdot M \quad j = 1, \ldots, n; t = 1, \ldots, n \tag{4.7}$$

Here, M is a large constant to ensure that the term on the right is at most zero if job j does not start in cycle t. The constant can be bounded by the maximum makespan that can be realized by a feasible left-aligned schedule. Because this can not be calculated easily, we can use $\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}$ as a value for M as this is clearly an upper bound for the makespan.

To model lateness we will use auxiliary variables $L_j \in \mathbb{R}$, corresponding to the lateness of all jobs $j = 1, \ldots, n$:

$$L_j \geq C_j - d_j \quad j = 1, \ldots, n \tag{4.8}$$

The maximum lateness L of an instance can then be bounded by

$$L \geq L_j \quad j = 1, \ldots, n. \tag{4.9}$$

To minimize the number of late jobs, a binary variable $U_j \in \{0, 1\}$ is introduced for each job:

$$M \cdot U_j \geq L_j \quad j = 1, \ldots, n \tag{4.10}$$

Again, a large constant M is used in constraint (4.10) which sets the auxiliary variable U_j to 1 iff the respective job is late. In this case, M can be bounded by the maximum lateness that can be realized by a feasible left-aligned schedule. Again, this can not be calculated easily in general. Thus, it can be set to $\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} - \min_{j=1,\ldots,n} d_j$ which is clearly an upper bound.

With these auxiliary variables we can extend the MILP formulation (4.2)-(4.6) to model the problems for different objective functions: The objective (4.1) needs to be changed to the desired function (e.g. $\min L$; $\min \sum_j C_j$; $\min \sum_j U_j$) and the respective constraints and variables have to be added.

4.2.2 Makespan minimization for two dominating machines

We shortly repeat some of the results described in Section 2.3.2 for the presence of two dominating machines. In this case, minimizing the makespan can be decomposed into minimizing the sum of the lengths of the subsequences. Similar to the algorithm of Gilmore and Gomory discussed in Section 4.1, in which the synchronous flow shop was modeled as a traveling salesman problem, the situation with multiple subsequences can be modeled as a vehicle routing problem. Therein, each job is represented by a node and the subsequences of the synchronous flow shop are modeled as tours that start and end in an auxiliary node, analogously to the zero job in the algorithm of Gilmore and Gomory. The distance of two nodes in the vehicle routing problem is again defined by the cycle time that occurs when the corresponding jobs are processed on the two dominating machines. Kampmeyer (2015) evaluated multiple approaches to model this situation as a vehicle routing problem. In the following, we present the formulation with the best performance in regard of computational time. Let k_1, k_2 be the indices of the two dominating machines and $\kappa := k_2 - k_1$. Introduce the following binary variables $x_{ij}, s_j, e_j \in \{0, 1\}$ for $i, j = 1, \ldots, n$ with $i \neq j$:

$$x_{ij} = \begin{cases} 1, & \text{if job } i \text{ is processed on } M_{k_2} \text{ while job } j \text{ is processed on } M_{k_1} \\ 0, & \text{otherwise.} \end{cases}$$

$$s_j = \begin{cases} 1, & \text{if job } j \text{ is processed on } M_{k_1} \text{ while no job is processed on } M_{k_2} \\ & \text{i.e. job } j \text{ is the first job of a subsequence} \\ 0, & \text{otherwise.} \end{cases}$$

$$e_j = \begin{cases} 1, & \text{if job } j \text{ is processed on } M_{k_2} \text{ while no job is processed on } M_{k_1} \\ & \text{i.e. job } j \text{ is the last job of a subsequence} \\ 0, & \text{otherwise.} \end{cases}$$

Further, we define continuous variables $u_j \in \left[1, \lceil \frac{n}{\kappa} \rceil\right]$ for $j = 1, \ldots, n$, where u_j models the position of j within its subsequence. These variables were introduced by Miller et al. (1960) as subtour elimination constraints in traveling salesman problems. Here, they are used to ensure that there are no subsequences of size larger than $\lceil \frac{n}{\kappa} \rceil$ or smaller than $\lfloor \frac{n}{\kappa} \rfloor$. Then, the following MILP models the problem as the search for κ subsequences of jobs. The costs are defined by $c_{ij} = \max(p_{k_2 i}, p_{k_1 j})$ with $c_{0j} = p_{k_1 j}$ and $c_{j0} = p_{k_2 j}$.

$$\min \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_{ij} + \sum_{j=1}^{n} c_{0j} s_j + \sum_{j=1}^{n} c_{j0} e_j \tag{4.11}$$

$$\text{s.t.} \quad s_j + \sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n \tag{4.12}$$

$$e_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{4.13}$$

$$s_j + e_j \leq 1 \qquad j = 1, \ldots, n \tag{4.14}$$

$$\sum_{j=1}^{n} s_j = \kappa \tag{4.15}$$

$$\sum_{j=1}^{n} e_j = \kappa \tag{4.16}$$

$$u_j - u_i + \left\lceil \frac{n}{\kappa} \right\rceil (1 - x_{ij}) \geq 1 \qquad i, j = 1, \ldots, n, i \neq j \tag{4.17}$$

$$u_j \geq \left\lfloor \frac{n}{\kappa} \right\rfloor e_j \qquad j = 1, \ldots, n \tag{4.18}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n, i \neq j \tag{4.19}$$

$$s_i, e_i \in \{0, 1\} \qquad i = 1, \ldots, n \tag{4.20}$$

$$u_i \in \left[1, \left\lceil \frac{n}{\kappa} \right\rceil\right] \qquad i = 1, \ldots, n \tag{4.21}$$

Constraint (4.12) assures that each job is either the first of a subsequence or is preceded by exactly one job in its subsequence. Analogously, each job is either the last of a subsequence or succeeded by exactly one job as stated by constraint (4.13). Further, no job can be both the first and the last job of a subsequence (4.14). Constraints (4.15) and (4.16) force the number of first and last jobs of the subsequences to be exactly κ. Finally, (4.17) and (4.18) are the subtour elimination constraints based on Miller et al. (4.17) ensures that no subsequence contains more than $\left\lceil \frac{n}{\kappa} \right\rceil$ jobs while (4.18) ensures that each subsequence contains at least $\left\lfloor \frac{n}{\kappa} \right\rfloor$ jobs. Constraint (4.18) can be removed when $n \bmod \kappa = 0$ as in this case (4.17) already ensures that all subsequences contain exactly $\frac{n}{\kappa}$ jobs. Note that constraints (4.14) do not hold true if $n < 2\kappa$, because in this case there exist subsequences containing less than two jobs. In this case we need to drop this constraint. However, in this case the problem can be easily solved in polynomial time without the need of formulating it as a MILP. If $\kappa \geq n$, each subsequence consists of at most one job and the makespan of all feasible (left-shifted) schedules is the same. If $\frac{1}{2}n < \kappa \leq n$ the problem can be solved as a non-bipartite matching problem: For each job create a node and define the cost between two nodes i, j as the minimum of costs $c_{0i} + c_{ij} + c_{j0}$ and $c_{0j} + c_{ji} + c_{i0}$ as defined above. This represents the cost that occurs when a subsequence consists of only these two jobs. Further, add $2\kappa - n$ auxiliary nodes and define the cost of each auxiliary node to an actual node j as $c_{0j} + c_{j0}$, representing the cost that occurs when a subsequence consists only of j as single job. The cost between two auxiliary nodes is set to a large constant $M > \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}$. Then, a minimum cost perfect matching leads to an optimal solution of this instance. This can be achieved in polynomial time, e.g. by the algorithm of Edmonds (1965).

4.2.3 Extensions

In addition to the basic mixed integer program formulations we will shortly describe how to model the extensions discussed in Section 2.4.

Idle Jobs

There are two possible ways to deal with idle jobs in the MILP formulation of the synchronous flow shop problem. In both, an upper bound $\bar{n}$ on the number of potentially used idle jobs has to be specified. One possibility is to modify a given instance of the synchronous flow shop problem with n jobs to include idle jobs $j' = n+1, n+2, \ldots, n+\bar{n}$ and to solve the MILP for this new instance. Alternatively, the integer programming formulation can be altered to allow for $n + \bar{n} + m - 1$ cycles and changing constraint (4.3) into

$$\sum_{j=1}^{n} x_{jt} \leq 1 \text{ for } t = 1, \ldots, n + \bar{n} \tag{4.22}$$

such that for each of the first $n + \bar{n}$ positions, at most one job has to start. Further, in all other constraints and the objective function, the parameter n has to be replaced by $n + \bar{n}$. As discussed in Section 2.4.1, an upper bound on the number of idle jobs can be given by $\bar{n} \leq (n-1)(m-1)$.

Job Splitting

If splitting of jobs is allowed, the processing times of jobs on the machines are no longer fixed and need to be calculated. To cope with this situation, we can introduce new variables $y_{jt} \in \mathbb{R}$ which denote the processing time for each job $j = 1, \ldots, n$ in cycle $t = 1, \ldots, n+m-1$. For a job j starting in cycle t, its actual processing time $\tilde{p}_{ij}$ is then equal to $y_{j,t+i-1}$. The calculation of cycle times can be changed by replacing constraint (4.2) by

$$y_{jt} \leq c_t \text{ for } j = 1, \ldots, n\ , t = 1, \ldots, n+m-1. \tag{4.23}$$

For all combined processing times $p_{[i_1,i_2],j}$ that have to be split across machines $M_{i_1}, \ldots, M_{i_2}$ we introduce a constraint

$$\sum_{l=i_1-1}^{i_2-1} y_{j,t+l} = p_{[i_1,i_2],j} x_{jt} \text{ for } j, t = 1, \ldots, n. \tag{4.24}$$

to assure that job j is processed for the required time in cycles $t + i_1 - 1, \ldots, t + i_2 - 1$ iff job j starts its processing on machine M_1 in cycle t. Mandatory processing times for all jobs can be included in the same way by setting

$$y_{j,t+i-1} \geq \underline{p}_{ij} x_{jt} \text{ for } j, t = 1, \ldots, n\ , i = 1, \ldots, m \tag{4.25}$$

and

$$y_{j,t+i-1} \leq \overline{p}_{ij} x_{jt} \text{ for } j, t = 1, \ldots, n\ , i = 1, \ldots, m. \tag{4.26}$$

It should be noted that in the case without upper limits on the actual processing times, it is possible to formulate a mixed integer program without the auxiliary variables y_{jt}. However, constraints need to be added to assure that the cycle times c_t of all cycles $t = 1, \ldots, n+m-1$ allow for a feasible actual processing time for each job. Then, in a post-processing step, we can determine actual processing times for each job such that the makespan does not increase. To achieve this, we replace constraints (4.23) and (4.24) for all combined processing times $p_{[i_1,i_2],j}$ by

$$\sum_{j=1}^{n} p_{[i_1,i_2],j} x_{j,t} \leq \sum_{l=i_1-1}^{i_2-1} c_{t+l} \text{ for } t = 1, \ldots, n. \tag{4.27}$$

This constraint assures that the cycle times of cycles $t + i_1 - 1, \ldots, t + i_2 - 1$ are large enough such that the combined processing time $p_{[i_1,i_2],j}$ of the job starting in cycle t can

be distributed over these cycles. Further, we drop constraint (4.26) and replace constraint (4.25) by

$$c_{t+i-1} \geq \underline{p}_{ij} x_{jt} \text{ for } j, t = 1, \ldots, n\ , i = 1, \ldots, m. \tag{4.28}$$

Constraints (4.27) and (4.28) assure that in a post-processing step, the actual processing times can be calculated by Algorithm 4.2. For a job j with combined processing time $p_{[i_1,i_2],j}$ starting in cycle t its actual processing times on machines $M_{i_1}, \ldots, M_{i_2}$ are calculated iteratively in lines 5-7. The assigned actual processing time may not lead to infeasible processing. Thus, it may not exceed the sum of the remaining lower limits on the actual processing time required in the following cycles. Therefore, in line 6 the actual processing time is calculated as the minimum of this permitted time and the cycle time as determined by the MILP. Because of constraints (4.27) and (4.28), the actual processing times can be feasibly assigned in this way.

Input: sequence of jobs $\sigma = (\sigma_1, \ldots, \sigma_n)$
Input: cycle times c_t, combined processing times $p_{[i_1,i_2],j}$
Output: actual processing times $\tilde{p}_{ij}$ for $i = i_1, \ldots, i_2; j = 1, \ldots, n$

```
1 for t = 1, ..., n do
2     j ← σ_t
3     p ← p_{[i1,i2],j}
4     for l = i1, ..., i2 do
5         p' ← p − Σ_{k=l+1}^{i2} p_{kj}
6         p̃_{lj} ← min(p', c_{t+l−1})
7         p ← p − p̃_{lj}
8     end
9 end
```

Algorithm 4.2: Post-processing step to find actual processing times given a sequence and cycle times in the case of job splitting without upper limits.

Unfortunately, if upper limits on the actual processing time are defined, it is not possible to remove the variables resembling the actual processing times of the individual jobs from the MILP formulation (cf. Bultmann (2015)).

Resources

First we consider the case where each operation O_{ij} may need additional resources. Let R be the set of resources and for each resource $r \in R$ let $\bar{r}$ be the quantity in which resource r is available. Further, for $r \in R; j = 1, \ldots, n$ and $i = 1, \ldots, m$ let the parameter $\xi_{ijr} \in \mathbb{N}$ be set to the quantity of resource r that is required by operation O_{ij}. Then, adding the constraint

$$\sum_{\tau=\max(t-m+1,1)}^{\min(n,t)} \sum_{j=1}^{n} x_{j\tau}\xi_{(t-\tau+1),j,r} \leq \bar{r} \text{ for } r \in R\ ,\ t = 1, \ldots, n+m+1 \tag{4.29}$$

ensures that each resource r is only used at most $\bar{r}$ times in each cycle, allowing for a feasible resource assignment.

To incorporate resources that are needed over the whole course a job is processed on machines M_1 to M_m, we define parameter $\xi_{jr} \in \mathbb{N}$ which is set to the quantity of resource r required by job j. Again, adding the constraint

$$\sum_{\tau=\max(t-m+1,1)}^{\min(n,t)} \sum_{j=1}^{n} x_{j,(t-\tau+1)}\xi_{j,r} \leq \bar{r} \text{ for } r \in R\ ,\ t = 1, \ldots, n+m+1 \tag{4.30}$$

ensures that each resource r is used at most $\bar{r}$ times in each cycle.

Circular production and changeover times

When we consider circular production, we can introduce auxiliary variables $z_{jh} \in \{0, 1\}$ that indicate whether a job h is scheduled exactly m positions after job j. In this case, a changeover might be necessary between these two jobs. We link the variables z_{jh} to the variables x_{jt} via the constraint

$$x_{j,t-m} + x_{ht} - z_{jh} \leq 1 \text{ for } j, h = 1, \ldots, n; t = m+1, \ldots, n \tag{4.31}$$

and adjust the objective function by adding the changeover times, resulting in

$$\min \sum_{t=1}^{n+m+1} c_t + \sum_{j=1}^{n}\sum_{h=1}^{n} s_{f(j)f(h)}z_{jh} \tag{4.32}$$

where $f(j)$ is the job family of job j.

Order scheduling

As discussed in Section 2.4, considering order scheduling is only relevant for objective functions that depend on individual completion times. To incorporate order scheduling into the MILPs discussed in Section 4.2.1, we can define for each order Θ_i the completion time C_{Θ_i} by the constraint

$$C_{\Theta_i} \geq C_j \text{ for } j \in \Theta_i. \tag{4.33}$$

All other auxiliary variables indicating the lateness of orders and the total number of late orders can then be defined accordingly using the completion time of an order.

4.3 Branch and bound

A branch and bound algorithm represents an exact method to solve optimization problems by searching the complete solution space for an optimal solution. It employs a divide and conquer strategy by dividing the solution space into smaller subspaces (branching) with the intent of finding a good evaluation (i.e. a lower bound) of the quality of the solutions within each subspace. Whenever it can be shown that it is not possible for any solution within a subspace to be strictly better than the best solution found so far, the whole subspace can be disregarded and the search is intensified in the solution space where better solutions are still possible.

The quality of a branch and bound algorithm is predominantly determined by three attributes: The branching strategy, i.e. how the solution space is divided, the quality of the evaluation function which calculates the lower bound on the objective value of the divided subspaces, and the quality of the initial and later found solutions. In Section 4.4 we will discuss lower bounds for various objective functions for the synchronous flow shop which can be used within the branch and bound framework. The most common branching strategy for permutation problems consists of fixing the start of a permutation and branching over all possible extensions. Initial solutions can be obtained by applying heuristics as described in Chapter 5. Algorithm 4.3 shows the outline of a general branch and bound algorithm for synchronous flow shop problems. The algorithm is fed a set of jobs, a (possibly empty) fixed starting sequence of jobs and an upper bound on the objective value which is given by the best solution value found so far. In line 4 of the algorithm, the solution space is divided into subspaces which are identified by which of the jobs is fixed next in the sequence. For each of the resulting sequences, a lower bound is calculated in line 6. For this, depending on the objective functions, one of the algorithms described in Section 4.4 is used. Only if the lower bound is smaller than the hitherto best found solution value, the search is intensified in this subspace of the solution space by recursively executing the branch and bound algorithm within this subspace in line 8. Otherwise, no sequence which starts with this subsequence can achieve a better objective value than the best hitherto found solution. Therefore, this subspace of the solution space

is disregarded. Afterwards, if a better solution is found within this subspace, the upper bound is updated in line 10.

```
    Input: set of jobs J, objective function F
    Input: fixed sequence of jobs σ = (σ_1, ..., σ_l), upper bound UB
    Output: optimal value for F
 1  if all jobs fixed then
 2  |  return F(σ);
 3  end
 4  for j ∉ σ do
 5  |  σ' ← append j to σ
 6  |  LB ← compute lower bound for J, F and σ'
 7  |  if LB < UB then
 8  |  |  c ← recursively call B&B (Algorithm 4.3) for J, F, UB and σ'
 9  |  |  if c < UB then
10  |  |  |  UB ← c
11  |  |  end
12  |  end
13  end
```

Algorithm 4.3: Outline of a general branch and bound algorithm for synchronous flow shop problems

4.4 Lower Bounds

In this section, we will investigate lower bounds for synchronous flow shop problems. These can be used in branching algorithms (cf. Section 4.3) or to evaluate heuristic approaches that will be discussed in Chapter 5. We will present lower bounds for synchronous flow shop problems and the objective functions of minimizing the makespan, maximum lateness and total completion time. In all of these cases, there exist polynomially solvable special cases which we will exploit to find the respective lower bounds. In the following, we will concentrate on the basic synchronous flow shop model without any extensions. For each objective function we discuss how to get an initial lower bound for a given instance as well as how to obtain lower bounds if we are already given a fixed starting sequence of jobs which can be used in a branch and bound algorithm as described in Section 4.3.

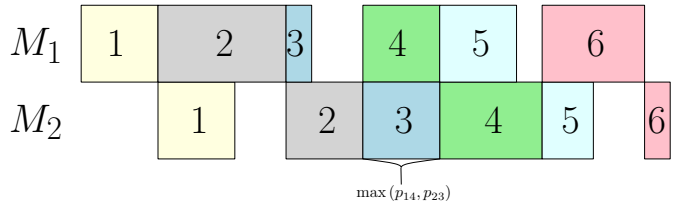
4.4.1 Makespan

In the following, we will discuss several lower bounds for the makespan of synchronous flow shop problems. For this, we will initially consider special cases with two dominating machines. Afterwards, we will take a look at the general case without any machine dominance.

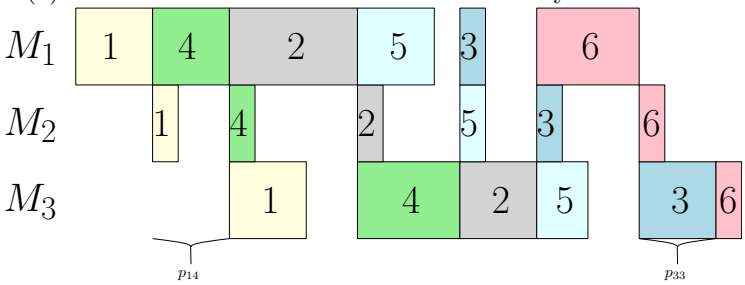
Two dominating machines

First, we will discuss problems of type $F|\text{synmv}, \text{dom}\,(k_1, k_2)\,, p_{ij}^{ndom} = 0|C_{\max}$ with two dominating machines $1 \leq k_1 < k_2 \leq m$. If $k_2 = k_1 + 1$, the problem can be solved optimally in polynomial time using the algorithm of Gilmore and Gomory. Thus, in the following we may assume that $k_2 > k_1 + 1$. In this case, we can exploit the observations we made in Section 2.1.2 that any schedule can be decomposed into $\kappa := k_2 - k_1$ subsequences and that the sum of the subsequences' makespans is equal to the makespan of the schedule. Theorem 4.1 proves that the optimal makespan of a synchronous flow shop with two adjacent dominating machines can be used as a lower bound for instances in which the two dominating machines are not adjacent. The lower bound is achieved by showing that the makespan of this instance is not larger than the sum of the makespans of the κ subsequences for the case that the two dominating machines are not adjacent. We denote this bound as the Gilmore-Gomory lower bound.

In the following, to improve the readability of the proofs we assume that $n \bmod \kappa = 0$. However, all results can be generalized for arbitrary values of $n > \kappa$.



(a) Schedule achieved for the Gilmore-Gomory lower bound



(b) Corresponding schedule for the initial instance with two non-adjacent dominating machines

Figure 4.4: Approximation of the Gilmore-Gomory lower bound for two dominating machines in the case of $F3|\text{synmv}, \text{dom}\,(1,3)\,|C_{\max}$.

Theorem 4.1. *Let I be an instance of $F|synmv, dom\,(k_1, k_2)\,, p_{ij}^{ndom} = 0|C_{\max}$ with jobs J and optimal makespan C. Create an instance I' of $F2|synmv|C_{\max}$ with jobs J' and $p_{1j'} = p_{k_1,j}; p_{2j'} = p_{k_2,j}$ for all $j \in J$. Then, for the optimal makespan C' of instance I', it holds that $C' \leq C$.*

Proof. As discussed in Section 2.3.2, the schedule of a flow shop with two non-adjacent dominant machines k_1, k_2 decomposes into κ independent sub-schedules of synchronous flow shops with two adjacent dominating machines and the makespan of the schedule

equals the sum of the makespans of these sub-schedules. Thus, for an optimal schedule σ of I with independent sub-schedules $\sigma^1, \ldots, \sigma^\kappa$, the concatenations of these sub-schedules results in a schedule σ' for instance I' with

$$\sigma' = \left((\sigma_1^1)', \ldots, (\sigma_{n/\kappa}^1)', (\sigma_1^2)', \ldots, (\sigma_{n/\kappa}^2)', \ldots, (\sigma_1^\kappa)', \ldots, (\sigma_{n/\kappa}^\kappa)'\right)$$

which has a makespan of at most C. Because an optimal makespan C' of I' is at most as large as this makespan, it holds that $C' \leq C$. □

Further, we even can show how well this lower bound performs in comparison to an optimal solution:

Theorem 4.2. *Let I be an instance of $F|synmv, dom\,(k_1, k_2)\,, p_{ij}^{ndom} = 0|C_{\max}$ with optimal makespan C^* and let C_{GG} be the Gilmore-Gomory lower bound. Then, $C_{GG} \geq C^* - (\kappa - 1) \cdot (\max_{i,j} p_{ij})$.*

Proof. Let $\sigma' = (\sigma'_1, \ldots, \sigma'_n)$ be a sequence of jobs for an instance I' of $F2|\text{synmv}|C_{\max}$ with jobs J' and $p_{1j'} = p_{k_1,j}; p_{2j'} = p_{k_2,j}$ for all $j \in J$ which achieves the Gilmore-Gomory lower bound C_{GG}. Construct a sequence σ for the original instance I as follows. For $\lambda = 1, \ldots, \kappa$, let σ^λ be the independent subsequences of σ. Set

$$\sigma_i^\lambda = \sigma'_{(\lambda-1)\frac{n}{\kappa}+i}; i = 1, \ldots, \frac{n}{\kappa} - 1$$

which resembles the splitting of the sequence σ' into κ subsequences. In the sequence σ, the operations on machine M_{k_1} of the first κ jobs and the operations on machine M_{k_2} of the last κ jobs are the only jobs on a dominating machine in their respective cycles while in the sequence σ' of instance I', the operation on machine M_{k_1} of the t-th job of σ falls together with the operation of the t-last jobs of σ for $t = 2, \ldots, \kappa$. An example is depicted in Figure 4.4 for the case of $F3|\text{synmv}, \text{dom}\,(1,3)\,|C_{\max}$ where the processing times of the second and second-last jobs fall together in the two-machine sequence. Thus, the makespan $C_{\max}$ of σ can be calculated by

$$\begin{aligned} C_{\max}(\sigma) &= C_{GG} + \sum_{t=2}^{\kappa} p_{k_1,\sigma_t} + \sum_{t=n+m-(2+\kappa)}^{n+m-2} p_{k_2,\sigma_t} - \sum_{t=2}^{\kappa} \max\left(p_{k_1,\sigma_t}, p_{k_2,\sigma_{n+m-t}}\right) \\ &= C_{GG} + \sum_{t=2}^{\kappa} \min\left(p_{k_1,\sigma_t}, p_{k_2,\sigma_{n+m-t}}\right) \\ &\leq C_{GG} + (\kappa - 1) \cdot \left(\max_{i,j} p_{ij}\right). \end{aligned}$$

Because an optimal makespan C^* of the instance I is not larger than $C_{\max}(\sigma)$, the claimed approximation bound of the Gilmore-Gomory lower bound is valid. □

Given a fixed sequence of jobs $\sigma = (\sigma_1, \ldots, \sigma_l)$, we can derive a lower bound on all possible schedules that contain σ as a starting sequence by using Algorithm 4.4. Because

the machines M_{k_1} and M_{k_2} are dominating, the cycle times c_t of all cycles $t = 1, \ldots, l+k_1-1$ will not be increased regardless of the sequence of jobs to be appended. Thus, the makespan of the complete schedule is at least as large as the sum of the first $l + k_1 - 1$ cycle times (cf. line 1). Figure 4.5 depicts this situation for five machines where machines M_2 and M_4 are dominating. Only cycle times of cycles $l + 2, \ldots, n + m - 1$ will be affected by the remaining jobs.

As shown in Theorem 4.1, the makespan of the remaining jobs not scheduled within the starting sequence is at least as large as the makespan of the relaxation into a two-machine synchronous flow shop. The processing times of the operations on the second dominating machine in those cycles in which no operation is scheduled on the first dominating machine can be incorporated as well. The procedure is similar to the method described in the end of Section 4.1 where a starting sequence was given for scheduling a two-machine problem. There, the algorithm of Gilmore and Gomory was altered in such a way that the job 0 had processing time of zero on the first machine and processing time equal to the processing time of the last job of the starting sequence on the second machine. For non-adjacent dominating machines k_1, k_2, the processing times of jobs $j = \sigma_{l-\kappa+1}, \ldots, \sigma_l$ can be included by introducing jobs with processing times of zero on the first as well as processing times of $p_{k_2 j}$ on the second machine (cf. line 9).

Then, the sum of the first $l+k_1-1$ cycle times plus the optimal makespan of the relaxed instance serves as a lower bound on all schedules starting with the starting sequence σ. Theorem 4.3 proves the correctness of Algorithm 4.4.

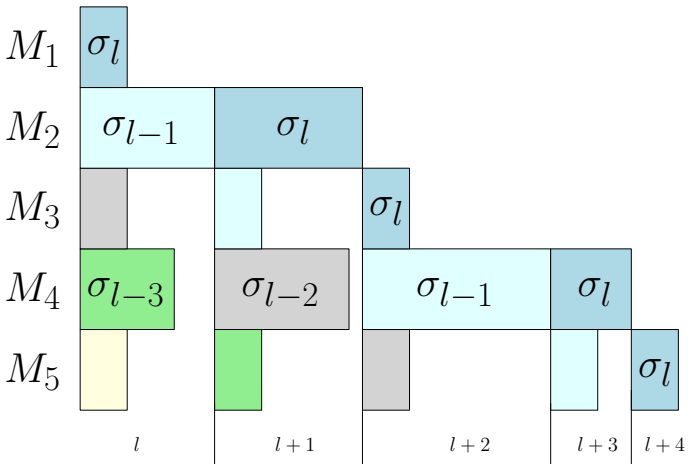


Figure 4.5: Starting sequence of a schedule for a synchronous flow shop with two non-adjacent dominating machines.

Theorem 4.3. *Let I be an instance of $F|synmv, dom\,(k_1, k_2)\,, p_{ij}^{ndom} = 0|C_{\max}$ and let $\sigma = (\sigma_1, \ldots, \sigma_l)$ be a fixed starting sequence. Then, Algorithm 4.4 finds a lower bound on the optimal makespan in runtime $O\,(n \log n)$.*

Proof. Algorithm 4.4 consists of a single execution of the algorithm by Gilmore and Gomory on the transformed instance, leading to the stated runtime. Because the non fixed jobs $j \notin \sigma$ do not affect the cycle times of the first $l + k_1 - 1$ cycles we only need to show that M is a lower bound on the remaining sum of cycle times to prove the correctness of the lower bound in line 12. This, however, is a direct consequence of Theorem 4.1. Thus, the lower bound is valid. □

Input: set of jobs J, two dominating machines k_1, k_2.
Input: fixed sequence of jobs $\sigma = (\sigma_1, \ldots, \sigma_l)$.
Input: cycle times c_t for $t = 1, \ldots, l + m - 1$
Output: lower bound on $C_{\max}$
1 $LB_0 \leftarrow \sum_{t=1}^{l+k_1-1} c_t$
2 $J' \leftarrow \emptyset$
3 **for** $j \in J$ **do**
4 | **if** $j \notin \sigma$ **then**
5 | | Create a job j' with $p_{1j'} = p_{k_1,j}, p_{2j'} = p_{k_2,j}$ and add it to J'
6 | **end**
7 **end**
8 **for** $i = 0, \ldots, \min\{l, \kappa\} - 1$ **do**
9 | Create a job j' with $p_{1j'} = 0, p_{2j'} = p_{k_2,\sigma_{l-i}}$ and add it to J'
10 **end**
11 $M \leftarrow$ optimal makespan for jobs J' by Algorithm 4.1
12 $LB \leftarrow LB_0 + M$
13 **return** LB

Algorithm 4.4: Gilmore-Gomory lower bound (LB-GG) for a fixed starting sequence in the case of two dominating machines.

The general case

One possibility to obtain a lower bound on the optimal makespan of an instance of the general synchronous flow shop problem is to relax the instance by transforming it into a problem consisting of only two dominating machines. For an instance I of the synchronous flow shop problem $F|\text{synmv}|C_{\max}$ with jobs J consider the following relaxation: For $1 \leq k_1 < k_2 \leq m$ construct an instance I_{k_1,k_2} with the same jobs J, but set all processing times on machines M_k with $k \notin \{k_1, k_2\}$ to zero. Clearly, this relaxed instance is an instance of a synchronous flow shop with two dominating machines and the optimal makespan of this relaxed instance is a lower bound on the makespan of the original instance I. If $k_2 = k_1+1$, the two dominating machines are adjacent and the makespan of the instance can be determined by the algorithm of Gilmore and Gomory. Otherwise, if the two dominating machines are not adjacent, we can use the result of Theorem 4.1. Then, the maximum of the makespans of all instances I_{k_1,k_2} for all $1 \leq k_1 < k_2 \leq m$ results in a lower bound for the original instance.

Given a fixed starting sequence of jobs σ, we can use Algorithm 4.5 which consists of calculating the makespan for all relaxed instances I_{k_1,k_2} for all $1 \leq k_1 < k_2 \leq m$ for the unscheduled jobs. The largest of these calculated makespans yields a lower bound for the makespan of the remaining jobs and thus can be used to derive a lower bound on the makespan of the original instance I with the fixed starting sequence. We denote this lower bound as the (iterative) Gilmore-Gomory lower bound.

Input: set of jobs J
Input: fixed sequence of jobs $\sigma = (\sigma_1, \ldots, \sigma_l)$
Input: cycle times c_t for $t = 1, \ldots, l + m - 1$
Output: lower bound on $C_{\max}$

$LB \leftarrow \sum_{t=1}^{l+m-1} c_t$
for $k_1 = 1, \ldots, m-1$ **do**
  $S \leftarrow \sum_{t=1}^{l+k_1-1} c_t$
  for $k_2 = k_1 + 1, \ldots, m$ **do**
    $L_{k_1k_2} \leftarrow$ optimal makespan for Jobs J, starting sequence σ and dominating machines k_1, k_2 as determined by Algorithm 4.1
    $LB \leftarrow \max(LB, L_{k_1k_2} + S)$
  end
end
return LB

Algorithm 4.5: Gilmore-Gomory lower bound (LB-GG) for a fixed starting sequence

Theorem 4.4. *Let I be an instance of $F|synmv|C_{\max}$ and let $\sigma = (\sigma_1, \ldots, \sigma_l)$ be a fixed starting sequence. Then, Algorithm 4.5 finds a lower bound on the optimal makespan in runtime $O\left(m^2 \cdot n \log n\right)$.*

Proof. Algorithm 4.5 consists of $O\left(m^2\right)$ executions of Algorithm 4.1, leading to the claimed runtime. Let $LB > \sum_{t=1}^{l+k_1-1} c_t$, because otherwise LB clearly is a lower bound. Then, $LB = L_{k_1k_2} + C_{k_1}$ for some indices k_1, k_2. Assume there exists a sequence σ' to instance I with $\sigma'_i = \sigma_i$ for $i = 1, \ldots, l$ and makespan $C_{\max}(\sigma') < LB$. Consider the cycle times of cycles t with $k_1 + l \leq t \leq n + m - 1$. It must hold that $\sum_{t=k+1}^{n+m-1} c_t(\sigma') < L_{k_1k_2}$ where $c_t(\sigma')$ is the cycle time of cycle t for sequence σ'. For all jobs $\sigma'_{l+1}, \ldots, \sigma'_n$ set the processing times to zero on machines M_i with $i \neq k_1, k_2$. Clearly, the sum of the cycle times does not increase. Further, L_{k_1,k_2} is a lower bound on the sum of cycle times because of Theorems 4.1 and 4.3. Therefore it is also a lower bound on the sum of cycle times for the original sequence with arbitrary processing times on all machines. This, however, is a contradiction to $C_{\max}(\sigma') < LB$. Thus, the lower bound holds. □

Another lower bound on the makespan can be achieved by relaxing the synchronous flow shop in the following way: For all jobs we allow that the operations of each job may

be processed in any order, do not have to be processed in consecutive cycles and may even be processed within the same cycle on different machines. Thus, the operations of the jobs may be distributed freely over all cycles with the only constraint that no machine may process more than one operation in the same cycle. Then, similar to the first stage of the algorithm by Gilmore and Gomory, sorting the processing times on each machine and matching the k-th smallest processing times together for all $k = 1, \ldots, n$ such that they are processed in the same cycle, leads to a lower bound on the makespan. This lower bound can be further improved by taking into account the structure of the synchronous flow shop. Only in cycles $m, \ldots, \max(m-1, n)$ there is a job being processed on every machine while in the first $m-1$ cycles and the last $m-1$ cycles not all machines are occupied. Therefore, to improve the lower bound we require that no operations may be processed in cycles $t = 1, \ldots, m-1$ on machines $M_{t+1}, \ldots, M_m$ and no operations may be processed in cycles $t = n+1, \ldots, n+m-1$ on Machines $M_1, \ldots, M_{t-n}$. Then, the lower bound can be described as matching together the largest processing times on the machines in the cycles when all machines are occupied and moving the smaller processing times to the front or back of the schedule.

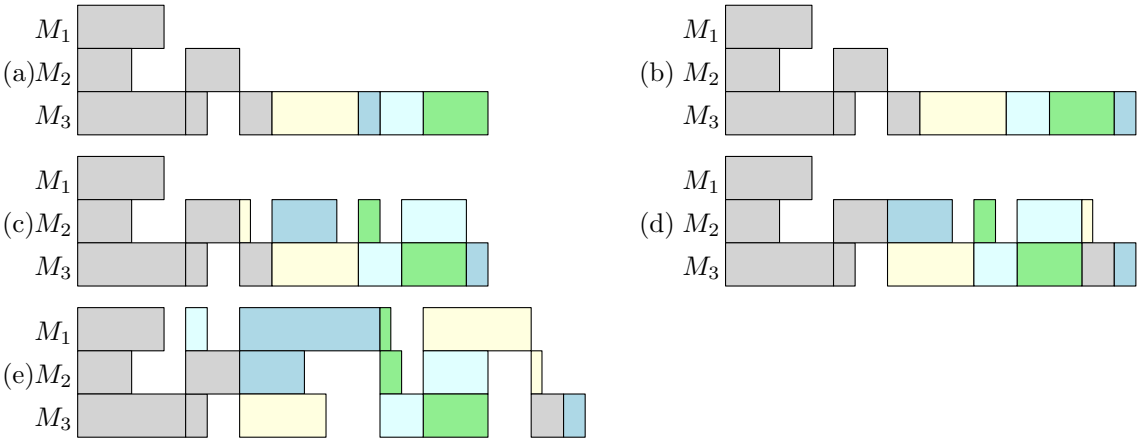


Figure 4.6: Exemplary application of the matching lower bound (LB-M) with a given starting sequence

In the following, we will describe the construction of the lower bound in more detail for the case when we are to find a lower bound for a schedule in the presence of a fixed starting sequence $\sigma = (\sigma_1, \ldots, \sigma_l)$. Naturally, this can also be applied to an empty starting sequence as well. The schedule is relaxed in the manner described above such that operations of jobs do not have to adhere to the flow shop constraints and operations of the same job can be processed on different machines in the same cycle. Starting with machine M_m, we iteratively distribute the operations for machine M_k over the cycles $l+m-k, \ldots, n+k-1$. In each iteration k we match the i-th smallest operation on machine M_{m-k+1} with the i-th smallest cycle time already determined by the previous iterations for machines $M_{m-k+2}, \ldots, M_m$. Afterwards, the operations that are processed in the cycle with the

smallest cycle time are moved to the back of the schedule, such that cycle $n+m-k$ consists of these operations and such that they are disregarded in future iterations. Figure 4.6 gives an exemplary application of the matching lower bound for three machines. In step (a) operations on M_3 are distributed arbitrarily. In step (b), the smallest operation on M_3 is moved to last cycle and this cycle will not be altered in future steps. In step (c), the operations on the second machine are distributed over the cycles such that the i-th smallest operation on machine M_2 coincides with the i-th smallest cycle time determined by the operations on machine M_3. Afterwards, the smallest cycle is moved to the back again in step (d). Finally, in the last iteration (step (e)) the remaining non-fixed operations on the first machine are distributed.

Input: set of jobs J, fixed sequence of jobs $\sigma = (\sigma_1, \ldots, \sigma_l)$
Input: cycle times c_t for $t = 1, \ldots, l+m-1$
Output: lower bound on $C_{\max}$
1 $\Gamma^0 \leftarrow$ list of size n with all entries set to 0
2 $LB \leftarrow 0$
3 **for** $k = 1, \ldots, m$ **do**
4 $\quad \mu^k \leftarrow$ list of operations $O_{(m-k+1),j}$ sorted non-increasingly in $p_{(m-k+1),j}$
5 $\quad \Gamma^k \leftarrow$ list of size n
6 $\quad$ **for** $i = 1, \ldots, n$ **do**
7 $\quad\quad \Gamma_i^k \leftarrow \max\left(p(\mu_i^k), \Gamma_i^{k-1}\right)$
8 $\quad$ **end**
9 $\quad LB \leftarrow LB + \Gamma_n^k$
10 $\quad \Gamma_n^k \leftarrow c_{l+m-k}$
11 $\quad$ Sort Γ^k in non-increasing order
12 **end**
13 **for** $i = 1, \ldots, n-1$ **do**
14 $\quad LB \leftarrow LB + \Gamma_i^m$
15 **end**
16 **return** LB

Algorithm 4.6: Matching lower bound (LB-M) on the makespan.

In the following, we describe this procedure more formally and prove the validity of Algorithm 4.6 to obtain a lower bound. For the k-th iteration let μ^k be the list of operations $O_{(m-k+1),j}$ of the jobs $j \notin \sigma$, sorted in non-increasing order of $p_{(m-k+1),j}$. Distribute the operations such that the operation μ_i^k with the i-th smallest processing time is processed in the cycle with the i-th smallest maximum processing time determined by the operations that were matched together in previous iterations. In Algorithm 4.6, the matching is performed in line 7. Therein, Γ^{k-1} is a list of the maxima of processing times of the hitherto fixed operations on machines $m-k+2, \ldots, m$ in cycles $l+k, \ldots, n+k-1$, again sorted in non-increasing order and $p(\mu_l^k)$ is the processing time of operation μ_l^k. The new cycle time Γ_i^k is obtained by matching the i-th largest cycle time for the hitherto fixed

operations Γ_i^{k-1} with the i-th largest processing time of an operation on M_{m-k+1}. Then, cycle $n+k-1$, represented by Γ_n^k is the one with the smallest cycle time. No further operation is added to this cycle in the remaining iterations. Thus, the lower bound is increased by the length of this cycle in line 9. In lines 10 and 11, the list Γ is prepared for the next iteration by replacing the cycle time of the cycle $n+k-1$ by the cycle time of cycle $l+m-k$ as determined by the starting sequence. After the m-th iteration, all operations are fixed and thus the lower bound can be calculated by adding together all not yet considered cycle times (cf. line 14). We refer to this lower bound as the matching lower bound (LB-M).

Theorem 4.5. *Let I be an instance of $F|synmv|C_{\max}$ and let $\sigma = (\sigma_1, \ldots, \sigma_l)$ be a fixed starting sequence. Then, Algorithm 4.6 finds a lower bound on the makespan in runtime $O(m \cdot n \log n)$.*

Proof. For each machine, the operations are sorted and distributed over the cycles, leading to the claimed runtime.

Let $\sigma' = (\sigma'_1, \ldots, \sigma'_n)$ be a sequence of jobs with $\sigma'_i = \sigma_i$ for $i = 1, \ldots, l$ with minimal makespan $C^* := C_{\max}(\sigma')$. In the following, we will relax the problem as described above. Starting with the distribution of operations given by σ' (which still adhere to the flow shop constraints) we iteratively rearrange the operations in a way that results in a distribution of operations identical to Algorithm 4.6 without increasing the makespan. In the following we will denote the processing time of an operation o by $p(o)$.

Consider the first iteration of the algorithm with $k = 1$ in line 4 in which operations on M_m are distributed. Let operation $o := \mu_n^1$ be processed in cycle i and consider operation o' that is processed on machine M_m in cycle $n+m-1$. Let $\rho_{i1}, \ldots, \rho_{i,m-1}$ be the operations that are currently processed in cycle i on machines $M_1, \ldots, M_{m-1}$. Then, we can swap operations o and o', not increasing the makespan:

$$C' = C^* - \max\left(p(o), \max_{k=1}^{m-1}(p(\rho_{ik}))\right) - p(o') + \max\left(p(p), \max_{k=1}^{m-1}(p(\rho_{ik}))\right) + p(o) \leq C^*$$

Then, the last cycle time is equal to $p(o) = p\left(\mu_n^1\right) = \Gamma_n^1$, resembling the cycle time of Algorithm 4.6 in cycle $n+m-1$.

Now, consider iteration k with $k > 1$ where the operations on machine M_{m-k+1} are distributed. Let C be the makespan of the schedule with the current distribution of operations. For $i = 1, \ldots, n$ let Γ_i^{k-1} be the maximum of the processing time of the operations on machines $m-k+2, \ldots, m$ in the cycles $l+m-k+1, \ldots, n+m-k$, i.e.

$$\Gamma_i^{k-1} := \max_{h=m-k+2}^{m} \rho_{l+m-k+i,h}.$$

Sort the cycles such that $\Gamma_h^{k-1} \geq \Gamma_i^{k-1}$ for $h \leq i$. Again, denote with ρ_{ik} the operation that is processed in cycle i on machine k in the schedule with the current distribution of operations. Let $l+m-k+s, l+m-k+t$ be two cycles with $\Gamma_s^{k-1} \geq \Gamma_t^{k-1}$ but $p(\rho_{l+m-k+s,m-k+1}) < p(\rho_{l+m-k+t,m-k+1})$, i.e. where the maximum of the processing times of operations on machines $m-k+2, \ldots, m$ in cycle $l+m-k+s$ is at least as large as

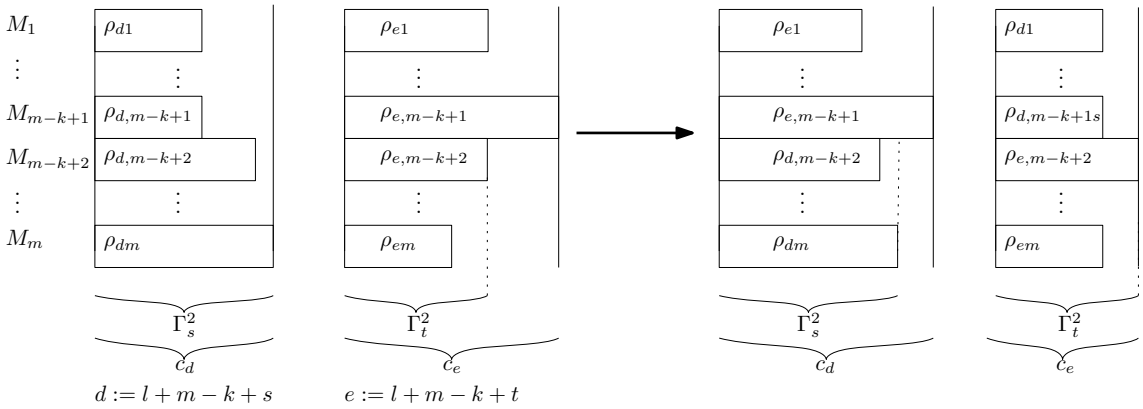


Figure 4.7: Exemplary exchange for iteration k. To improve readability, we set $d := l+m-k+s$ and $e := l+m-k+t$. Because $p(\rho_{d,m-k+1}) < p(\rho_{e,m-k+1})$ while $\Gamma_s^2 \geq \Gamma_t^2$, the operations on machines $M_1, \ldots, M_{m-k+1}$ are swapped, resulting in no increase in the makespan.

the processing times of the operations on these machines in cycle $l + m - k + t$ but the processing time of the operation on machine $m-k+1$ is smaller in cycle $l+m-k+s$ than in cycle $l + m - k + t$. In this case, for these two cycles swap the operations on machines $1, \ldots, m-k+1$ such that for each machine the processing time of the operation processed in cycle $l + m - k + s$ is at least as large as the processing time on this machine in cycle $l+m-k+t$. Obviously, the makespan does not increase. Figure 4.7 depicts an exemplary configuration and swap. After the swaps, Γ^k is the maximum of processing times of the processes on machines $m - k + 1, \ldots, m$ for these cycles set like in line 7 of the algorithm. The cycle time of cycle $n + m - k$ is then equal to Γ_n^k and the cycle will not be affected by future iterations.

Thus, the result of the algorithm can be achieved by starting with a distribution of operations with optimal makespan and iteratively swapping operations in a way such that the resulting cycle times resemble the result of Algorithm 4.6 while not increasing the makespan. Therefore, the algorithm achieves a lower bound on the optimal makespan. □

In the case that no fixed starting sequence $\sigma = (\sigma_1, \ldots, \sigma_l)$ is given, the algorithm has to be altered in a minor way: Instead of setting Γ_n^k to the cycle time of cycle $(l+m-k)$ of the starting sequence in line 10 it can simply be set to zero. In this case, the list Γ^k does not need to be sorted again in each iteration because the operations in line 7 retain the sorting of the list and all processing times are at least zero. Thus, the list Γ^k is already sorted in non-increasing order in this case.

4.4.2 Maximum lateness

To achieve a lower bound on the maximum lateness for synchronous flow shop instances, we will use the results of Section 3.3.1 which gave an efficient way to solve the problem in the case of only one dominant machine. Similar to constructing lower bounds on the makespan, we will use this algorithm on relaxed versions of the original problem. For an instance I with jobs J of the synchronous flow shop problem $F|\text{synmv}|L_{\max}$ consider the following relaxation: For $k \leq m$ construct an instance I_k with the same jobs J, but set all processing times on machines M_i with $i \neq k$ to zero. Clearly, this instance of a synchronous flow shop with a single dominating machine is a relaxation of the original instance and an optimal solution of the problem $F|\text{synmv}, \text{dom}(k), p_{ij}^{ndom} = 0|L_{\max}$ for this relaxed instance is a lower bound on the minimal maximum lateness of the original instance I. Thus, also the maximum of these bounds is a lower bound for the original instance.

Input: set of jobs J, fixed sequence of jobs $\sigma = (\sigma_1, \ldots, \sigma_l)$
Input: cycle times c_t for $t = 1, \ldots, l + m - 1$
Output: lower bound on $L_{\max}$
1 $LB \leftarrow L_{\max}(\sigma)$
2 **for** $k = 1, \ldots, m$ **do**
3 $\quad S \leftarrow \sum_{h=1}^{l+k-1} c_h$
4 $\quad J' \leftarrow \emptyset$
5 $\quad$ **for** $j \in J$ **do**
6 $\quad\quad$ **if** $j \notin \sigma$ **then**
7 $\quad\quad\quad$ Create a job j' with $p_{kj'} = p_{kj}; p_{hj'} = 0$ for $h = 1, \ldots, m; h \neq k$ and $d_{j'} = d_j - S$ and add it to J'
8 $\quad\quad$ **end**
9 $\quad$ **end**
10 $\quad L \leftarrow$ minimum lateness for jobs J' as determined by Algorithm 3.2
11 $\quad LB \leftarrow \max(LB, L)$
12 **end**
13 **return** LB

Algorithm 4.7: Lower bound on the maximum lateness for a fixed starting sequence

For a given fixed starting sequence $\sigma = (\sigma_1, \ldots, \sigma_l)$, Algorithm 4.7 determines a lower bound on the maximum lateness. Therein, the lower bound for all sequences that start with this fixed starting sequence is determined as follows: First, the maximum lateness of the starting sequence is calculated as this can not be decreased independent of the sequence of the remaining jobs. Afterwards, a lower bound on the maximum lateness of the remaining job is determined. For each $k = 1, \ldots, m$, the remaining jobs are relaxed in such a way that only their processing times on machine M_k are considered. For this, an instance I_k of $F|\text{synmv}|kL_{\max}$ is constructed (cf. line 4). To simulate that the processing of jobs on machine M_k in the complete schedule would start in the $(l + k)$-th cycle, the due dates of

each job j' in the relaxed instance are set to $d_j - \sum_{h=1}^{l+k-1} c_h$ (cf. line 7). Then, the maximum lateness of the constructed instance gives a lower bound for the maximum lateness for the remaining jobs.

Theorem 4.6. *Let I be an instance of $F|synmv|L_{\max}$ and let $\sigma = (\sigma_1, \ldots, \sigma_l)$ be a fixed starting sequence. Then, Algorithm 4.7 finds a lower bound on the minimal maximum lateness in time $O(m \cdot n^3 \log n)$.*

Proof. The arguments are similar to the proof of Theorem 4.4. □

4.4.3 Total completion time

Similar to the minimization of the maximum lateness we will use the results of Section 3.3.1 which gave an efficient way to solve the problem in the case of only one dominant machine. For an instance I with jobs J of the synchronous flow shop problem $F|\text{synmv}|\sum C_j$ construct instances I_k for $k \leq m$ with the same jobs J, but set all processing times on machines M_i with $i \neq k$ to zero. Again, an optimal solution of $F|\text{synmv}, \text{dom}\,(k)\,, p_{ij}^{ndom} = 0|\sum C_j$ of each of these instances offers a lower bound on the total completion time of the original instance.

Input: set of jobs J, fixed sequence of jobs $\sigma = (\sigma_1, \ldots, \sigma_l)$
Input: cycle times c_t for $t = 1, \ldots, l + m - 1$
Output: lower bound on $\sum C_j$

```
1  LB_0 ← Σ C_j(σ)
2  for k = 1, ..., m do
3  |  S ← Σ_{h=1}^{l+k-1} c_h
4  |  J' ← ∅
5  |  for j ∈ J do
6  |  |  if j ∉ σ then
7  |  |  |  Create a job j' with p_{1j'} = p_{kj}; p_{hj'} = 0 for h = 1, ..., m; h ≠ k
   |  |  |    and add it to J'
8  |  |  end
9  |  end
10 |  L ← minimal total completion time for jobs J'
   |    as determined by SPT ordering
11 |  LB_k ← LB_0 + |J'| · S + L
12 end
13 LB ← max_{k=0}^{m} (LB_k)
14 return LB
```

Algorithm 4.8: Lower bound on the total completion time for a fixed starting sequence

For a fixed starting sequence $\sigma = (\sigma_1, \ldots, \sigma_l)$, Algorithm 4.8 calculates a lower bound on the total completion time. The algorithm works in the same way as Algorithm 4.7 in Section 4.4.2 above. First, the total completion time for the starting sequence is calculated.

Afterwards, for each $k = 1, \ldots, m$, an instance of $F|\text{synmv}, \text{dom}\,(k)\,|\sum C_j$ is constructed. For each of the remaining jobs, its completion time is increased by the cycle times of the first $l + k - 1$ cycles (cf. line 11).

Theorem 4.7. *Let I be an instance of $F|synmv|\sum C_j$ and let $\sigma = (\sigma_1, \ldots, \sigma_l)$ be a fixed starting sequence. Then, Algorithm 4.8 finds a lower bound on the total completion time in time $O(m \cdot n \log n)$.*

Proof. Once again, the arguments are similar to the proofs of Theorems 4.4 and 4.6. □

Another lower bound for minimizing the total completion time of synchronous flow shop problems can be achieved by transforming the problem into a parallel machine problem. For parallel machines, finding a schedule that minimizes the total completion time is solvable in polynomial time even if the number of machines is part of the input (cf. Brucker (2007)). The algorithm consists of first sorting the jobs in SPT order and then iteratively scheduling each job on the machine with the smallest index on which the fewest jobs have been scheduled so far. Algorithm 4.9 determines the optimal total completion time for a parallel machine problem. For each machine M_i, the completion time of the last job currently scheduled on M_i is stored in list P (cf. line 2). Starting with the first machine, the jobs are appended iteratively as described above. For each job j that is scheduled on machine M_i, its completion time can be calculated by the completion time of the preceding job on machine M_i, increased by p_j (cf. 6). In the following, we show that the optimal total completion time of the parallel machine problem is a lower bound on the total completion time of the synchronous flow shop problem.

```
   Input: set of jobs J
   Output: optimal value for ∑ C_j
 1 C ← 0
 2 P ← list of size m, all values 0
 3 Sort jobs in SPT order
 4 i ← 1
 5 for j = 1, ..., n do
 6 |   P_i ← P_i + p_j
 7 |   C ← C + P_i
 8 |   i ← i + 1
 9 |   if i > m then
10 |   |   i ← i − m
11 |   end
12 end
13 return C
```

Algorithm 4.9: Optimal algorithm for $P||\sum C_j$

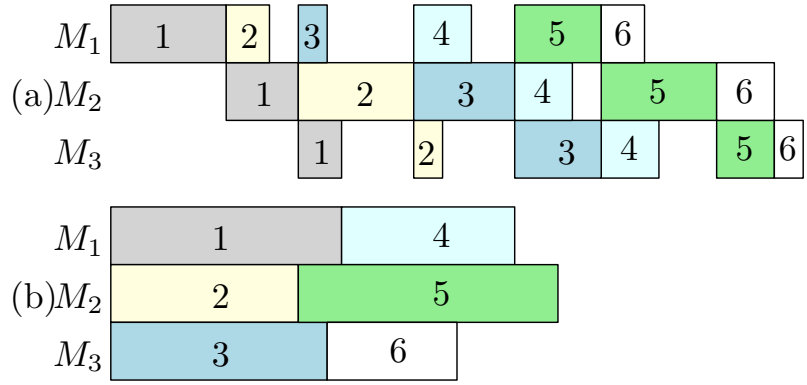


Figure 4.8: Exemplary transformation of a synchronous flow shop schedule (a) into a parallel machine schedule (b). The completion times of all jobs in the parallel machine schedule are at most as large as the completion times of the corresponding jobs in the synchronous flow shop.

Theorem 4.8. *Let I be an instance of $F|synmv|\sum C_j$ with J and optimal total completion time C. Construct an instance I' of the parallel machine problem $P||\sum C_j$ with jobs J' and $p_{j'} = \sum_{i=1}^{m} p_{ij}$ for all $j \in J$. Let C' be the optimal total completion time of I', then it holds that $C' \leq C$.*

Proof. Let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be a sequence of jobs for I with optimal total completion time C. To improve readability, let w.l.o.g. $n \bmod m = 0$. Construct the following schedule σ^P of the parallel machine problem: For $i = 1, \ldots, m$, schedule all jobs j' for which j is scheduled in position $i + km$ with $k = 0, \ldots, \frac{n}{m} - 1$ in this order on machine M_i. Consider the start and completion times of all jobs: In the synchronous flow shop, job σ_i starts at time $\sum_{h=1}^{i-1} c_h$ and is completed at time $\sum_{h=1}^{i+m-1} c_h$. In the parallel machine environment, for $k = 1, \ldots, m$, job $(\sigma_k)'$ starts at time 0 and is completed at time $p_{(\sigma_k)'}$. As $p_{(\sigma_k)'} = \sum_{i=1}^{i+m-1} p_{i,\sigma_k} \leq \sum_{h=1}^{k+m-1} c_h$ for all $k = 1, \ldots, m$, the completion time of the first m jobs in the parallel machine environment is at most as large as in the synchronous flow shop. Figure 4.8 depicts the situation for the first six jobs for a synchronous flow shop with three machines. Iteratively, the start and completion times of all jobs $(\sigma_k)'$ in the parallel machine environment are at most as large as the start and completion times of the corresponding jobs σ_k in the synchronous flow shop. Thus, also the total completion time C^P of the schedule σ^P is at most as large as the total completion time C. Since $C' \leq C^P$ has to hold, it follows that $C' \leq C$. □

For a given a starting sequence, we can alter Algorithm 4.8 by replacing lines 2-11 with transforming the flow shop instance into an instance of a parallel machine problem as described above and calculating the final lower bound in 13 by

$$LB \leftarrow LB_0 + |J'| \cdot \sum_{h=1}^{l} c_h + LB_P$$

where LB_P is the optimal total completion time of the transformed instance as calculated by Algorithm 4.9.

Chapter 5

Heuristic methods

Due to the complexity of the synchronous flow shop problem, finding an optimal solution for larger problem instances may not be computationally tractable using exact methods discussed in Chapter 4. Computational results (cf. Chapter 7) show that the optimal makespan of synchronous flow shop instances consisting of 5 machines and 20 jobs in general can not be computed within a 30 minute time limit. Minimizing the maximum lateness and minimizing total completion time turn out to be even more difficult in general. Therefore, in this chapter we will discuss various approaches to determine good solutions in reasonable time for larger instances of synchronous flow shops.

In the literature there exist several classification schemes for flow shop heuristics. The most prevalent way is to distinguish between constructive and improvement heuristics. A constructive heuristic can be described as a procedure which creates feasible solutions for the underlying optimization problem from scratch. Improvement heuristics alter feasible solutions in order to obtain solutions with better objective value. Recent reviews and surveys of heuristics for classical flow shop scheduling include Framinan et al. (2004) as well as Ruiz and Maroto (2005) for minimizing makespan and Framinan et al. (2005) as well as Pan and Ruiz (2013) for minimizing total completion time.

In their survey, Framinan et al. (2004) consider a third, separate stage of heuristics in the form of index development. In their framework, the index development stage consists of sorting the jobs according to some property based on their input data or by transforming the problem into an analogous problem that can be easily solved. The sequence of jobs obtained during the index development stage can either be used directly as a solution or serve as a sorting method which is used for constructive and improvement heuristics. While the introduction of index development as an additional stage of heuristics can be useful when distinguishing between heuristics that only differ by their initial sorting procedure, it is not always easy to clearly identify an algorithm as either an index development or an iterative solution construction routine. In Framinan et al. (2005), heuristics are further divided into simple and composite methods. Therein, a heuristic is regarded as composite if it employs another heuristic for one of the three stages. Otherwise, it is called simple.

This chapter is structured as follows. In Section 5.1 we will describe constructive heuristics. These heuristics will be considered for the three basic objective functions discussed

in this thesis; minimization of makespan, maximum lateness and total completion time. Improvement heuristics will be discussed in Section 5.2. Finally, in Section 5.3 we present a multi agent framework.

5.1 Constructive heuristics

In this section, we present constructive heuristics for synchronous flow shop problems. A heuristic solution can be constructed by iteratively adding one or more jobs to an initially empty sequence based on the attributes of the jobs and the current partial sequence. Alternatively, the problem can be transformed into a problem that can be solved more efficiently. Afterwards, the solution of the transformed problem is converted back into a solution of the initial problem.

5.1.1 Makespan

Heuristics for minimizing the makespan of classical flow shops are well studied in the literature. Framinan et al. (2004) and Ruiz and Maroto (2005) offer a thorough survey and evaluation of these approaches. In comparison, Soylu et al. (2007) up to now were the only authors to propose heuristics for synchronous flow shop scheduling. One of the strongest heuristics for the classical flow shop is by Nawaz et al. (1983), now referred to as the NEH heuristic: The jobs are initially sorted by their total processing time $T_j = \sum_{i=1}^{m} p_{ij}$ in non-increasing order. Afterwards, the jobs are iteratively inserted into the best position in the hitherto obtained partial schedule. Other variants of this algorithm define different ways to sort and append jobs to the schedule.

Similar to the synchronous flow shop, minimizing the makespan is $\mathcal{NP}$-hard for classical flow shop problems with three or more machines. For two machines, it is solvable in polynomial time by the algorithm of Johnson (1954). Several heuristics to minimize the makespan of classical flow shop problems use Johnson's algorithm by transforming the initial problem instance with more than two machines into a two machine problem. One approach is to aggregate processing times of the machines to obtain an instance of the two machine problem which is then solved optimally and an optimal sequence is transformed back into a solution for the original problem. Variation of these heuristics can be used for synchronous flow shops as well. We can transform instances of general synchronous flow shop problems into instances of two machine problems which can be solved efficiently by the algorithm of Gilmore and Gomory.

Iterative solution construction

Iterative solution construction heuristics consist of two phases, a job selection and a job insertion phase. Soylu et al. (2007) proposed several heuristics for the synchronous flow shop problem, described in the following. The first two heuristics, (S1) and (S2) iteratively assign a not yet assigned job to one of the n possible positions in the schedule. Algorithm 5.1 depicts heuristic (S1). Therein, we generate m different sequences. To construct the i-th sequence, the jobs are initially sorted in non-increasing order of processing time on

machine M_i (cf. line 6). The jobs with large processing times on machine M_i are then scheduled in such a way that their operations on machine M_i are scheduled in cycles in which as many other machines as possible are occupied. In the synchronous flow shop, all machines are occupied in cycles $m, \ldots, n$ while in the other cycles at least one machine is empty. Operations on machine M_i are processed in cycles $i, \ldots, n+i-1$. These cycles are sorted in line 8 in decreasing order of number of machines that are occupied in the respective cycle. Afterwards, the jobs are scheduled such that the job with the k-th largest processing time on machine M_i is scheduled such that its operation on this machine is processed in the cycle with the k-th most occupied machines. Out of the m generated sequences, the one with the best makespan is returned. Obviously, this heuristic performs better when the number of machines is small in comparison to the number of jobs. If the number of machines increases in comparison to the number of jobs, all machines are occupied in almost all cycles.

Input: set of n jobs J
Output: sequence of jobs σ
1 $\sigma \leftarrow$ empty sequence of size n
2 $C \leftarrow \infty$
3 **for** $i = 1, \ldots, m$ **do**
4 $\quad \sigma' \leftarrow$ empty sequence of size n
5 $\quad L \leftarrow$ list of jobs J
6 $\quad$ Sort L, s.t. $p_{ir} \geq p_{is}$ for $r < s$
7 $\quad T \leftarrow [i, \ldots, n+i-1]$
8 $\quad$ Sort T, s.t. for $r < s$ in cycle T_r are at least as much operations processed as in cycle T_s
9 $\quad$ **for** $k = 1, \ldots, n$ **do**
10 $\quad\quad \sigma'_{T_k - i + 1} = L_i$
11 $\quad$ **end**
12 $\quad C' \leftarrow C_{\max}(\sigma')$
13 $\quad$ **if** $C' < C$ **then**
14 $\quad\quad C \leftarrow C'$
15 $\quad\quad \sigma \leftarrow \sigma'$
16 $\quad$ **end**
17 **end**
18 **return** σ

Algorithm 5.1: Heuristic (S1) by Soylu et al.

Algorithm 5.2 shows (S2), the second heuristic proposed by Soylu et al. Therein, we start with a schedule occupied by n idle jobs with processing times zero on all machines which are iteratively replaced by actual jobs until all jobs are scheduled. Let T (cf. line 2) contain all cycles in which the remaining idle jobs start. In each iteration, the maximum processing time of any operation of the remaining non-scheduled jobs is determined in line 4. Then, from all cycles in which the corresponding job can still be started (i.e. replace

an idle job), one is chosen which results in this operation to be processed in the cycle with largest cycle time (cf. line 5).

Input: set of n jobs J
Output: sequence of jobs σ
1 $\sigma \leftarrow$ sequence of n idle jobs with processing times zero on all machines
2 $T \leftarrow \{1, \ldots, n\}$
3 **for** $h = 1, \ldots, n$ **do**
4 | Let O be an unscheduled operation with largest processing time.
| Let k and l be the corresponding machine and job index.
5 | $s \leftarrow \text{argmax}_{t \in T} c_{t+k-1}$
6 | $\sigma_s \leftarrow l$
7 | $T \leftarrow T \setminus \{s\}$
8 **end**
9 **return** σ

Algorithm 5.2: Heuristic (S2) by Soylu et al.

Instead of assigning a job to one of the n possible positions of a schedule, heuristic (S3) by Soylu et al. (2007) starts from an empty sequence and iteratively appends a job to the end of the schedule (cf. Algorithm 5.3). For this, in each iteration the cycle times of the last $m-1$ cycles of the current schedule are evaluated (cf. line 4). Afterwards, one of the remaining jobs is chosen which increases the makespan by the smallest amount (cf. line 5).

Input: set of n jobs J
Output: sequence of jobs σ
1 $\sigma \leftarrow$ empty sequence of size n
2 Append to σ a job j with minimal total processing time $\sum_{i=1}^{m} p_{ij}$
3 **for** $i = 2, \ldots, n$ **do**
4 | Calculate cycle times of cycles $i, \ldots, i+m-2$
5 | Append to σ an unscheduled job j, s.t.
| $\sum_{k=i}^{i+m-2} \max(0, p_{1+k-i,j} - c_k) + p_{mj}$ is minimal
6 **end**
7 **return** σ

Algorithm 5.3: Heuristic (S3) by Soylu et al.

Based on heuristic (S3) by Soylu et al. (Algorithm 5.3) we propose the following adjustments for the selection of the next job in line 5 of Algorithm 5.3:

> **(S3I)**: We define a second criterion in case of ties when two jobs increase the makespan by the same amount. For this, sum up the idle times that would occur for each operation of the job to be scheduled, i.e. $\sum_{k=i}^{i+m-2} \max(0, c_k - p_{1+k-i,j})$. Choose a job with the lowest sum of idle times.

(S3W): Instead of evaluating the two criteria of (S3I) in lexicographic order, use a weighted function of the increase of makespan as well as the sum of idle times to determine the next job.

(S3DE): Start with two empty sequences σ^1, σ^2, one representing the start and the other representing the end of the schedule. In each iteration evaluate the increase in makespan and/or idle time if a job is appended to the back of σ^1 or to the front of σ^2. Afterwards, concatenate both sequences. Lexicographic ordering as well as a weighted function of the two criteria can both be used for this.

Algorithm 5.4 demonstrates the NEH heuristic, developed by Nawaz et al. (1983) for classical flow shop problems. At first, the jobs are sorted in non-decreasing order of total processing times $\sum_{i=1}^{m} p_{ij}$. Afterwards, starting with an empty schedule the jobs are inserted in this order. In each iteration, the next job to be scheduled is inserted in the position of the current schedule which results in a schedule of minimal makespan. The NEH heuristic can be used for all other objective functions of the synchronous flow shops as well.

Input: set of n jobs J
Output: sequence of jobs σ
1 $\sigma \leftarrow$ empty sequence of size n
2 Sort jobs j in non-decreasing order of total processing times $\sum_{i=1}^{m} p_{ij}$
3 **for** $j = 1, \ldots, n$ **do**
4 $C \leftarrow \infty$
5 $b \leftarrow 0$
6 **for** $i = 1, \ldots, j$ **do**
7 $\sigma' \leftarrow \sigma$
8 Insert j in position i in σ'
9 $C' \leftarrow$ makespan of the partial schedule σ'
10 **if** $C' < C$ **then**
11 $C \leftarrow C'$
12 $b \leftarrow i$
13 **end**
14 **end**
15 Insert j in position b in σ
16 **end**
17 Return σ

Algorithm 5.4: Heuristic (NEH) by Nawaz et al.

The NEH heuristic can be further improved by considering all jobs in each iteration:

(NEHA): The jobs are not sorted prior to the iterative insertion. In each iteration evaluate the makespan for the insertion of all remaining jobs for all positions the

remaining jobs can be inserted. Afterwards, choose a job and a position which results in a schedule of minimal makespan.

Instead of using the total processing times of the jobs as an initial sorting for the NEH heuristic, another sorting procedure or some other heuristic may be chosen.

Transformation into a two-machine problem

In the following, we present several approaches to convert an instance of $F|\text{synmv}|C_{\max}$ into an instance of $F2|\text{synmv}|C_{\max}$ in order to employ the algorithm of Gilmore and Gomory. This approach resembles the technique used in Section 4.4.1 to obtain lower bounds where we used a two machine flow shop as a relaxation of general flow shop instances. First, we will focus on synchronous flow shop problems with two non-adjacent dominating machines k_1, k_2. Like in the previous chapters, to improve readability we set $\kappa := k_2 - k_1$ and assume $\kappa \bmod n \equiv 0$. The algorithms can be easily altered to be used if κ is no factor of n. Then, in Algorithm 5.5 the problem is relaxed into a synchronous flow shop problem with two adjacent dominating machines. Afterwards, the sequence obtained for the relaxed instance is split into κ sequences which are used as the subsequences $\sigma^1, \ldots, \sigma^\kappa$ of the original problem.

Input: set of n jobs J, two dominating machines k_1, k_2
Output: sequence of jobs σ
$J' \leftarrow \emptyset$
for $j \in J$ **do**
  Create a job j' with $p_{1j'} = p_{k_1 j}, p_{2j'} = p_{k_2 j}$ and add it to J'
end
$\sigma' \leftarrow$ execute Algorithm 4.1 of Gilmore and Gomory for J'
$\sigma \leftarrow$ empty sequence of size n
$\kappa \leftarrow k_2 - k_1$
for $\lambda = 1, \ldots, \kappa$ **do**
  for $i = 1, \ldots, \frac{n}{\kappa}$ **do**
    $\sigma_{(i-1)\frac{n}{\kappa}+\kappa} \leftarrow \sigma'_{(\kappa-1)\frac{n}{\kappa}+i}$
  end
end
return σ

Algorithm 5.5: Heuristic (GG-Split)

A similar approach can be applied for the general case without machine dominance. In Algorithm 5.6, the general problem is relaxed by setting all processing times except on two machines $1 \leq k_1 < k_2 \leq m$ to zero, thus transforming the problem into a problem with two dominating machines for which Algorithm 5.5 is applied. The sequence obtained by Algorithm 5.5 is evaluated with the original processing times and the best sequence is returned.

```
   Input: set of n jobs J
   Output: sequence of jobs σ
 1 σ ← empty sequence of size n
 2 S ← ∞
 3 for k_1 = 1, ..., m − 1 do
 4 |   for k_2 = k_1 + 1, ..., m do
 5 |   |   σ' ← execute Algorithm 5.5 for J, k_1, k_2
 6 |   |   S' ← C_max(σ')
 7 |   |   if S' < S then
 8 |   |   |   S ← S'
 9 |   |   |   σ ← σ'
10 |   |   end
11 |   end
12 end
13 return σ
```

Algorithm 5.6: Heuristic (GG)

Another possibility to relax an instance of a synchronous flow shop problem into a problem with two adjacent dominating machines is similar to the approach of Campbell et al. (1970) who transformed instances of classical flow shop problems into flow shop problems with two machines in order to apply the algorithm of Johnson. In Algorithm 5.7, an instance of the synchronous flow shop problem with jobs J is transformed into an instance of a two-machine synchronous flow shop problem with jobs J' where the processing times of the jobs J' are calculated by aggregating the processing times of each job on the first i and the last $m - i$ machines for some $1 \leq i \leq m - 1$, i.e. setting $p_{1j'} = \sum_{k=1}^{i} p_{kj}$ and $p_{2j'} = \sum_{k=i+1}^{m} p_{kj}$. Within Algorithm 5.7 this relaxation is performed for all $i = 1, \ldots, m-1$ and the algorithm of Gilmore and Gomory is performed on the relaxed instance. Afterwards, the obtained sequence is evaluated with the original processing times and the best sequence is returned.

Transformation into a traveling salesman problem

Instead of transforming an instance of a synchronous flow shop with more than two machines into an instance with two machines, another possibility is to transform the flow shop scheduling problem into a traveling salesman problem in a similar way as in the algorithm of Gilmore and Gomory. Each job is identified with a node and the distance between nodes is defined based on the processing times of the respective jobs. In the original version, the jobs were simply sorted according to their processing times on machines M_1 and M_2 and the first successor for each job was determined in this way. In the general case we supply the algorithm with a function $c : J \times J \to \mathbb{R}$ which indicates the cost of two jobs being processed in succession and the first successor for each job is determined by solving a bipartite matching problem in a complete bipartite graph $G = (J \cup J', E)$ where J' contains a copy of each job $j \in J$ and edges exist between all $j \in J$ and $j' \in J$ with weights of the

Input: set of n jobs J
Output: sequence of jobs σ
$\sigma \leftarrow$ empty sequence of size n
$S \leftarrow \infty$
for $i = 1, \ldots, m-1$ **do**
  $J' \leftarrow \emptyset$
  for $j \in J$ **do**
    Create a job j' with $p_{1j'} = \sum_{k=1}^{i} p_{kj}, p_{2j'} = \sum_{k=i+1}^{m} p_{kj}$ and add it to J'
  end
  $\sigma' \leftarrow$ sequence obtained by Algorithm 4.1 of Gilmore and Gomory for J'
  $S' \leftarrow C_{\max}(\sigma')$
  if $S' < S$ **then**
    $S \leftarrow S'$
    $\sigma \leftarrow \sigma'$
  end
end
return σ

Algorithm 5.7: Heuristic (CDS)

edges determined by the cost function c. Then, the first successor of each job is its partner in an optimal solution of the bipartite matching problem, once again leading to a number of small disjoint cycles. These cycles are joined similarly to the algorithm of Gilmore and Gomory.

As a possible cost function for two jobs i, j if scheduling the two jobs in direct succession results we can use

$$c_{ij} := \sum_{k=2}^{m} \max(0, p_{k-1,j} - p_{ki}) + p_{mj},$$

i.e. the amount of which the makespan increases if job j is scheduled after job i ignoring all other jobs. This cost function resembles the formula used in heuristic (S3) (cf. Algorithm 5.3). In a similar way to heuristics (S3-I) and (S3-W) the cost function can also be altered to take into account the idle times that arise for job j or a weighted function thereof. In Chapter 7 we evaluate the effects of these cost functions.

5.1.2 Maximum lateness

In comparison to the makespan objective, there is only very limited interest in constructive heuristics to minimize the maximum lateness of classical flow shops in the literature. A popular heuristic is to simply sort the jobs in order of non-decreasing due dates (EDD). Of course, a version of the NEH algorithm described above can be used as well. In each

Input: set of jobs $J = \{0, \ldots, n\}$, cost function $c : J \times J \to \mathbb{R}$
Output: sequence of n jobs σ

$\phi \leftarrow$ list of size $n + 1$
$J' \leftarrow$ copy of jobs J
Create a complete bipartite graph $B = (V, E)$ with $V = J \cup J'$ and $E = \{(j, j') | j \in J, j' \in J'\}$ and weights determined by c
Determine a minimum cost perfect matching in B
$\phi_j \leftarrow$ partner of j in J' in the minimum cost perfect matching
Create a graph $G = (V, E)$ with $V = J$ and $E = \{(j, \phi_j) | j = 0, \ldots, n\}$
$r_{i,j} \leftarrow -(c_{i,\phi_i} + c_{j,\phi j}) + c_{i,\phi_j} + c_{j,\phi_i}$ for all i, j
while G *consists of more than one component* **do**
    Find smallest value $r_{i,j}$ such that i and j are in distinct components
    Join the two components via interchange (i, j)
end
$\sigma \leftarrow$ sequence with length n
$\sigma_1 \leftarrow \phi_0$
for $i = 2, \ldots, n$ **do**
    $\sigma_i \leftarrow \phi_{\sigma_{i-1}}$
end
return σ

Algorithm 5.8: Heuristic (TSP)

iteration of the NEH heuristic the job and/or position is chosen which leads to a minimal maximum lateness.

As we discussed in Section 3.3.1, the EDD schedule in general leads to non-optimal solutions already for $F2|\text{synmv}, \text{dom}(1)|L_{\max}$. However, for a single dominating machine, Algorithm 3.2 finds a schedule with minimal maximum lateness in polynomial time. Thus, one possibility is to transform a general synchronous flow shop into a flow shop with a single dominating machine. The heuristic is similar to Algorithm 5.6 in which a synchronous flow shop instance was transformed into various instances of synchronous flow shops with two dominating machines.

(1DOM): For $k = 1, \ldots, m$ construct an instance I_k of $F|\text{synmv}, \text{dom}(k), p_{ij}^{ndom} = 0|L_{\max}$ in the following way: For each job j construct a job j' with $p_{kj'} = p_{kj}$ and $p_{ij'} = 0$ for all $i \neq k$. Again, the sequence of jobs obtained for the transformed problem can be evaluated as a sequence in the original problem.

In the following, we present another heuristic for minimizing the maximum lateness based on Algorithm 3.2 which was used to find optimal solutions for synchronous flow shop problems for one dominating machine. To obtain a feasible solution for a given threshold value for the maximum lateness, Algorithm 3.1 benefits from the fact that the makespan of a synchronous flow shop with a single dominating machine is independent of the schedule. Thus, when assigning jobs from the back of the sequence, the completion time of each job

is known at the time when it is scheduled. While this does not hold true for more than one dominating machine, we can still use the technique of the algorithm as a heuristic algorithm for the decision problem $L_{\max} \leq L$. Because the makespan is not sequence-independent, we use an approximation T for the makespan. Using this approximation as the completion time of the last job, we determine the set of all jobs j for which $d_j + L \geq T$. Similar to Algorithm 3.1 it would be favorable to schedule a job in the last position such that the completion time of the jobs that are scheduled before the chosen one are as small as possible. In contrast to the case with a single dominating machine this job can not be determined for certain. Therefore, we choose a job that results in the largest sum of cycle times for the cycles it is scheduled in. If no job is feasible, we choose a job with maximal due date. Afterwards, we recalculate the lower bound T on the makespan for the remaining jobs and continue iteratively.

Input: jobs $j = 1, \ldots, n$, threshold $L \in \mathbb{Z}$, approximation of makespan T of the resulting schedule
Output: job sequence σ

$\sigma \leftarrow$ sequence of jobs, initially empty
for $\lambda = 1, \ldots, n$ **do**
  feasibleJobs $\leftarrow$ unscheduled jobs j with $T - d_j \leq L$
  if *feasibleJobs* $\neq \emptyset$ **then**
    $l \leftarrow$ job $j \in$ feasibleJobs that maximizes $\sum_{t=n-\lambda+1}^{n+m-\lambda} c_t$ if scheduled in position $n - \lambda + 1$
  end
  else
    $l \leftarrow$ unscheduled job with maximum due date
  end
  $\sigma_{n-\lambda+1} \leftarrow l$
  Calculate cycle time $c_{n+m-\lambda}$
  $T \leftarrow T - c_{n+m-\lambda}$
end
return σ

Algorithm 5.9: Heuristic (BW) to find a schedule with maximum lateness at most L

If Algorithm 5.9 returns a sequence with maximum lateness $L_{\max} \leq L$, we can use a lower threshold value to minimize the maximum lateness in the same manner as we did in Algorithm 3.2 for the case of a single dominating machine. For this, we can once again decrease the maximum lateness of the returned sequence and use it as a threshold value L for the next iteration. On the other hand, if Algorithm 5.9 does return a sequence with a maximum lateness $L_{\max} > L$, we can not infer that no such sequence exists. Instead, this might be caused by a bad value chosen for the initial approximation of the makespan or because in one of the iterations a wrong job was chosen from the set of feasible jobs. Thus,

it might be beneficial to restart the algorithm with a different approximation value for T and retry the same threshold value L.

Several alternative implementations for heuristic (BW) and the iterative version are possible. For instance, there are many ways to obtain an approximation for the makespan. We can use an upper bound on the minimal makespan as determined by one of the constructive heuristics described in Section 5.1.1. Alternatively, we can use the makespan of the sequence that was obtained by Algorithm 5.9 in a previous step. Within (BW), the value of T could also be reevaluated in each iteration λ (cf. line 11) based on the remaining non-scheduled jobs instead of simply subtracting the cycle time of the $(n-m+\lambda)$-th cycle. Further, a different method to determine which of the feasible jobs should be scheduled in the last position can be applied, e.g. using a job j with the largest total processing time $\sum_{i=1}^{m} p_{ij}$. In Chapter 7, we present a study of various possibilities.

5.1.3 Total completion time

In classical flow shop scheduling, finding an optimal schedule minimizing the total completion time is $\mathcal{NP}$-hard for two or more machines. For a single machine problem, sorting the jobs in non-decreasing order of processing time (SPT) results in an optimal solution. A popular heuristic is to sort the jobs by non-decreasing total processing time or variations thereof. Afterwards, the jobs can be scheduled iteratively in order of the obtained sequence, e.g. with a version of the NEH algorithm or the sequence is altered using improvement heuristics discussed in Section 5.2. As discussed in Section 3.3.1, minimizing the total completion time is efficiently solvable for synchronous flow shops with a single dominating machine via the SPT rule, similar to single machine scheduling. Therefore, we propose the following three methods for index developing for synchronous flow shops.

(StPT): Sequence the jobs in non-increasing order of total processing time $\sum_{i=1}^{m} p_{ij}$.

(SwPT): Sequence the jobs in non-increasing order of weighted processing time $\sum_{i=1}^{m} w_i p_{ij}$ where w_i can be determined in multiple ways.

(SPT1D): For each machine $k = 1, \ldots, m$ sequence the jobs in non-decreasing order of p_{kj}. Use the sequence that results in the lowest total completion time for the actual instance. This approach is similar to Algorithm 5.6 and the (1DOM) heuristic for minimizing maximum lateness.

Further, many of the heuristics that are used for minimizing the makespan also generate good results when applied to the objective function of minimizing the total completion time. Especially heuristics which try to increase the sum of cycle times by the least amount in each iteration can be used for minimizing the total completion time.

5.2 Improvement Heuristics

In this section we will discuss various alternatives to improve schedules which were obtained by the constructive heuristics proposed in the previous section. The purpose of improvement heuristics is to take a schedule and improve it in regard of the underlying objective function.

In the following, we will focus on local search procedures that employ a neighborhood structure to explore the search space and change the solutions according to this structure. These local search methods are also classified as trajectory methods and more generally as metaheuristics. A metaheuristic is formally defined as an "iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions." (cf. Osman and Laporte (1996)) In this section, we describe some of the more contemplated metaheuristics which have been used for several scheduling problems including the classical flow shop. There exist an abundance of further metaheuristics for problems in scheduling in general and flow shop scheduling in particular that are based on other concepts which will not be discussed in this thesis. A good introduction in various methods is given by Osman and Laporte (1996), while Framinan et al. (2004) as well as Ruiz and Maroto (2005) discuss several applications of metaheuristics on flow shop scheduling.

In Section 5.2.1 we will discuss local search procedures which can be applied to all synchronous flow shop problems described in this thesis. Afterwards, in Section 5.2.2 we will describe an approach that can be applied to synchronous flow shops with two dominating machines.

5.2.1 Local search for general synchronous flow shops

The basic idea of a local search procedure is as follows: Starting from an initial solution which may be created by a constructive heuristic as described in Section 5.1, the local search iteratively makes changes to the current solution and tries to find a solution with a better objective value. In most cases, this is achieved by employing a neighborhood structure as described in the following. Given a schedule σ, its neighborhood is represented by schedules that can be created by conducting minor changes on σ. Obviously, neighborhood structures rely on the representation of these schedules. For instance, within this work, we predominantly represent synchronous flow shop schedules by a permutation of the jobs. Then, neighborhoods can consist of changing the permutation of jobs. Let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be a permutation of jobs. Popular neighborhoods to get a new permutation σ', representing a new solution of the synchronous flow shop, are

- *Swap neighborhood.* The position of two jobs in the permutation is swapped, e.g. $\sigma'_i = \sigma_j, \sigma'_j = \sigma_i$ and $\sigma'_k = \sigma_k$ for $k \neq i, j$.

- *Adjacent pairwise interchange (api).* Only swaps between two adjacent jobs are considered.

- *Shift (or insertion) neighborhood.* A job is removed from the permutation and inserted in a different position, e.g. in a left shift: $\sigma'_i = \sigma_j$, $\sigma'_k = \sigma_{k-1}$ for $k = i+1, \ldots, j-1$, $\sigma'_k = \sigma_k$ otherwise.

Iterative improvement

Starting from an in initial solution, either the complete neighborhood is evaluated and the solution is changed to a neighbor with the best objective value ("best fit") or the first neighbor with an objective value that is better than the current solution's value is used ("first fit"). The latter approach can be beneficial when evaluating neighbors requires a lot of computation time or when solving large instances. The procedure may either be executed for a predetermined number of iterations or continue until at one point there exists no neighbor with a better objective value than the current solution.

For minimizing the makespan of synchronous flow shops, Soylu et al. (2007) discussed two improvement strategies:

> **(I1)**: Iteratively go through the sequence from the back. In each iteration i evaluate the pairwise interchange of job σ_{n-i-1} with jobs σ_{n-i-2} and σ_{n-i}, respectively. Execute the interchange which leads to biggest improvement, if any.
>
> **(I2)**: Evaluate all pairwise interchanges and execute one which improves the objective value the most, if any. Continue until all interchanges lead to an inferior objective value.

For classical flow shop scheduling, there also exist several other iterative improvement procedures to improve the makespan:

> **(SU)**: Heuristic proposed by Suliman (2000). Starting with job $j = 1$ evaluate the interchange of job j with the job currently succeeding j in the sequence and make a pairwise interchange of the jobs if this leads to a better objective value. If an interchange was performed, iteratively continue with job j and evaluate the pairwise interchange with its new successor in the schedule. Continue until the interchange would lead to an inferior objective value. Proceed with all other jobs $j = 2, \ldots, n$.
>
> **(RACS)**: "Rapid access with close order search" heuristic proposed by Dannenbring (1977). For $i = 1, \ldots, n-1$ evaluate the pairwise interchange of jobs σ_i and σ_{i+1} and execute each interchange whenever it leads to an improvement.
>
> **(RACE)**: "Rapid access with extensive search" heuristic proposed by Dannenbring in which RACS is performed as long as an improvement is achieved. This can also be performed on I1 and SU.

While all of these improvement heuristics were created for minimizing the makespan, they can be used for other objective functions as well. For minimizing the maximum lateness we further propose the following heuristic:

(LR): Iteratively go through the sequence from the back. For $i = 1, \ldots, n-1$, determine the set of jobs out of $\sigma_1, \ldots, \sigma_{n-i}$ that can be moved to position $n-i+1$ without increasing the maximum lateness. Of all these jobs, either choose one that minimizes the lateness the most or in case of a tie, as a secondary criterion choose a job that minimizes the length of the schedule.

In Chapter 7 we will evaluate these heuristics as well as iterative improvement algorithms that employ the swap and shift neighborhood. While iterative improvement procedures are easy to implement and can be used to search the solution space very quickly, their biggest weakness lies in the inability to escape local optima.

Simulated annealing

In contrast to iterative improvement, local search procedures based on simulated annealing do not terminate as soon as all neighbors of the current solution have an inferior objective value. Instead, neighbors with worse objective values are accepted with a defined probability to form the next solution. The probability to accept worse neighbors is high in the beginning but is decreased during the run of the procedure. The procedure is named simulated annealing because it imitates the physical process of a hot material that slowly cools down (cf. Zimmermann (2008)). In this analogy the probability to accept a worse neighbor is compared to the temperature of the annealing process, it is "hot" in the beginning and then gets colder. On the extreme points, given a high enough probability to accept inferior neighbors, the procedure is closer to a random walk through the solution space, while in the later stages with low probabilities it is similar to a greedy iterative improvement algorithm.

Tabu search

Similar to simulated annealing, tabu search may choose neighboring solutions of inferior objective value as the next solution. The procedure employs a so-called tabu list which serves as a memory of formerly visited solutions to prevent cycles, i.e. the possibility to return to a solution during the run of the procedure. The tabu list does not necessarily store complete solutions but rather attributes of solutions or the move made in the neighborhood. A neighbor is accepted to be the next solution either if it is not tabu or if it has a better objective value than the best hitherto found solution. Thus, if it is not tabu, a neighbor may be accepted even if it has a worse objective value than the current solution.

5.2.2 Tabu search for two dominating machines

For synchronous flow shops with two dominating machines k_1, k_2, we can obtain a more efficient heuristic by using a different representation of the schedules based on the subsequences of jobs. For this, let σ be a schedule of jobs with subsequences σ^λ for $\lambda = 1, \ldots, \kappa = k_2 - k_1$. Then, for each of the subsequences σ^λ, an optimal sequence of the jobs can be obtained efficiently by the algorithm of Gilmore and Gomory (1964). Thus, instead of describing each subsequence by its explicit order, it is possible to only represent it by the set of jobs

that are part of the subsequence and to determine the order when needed, thereby vastly reducing the space of possible solutions. The main benefit of this representation is that no moves within subsequences need to be considered.

Using this representation, a solution is described by a collection $\mathcal{S}$ of κ disjoint subsets $S_1, \ldots, S_\kappa$ with $\bigcup_{k=1}^{\kappa} S_k = J$. Then, a corresponding schedule for problem $F|synmv, dom(k_1, k_2), p_{ij}^{ndom} = 0|C_{\max}$ is constructed as follows. Each subset S_k is considered as an instance of problem $F2|synmv|C_{\max}$ with jobs $j \in S_k$ and $p_{1j} = p_{k_1 j}, p_{2j} = p_{k_2 j}$. An optimal subsequence σ^λ is computed by the algorithm of Gilmore and Gomory. The complete sequence σ for the original problem is then again defined as $\sigma_{k+l\kappa} := \sigma_l^k$ for $k = 1, \ldots, \kappa; l = 0, \ldots, \frac{n}{\kappa} - 1$. As described in Section 2.3.2 its makespan is calculated by the sum of the makespans of the subsequences σ^k.

Using this representation, we define a neighborhood by swapping two jobs from two distinct subsets of a solution. Thus, if $\mathcal{S}$ is the current solution with disjoint subsets $S_1, \ldots, S_\kappa$, a neighbor $\mathcal{S}'$ with disjoint subsets $S'_1, \ldots, S'_\kappa$ can be constructed by choosing two subsets k, l and two jobs $u \in S_k, v \in S_l$ and setting $S'_k = (S_k \setminus \{u\}) \cup \{v\}, S'_l = (S_l \setminus \{v\}) \cup \{u\}$ as well as $S'_h = S_h$ for all $h \neq k, l$.

Using this neighborhood within a tabu search, we define the tabu list as pairs of jobs that were swapped between two subsets. If we move to the neighbor $\mathcal{S}'$ which involves swapping the two jobs k, l, we add the pair (k, l) to the tabu list.

5.3 Asynchronous teams

In this section we describe another metaheuristic approach employing a multi agent system, the asynchronous teams. One of the greatest strengths of asynchronous teams lie in their ability to be well applicable to decompose highly complex problems. Therefore, they are especially useful for extensions of synchronous flow shops. The idea of asynchronous teams is to combine several constructive and improvement heuristics and to use them on the same problem instance in order to get the best possible result. In this section we will introduce asynchronous teams and show how they can be used for synchronous flow shop problems. Akkiraju et al. (2001) and Rachlin et al. (1999) state that asynchronous teams, though used very seldom in recent literature, provide good results in rich optimization problems.

In Chapter 6 we will show how we employed an asynchronous team procedure in a practical application resembling a synchronous flow shop with resources and cyclic changeovers.

5.3.1 Introduction to asynchronous teams

The notion of an asynchronous team (A-team) is defined as "AI architecture that consists of multiple problem-solving methods (called agents) working together on a common problem" (see Talukdar et al. (1998)). Examples of usage include Akkiraju et al. (2001), Murthy et al. (1997) and Rachlin et al. (1999). They cite the A-team approaches for their products as "well received in the marketplace and currently being used in several paper mills within North America" (Akkiraju et al. (2001)) and note that "companies have reported substantial benefits using these technologies" (cf. Rachlin et al. (1999)). Communication (and

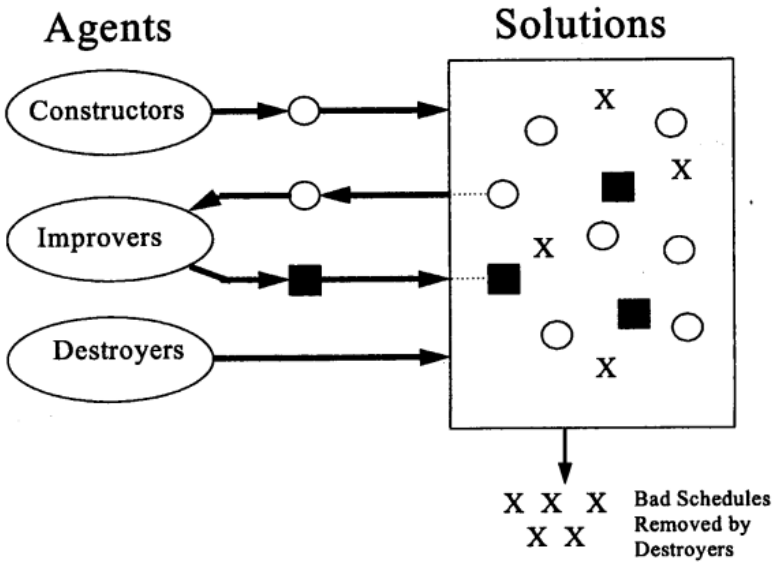


Figure 5.1: Structure of the asynchronous team architecture as presented by Murthy et al. (1997)

cooperation) takes place through a shared population of candidate solutions (cf. Murthy et al. (1997)). In an A-team three types of agents exist:

- *Constructors* which create new solutions and add them to the current population. These resemble the constructive heuristics as described in Section 5.1.
- *Improvers* which take solutions from the population and alter them. They resemble improvement heuristic as described in Section 5.2.
- *Destroyers* which remove weak solutions from the population.

The asynchronous team architecture is depicted in Figure 5.1 based on the work of Murthy et al.. The constructors and destroyers are used to control the size of the population. The destroyers' goal is to remove bad solutions from the population and to keep the population diverse by removing those solutions which are too similar to others. Constructors are used to refill the population by inserting starting solutions obtained from constructive heuristics. Each improver agent has its own set of rules on how he may act on a solution. Therefore, it is also easy to incorporate multiple objective functions where each agent tries to optimize the solution only according to its primary objective. Another approach might be to have each agent altering solutions according to different neighborhoods. There is also the possibility to let each agent simulate a distinct solution approach with alternating parameters taking one solution from the population as a starting solution. As it is left to the designer of an A-team system how sophisticated each agent should be, each agent can do as little as making only one move within its defined neighborhood or as much as doing a full local search until it reaches a local minimum.

5.3.2 Asynchronous teams for synchronous flow shops

For the synchronous flow shop model, the agents of the asynchronous team can apply heuristics as described in the previous sections to find good solutions. The constructor agents will employ constructive heuristics as described in Section 5.1. The improver agents work on the solutions in the solution pool using techniques of Section 5.2. We can use sophisticated agents which perform simulated annealing or tabu search for a couple of iterations as well as simple agents which only make small changes on the solutions like performing a single move in a defined neighborhood. The destroyers greedily remove the solutions with the worst objective values from the pool.

5.3.3 Asynchronous teams for synchronous flow shops with resources and changeovers

The strength of the asynchronous team architecture lies in its capability to be quickly adaptable when the underlying problem is extended. Further, it lends itself very well to decomposition approaches like decoupling the assignment of resources and the construction of schedules with little idle times. In the following, we will discuss two extensions for the asynchronous team structure discussed in the previous section. We will focus on the addition of resources and synchronous flow shops with changeover times. In Chapter 6, we show how an asynchronous team with these extensions is used to optimize a real world synchronous production system.

When dealing with cyclic production and changeover times, the constructive heuristics discussed in Section 5.1 may provide initial schedules which have low sums of cycle times but need a lot of changeovers. Therefore, we will add further constructors and improvers which only consider the minimization of changeover times. Since we do not take into account the processing times of the jobs in this case, we use another presentation of a schedule $\sigma = (\sigma_1, \ldots, \sigma_n)$. As changeovers take part between jobs that have a distance of m in the sequence, we define the following subsequences π^λ for $\lambda = 1, \ldots, m$ where $\pi^\lambda_i = \sigma_{\lambda+(i-1)m}$ for $i = 1, \ldots, \frac{n}{m}$ (w.l.o.g. we assume $n \bmod m = 0$). Then, a job i succeeds a job j in a subsequence when job i starts its processing on machine M_1 in the cycle after job j is completed. A changeover is necessary whenever two consecutive jobs in a subsequence belong to a different job family. We chose the letter π to avoid confusion with the concept of subsequences defined for synchronous flow shops with two dominating machines. For the schedule of the three-machine synchronous flow shop depicted in Figure 5.2, we obtain the following three subsequences: $\pi^1 = (1, 4, 7), \pi^2 = (2, 5, 8)$ and $\pi^3 = (3, 6, 9)$.

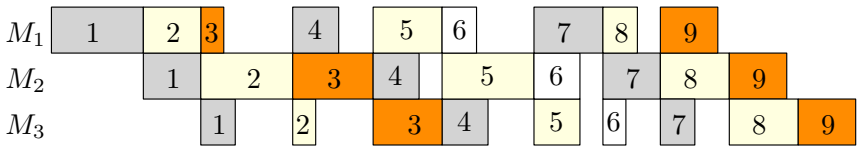


Figure 5.2: Three-machine synchronous flow shop giving rise to the subsequences $\pi^1 = (1, 4, 7), \pi^2 = (2, 5, 8)$ and $\pi^3 = (3, 6, 9)$

We can add the following constructors. Both try to construct schedules with few changeovers.

(C1) Consider the individual job families $f_1, \ldots, f_k$. Denote with $|f_i|$ the number of jobs in job family f_i. Start with job family f_1 and schedule all jobs of the family in positions π^1_h for $h = 1, \ldots, \max\left(|f_1|, \frac{n}{m}\right)$. If the job family is larger than $\frac{n}{m}$, schedule the remaining jobs of the family in the beginning of subsequence σ^2. Continue iteratively with all job families f_i by scheduling all jobs of the family consecutively after the jobs of family f_{i-1} in the same subsequence. Whenever a job is scheduled in position $\frac{n}{m}$ of subsequence π^l, schedule the remaining jobs of the family in the first positions of the next subsequence π^{l+1}. We can add an element of randomization to this step by shuffling the individual job families before executing the constructor.

(C2) Sort the individual job families in such a way that the number of changeovers is minimal. If changeover times are constant, this can be done in polynomial time (cf. Brockmeyer (2014)), but may still take considerably more time than the greedy approach. Note that this does not contradict the results of Section 3.4.4 as we do not consider the individual processing times.

We can also add the following improver:

(C3) For two subsequences π^k and π^l and two positions i, j as well as an integer h with $\max(i + h, j + h) \leq \frac{n}{m}$, swap jobs $\pi^k_i, \ldots \pi^k_{i+h}$ with jobs $\pi^l_j, \ldots, \pi^l_{j+h}$.

Chapter 6

Practical application

In this chapter we will present a practical application of a synchronous flow shop problem with changeovers. The real world example describes the production process of a subcontractor for kitchen installers who uses circular production units with synchronous transport are used to assemble shelf boards for kitchen elements. In the following, we will at first describe the production process with all of its constraints in Section 6.1 and then embed the problem more formally into our framework in Section 6.2. Since the problem is too difficult to solve exactly for real instances, we describe a multi agent approach based on A-Teams in Section 6.3. Later, in Chapter 7, the obtained algorithms are tested on real world data received from the practitioner and compared to the actual planning. The study of the practical application discussed in this paper has already been published in Waldherr and Knust (2014).

6.1 Problem description

The main process of producing the shelf boards consists of gluing wooden base plates to surrounding contours. For the assembly, a gluing form is used which holds both elements so that they can be glued together by a gluing robot. Both, plates and contours, are produced in a prior production step which has to be scheduled as well to assure that they are available in time to be glued together. The gluing process is done on three parallel circular production units. On each of the production units, eight stations are installed on the conveyor system in which gluing forms can be inserted for the production and transport of the shelf boards (see Figure 6.1).

There are distinct types of shelf boards which need different gluing forms for their production. Only a limited number of gluing forms for each type of shelf board is available. During the production process, every station is equipped with exactly one gluing form and every gluing form can be used on at most one station simultaneously. Additionally, each gluing form is only compatible with a subset of stations. Thus, for producing a shelf board a compatible form and a corresponding compatible non-occupied station at one of the three production units have to be available. Whenever a finished board is removed from a station, the gluing form remains on the station and either another product of the

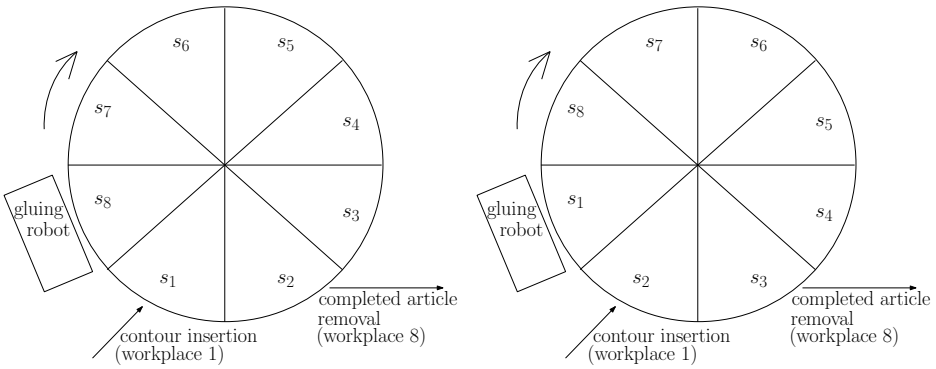


Figure 6.1: Production unit layout and one turn of all stations

same type can be inserted and produced on this station or the gluing form needs to be changed to allow for the production of a different type of shelf board. In the latter case a changeover time occurs in which the station is reconfigured and the new gluing form is inserted. During this changeover period the whole production unit has to wait for the amount of time required to change the forms.

The company takes orders from customers for a given quantity of product items and an assigned due date specifying a day until all items of the corresponding order have to be produced. The main goal of the company is to find a production schedule such that the number of late orders is minimized and as a second criterion to minimize the total lateness of such orders. A third (minor) target is to maximize the number of produced items during a given time frame.

6.2 Formal definitions

In the following we will describe the problem more formally. There exists a set of products (shelf boards) $P = \{p_1, \ldots, p_{|P|}\}$, a set of gluing forms $F = \{f_1, \ldots, f_{|F|}\}$, and a set of stations $S = \{s_1, \ldots, s_{24}\}$ (eight stations at each of the three circular production units). The orders to be scheduled form a set $O = \{o_1, \ldots, o_{|O|}\}$. Each order o_j specifies a single associated product $a_j \in P$ which has to be produced, its order volume v_j, i.e. the quantity of products to be produced, and a due date d_j (day). Thus, an order o_j consists of v_j jobs, each one presenting the production of one item of the associated product a_j. For each product $p \in P$, the gluing time $t^g(p)$ and the insertion time $t^i(p)$ are known. All other times are negligible. Additionally, for each product $p \in P$ the subset $\mathcal{F}(p) \subseteq F$ indicates which gluing forms can be used to produce the product. For each gluing form $f \in F$ the subset $\mathcal{S}(f) \subseteq S$ indicates which stations the gluing form is compatible with (see Figure 6.2).

Each circular production unit resembles a synchronous flow shop on eight machines where the machines M_1 and M_2 are dominating: At machines M_1 the plates and contours are inserted and at M_2 they are glued together, requiring processing times of t^i and t^g, respectively. After being glued together, the shelf board remains on the unit to dry until it is

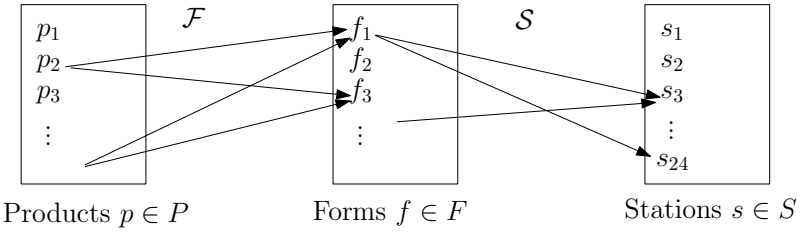


Figure 6.2: Dependencies of products, gluing forms and stations. Product p_2 may only be produced using gluing forms f_1 and f_3, form f_1 is compatible only with stations s_3 and s_{24}.

removed (workstep represented by machine M_8). The drying period and the time to remove the shelf boards from the unit are negligible and independent of the individual board. The gluing forms are resources that are needed during the whole production process of a shelf board. Further, gluing forms need to be exchanged to allow production of a shelf board of a different type on the same station. The gluing forms define disjoint job families. Two jobs are part of the same job family if and only if they can be produced with the help of the same gluing forms. The changeover time is constant and independent from the individual gluing forms. Because of the complexity of the problem and the need of the practitioner to have a rough idea of the production plan to allow for production of raw material, the problem was transformed into a hierarchical scheduling problem with two stages: In the first stage, production is only coarsely assigned on a daily basis without taking into consideration the sequence dependencies caused by the synchronous movement and the changeovers. In the second stage, the production is then planned for the current day based on the available raw material. As only the second stage deals with the synchronous movement of the production units, we refer to Waldherr and Knust (2014) for a full description of the hierarchical planning as well as the scheduling of the first stage. In the following we will cover the second stage where only a single day is to be scheduled.

In general, the number of orders for which raw material is available and thus the number of jobs to be produced is larger than the number of cycles that can be processed on each production unit on a single day. Within the problems discussed previously in this thesis, all jobs had to be scheduled and the number of cycles of the final schedule was known. In comparison, in the second stage of the production process we need to determine a schedule of a subset of jobs which is to be produced in a specified time frame. The actual number of cycles that can be completed within this time frame is not known beforehand and depends on the processing times and number of changeovers of the scheduled jobs. To model this, we define a number Γ of cycles for each production unit which has to be large enough to cover the number of hours to be planned on the specific day. Then, instead of scheduling all orders, only a subset of (possibly partial) orders will be chosen and assigned to these cycles. The schedule of the single day is then obtained by using all cycles that are completed within the time frame.

We denote by $C = \{1, \dots, \Gamma\} \subset \mathbb{N}$ the set of cycles that can be scheduled. Then, we need to determine an assignment $\mathcal{C} : C \to O \times F$ for each production unit. Therein, for each

cycle $c \in C$ we specify which order the shelf board that is inserted on the first machine in this cycle belongs to and which gluing form is used for its production. W.l.o.g. we assume that in the first cycle of the schedule the shelf board and gluing form are inserted at station s_1 at the first production unit and s_9 and s_{17} on the other units, respectively. Thus, in cycle c a job and a gluing form are inserted into station s_i with $i = (c-1) \bmod 8 + 1$ (analogous for the other two units). The assignment may also be empty if no shelf board is inserted on a station in a cycle. If for some $k \in \mathbb{N}$ the order o as well as the gluing form f in cycles $i + 8l$ for $l = 0, \ldots, k-1$ are the same, we will call this successive production of an order on the same station using the same gluing form an order fragment of order o starting at cycle i of size k.

In accordance with the practitioner we evaluate a schedule hierarchically by the following four criteria, presented in decreasing importance. The primary criterion is based on the total number of late orders which can not be produced on the current day d. As a secondary criterion the total lateness in days of the planned orders is to be minimized: For each order o_j that can not be completed within the given time frame, it results in a penalty of $(d - d_j + 1)^2$, i.e. the square of the order's lateness in days plus one. The third criterion is to minimize the number of jobs that are late and the fourth criterion is to maximize the total number of jobs produced within the time frame.

6.3 Solution approach

As described in Section 6.2, we only determine a production sequence for one day and only consider orders for which the supply of raw material is guaranteed. Still, the problem is too rich to be solved in acceptable time by exact methods. While the problem resembles a synchronous flow shop with two adjacent dominating machines, the first criterion to evaluate a schedule is a weighted sum of late orders. As we showed in Chapter 3, finding an optimal schedule is strongly $\mathcal{NP}$-hard even for $F2|\text{synmv}, \text{dom}(1) \mid \sum w_j U_j$. Additionally, even for other objective functions, solving synchronous flow shop problems is $\mathcal{NP}$-hard for two machines in the presence of cyclic changeovers even if they are constant. The gluing form resources in combination with the dependencies of jobs to gluing forms and stations (as depicted in Figure 6.2) further complicate the problem. Thus, the problem is also not applicable to the constructive heuristics described in Section 5.1. Instead, we choose an A-Team approach (see Section 5.3) to obtain heuristic solutions. As we described in Section 5.3 the A-Team is well capable of dealing with highly complex problems. In the following, we will give a short outline of the solution approach. The computational results for test instances that were supplied by the practitioner will be presented in Section 7.2.

A problem instance consists of orders $o_1, \ldots, o_{|O|}$ as well as a time frame that is to be scheduled on the current day d. Usually, the time frame spans 16 to 24 hours. Because the exact number of cycles that can be produced during the specified time frame depends on the schedule to be determined, the maximum number Γ of cycles to be scheduled is chosen such that it sufficiently covers the time frame. For example, this number can be chosen based on the mean insertion and gluing times of the products or based on historical data. On a typical day the practitioner is able to produce approximately 5000 shelf boards

(see Section 7.2 for real world production rates). We do not require the number of cycles to be large enough such that it is possible to completely schedule all orders up to their full volume. Given an assignment of orders and gluing forms to the cycles, the completion times for all jobs and orders can be calculated. If the completion times of all v_j jobs of an order o_j with due date $d_j \leq d$ are within the specified time frame, then the order is said to be produced on time (even if it is already late on that day), otherwise it is said to be late. An order is thus considered late either if it is not scheduled completely or if the completion time of an associated item is later than the time frame specified for the current day.

For our solution approach we use a broad range of improver agents, each of which alters its solution only slightly. In our early tests these rather simple agents provided better results than the usage of more sophisticated structures. We also tested an approach based on self-adaptive metaheuristics (as suggested in Meignan et al. (2010)), in which agents consist of multiple small operations and a second-order heuristic algorithm is used to determine which operations should be used and with which intensity. However, this did not yield better results than using the rather unsophisticated agents. Instead of assigning all jobs to the cycles individually, we try to schedule jobs of the same order in form of order fragments as defined above so that they are produced on the same station in direct succession. We do not require each order fragment to have size equal to the volume of its order, however the sum of sizes of order fragments of an order may not exceed the volume. In the following, we give a short description of the constructor and improver agents and their strategies. Note that each agent only constructs feasible solutions, i.e. the assignment and reassignment of order fragments and gluing forms is only performed if the constraints regarding the feasibility of gluing forms and stations are adhered to. Thus, whenever an order fragment is scheduled in a cycle, this is only done when the corresponding gluing form is available at this time and the gluing form is compatible with the station in which it is to be inserted in this cycle.

To create initial solutions we use the following constructors:

- Assign the first order fragment to start in cycle 1 at random and iteratively choose an order fragment for the next unassigned cycle such that the maximum of the processing times on machines M_1 of jobs and on machine M_2 of the already assigned jobs in this cycle is minimal. Let the size of each order fragment be as large as possible. If all orders consist of a single job, this heuristic resembles (S3) from Section 5.1.1

- Sort orders according to due dates and greedily assign each order fragment to the earliest feasible cycle according to this sorting.

- Randomly assign order fragments.

The assignment of order fragments resembles a variation of the constructors (C2) and (C3) defined in Section 5.3.3. The improvers consist of a couple of agents, each of which has its own neighborhood structure. The agents make one move within their neighborhood to change the current solution. For each agent we can choose whether it may be allowed to return only the best neighbor, any improving neighbor or an arbitrary neighbor. In most cases we allow the agents to return non-improving solutions as well. Note again that

moves within these neighborhoods are allowed only if the resulting assignment is feasible according to assignments of gluing forms and stations. As stated before, the number of cycles in general is not required to be large enough such that it is possible to completely schedule all orders up to their full volume. Thus, it is permitted to remove orders and order fragments from the plan such that not all jobs of an order are scheduled. We also allow that stations are not completely scheduled, i.e. that for some station s and some $c' \leq \Gamma$ no job is assigned to cycles $c \geq c'$ in which it were to be inserted at station s. We use the following operators:

- Remove an order fragment from the plan and move all succeeding order fragments on this station to the vacated cycles.
- Insert an order fragment into the plan and move all order fragments on this station that were previously occupying these cycles to later cycles (cut off at Γ if necessary).
- Reassign the gluing form of an order fragment.
- Split an order fragment into two.
- Shorten or lengthen an order fragment and move all succeeding order fragments accordingly.

Given these basic moves, improver agents alter solutions. Each improver in turn takes one solution at random and performs a series of the basic moves described above:

- Choose a late order o_j, i.e. an order which is not completed in the allowed time frame of the current solution. This order can either be chosen at random or orders with larger penalty values can be preferred. The solution is then altered via removal or shortening of order fragments and feasible insertions of order fragments containing o_j such that in the new solution the order o_j is completed in the allowed time frame.
- Choose an order fragment that is currently planned on a station and assign a succeeding order fragment on this station which uses the same gluing form.
- Fix a currently scheduled order fragment and reschedule order fragments in adjacent cycles such that the cycle times are minimized.
- Randomly swap or shift planned order fragments.
- Randomly insert or remove individual order fragments or all order fragments of an order.

To have a diverse population of solutions on which the agents can work on, three distinct pools of solutions are generated by the constructors. The agents work independently on all three solution pools and each time choose one solution at random to work on. The current best solutions of each pool are copied and saved separately not to be lost by manipulation of the improvers. After a number of steps T, the best solutions from each pool together with the saved solutions are copied into all three pools. All other solutions are removed

by the destroyers and the solution pools are refilled by the constructors. This process is done for a number of iterations. If during one iteration no solution within the three pools is found which is better than the best solution at the start of the iteration, only two of the solution pools are filled in the fashion described above while the third pool is filled completely with new solutions by the constructors. To "catch up" in quality with the other two solution pools, the agents only work on this solution pool before continuing their work on all three pools and continuing the iteration described above.

Chapter 7

Computational results

In this chapter we present computation results for the algorithms described in this thesis. For this, we evaluate the exact methods developed in Chapter 4 as well as the heuristic methods described in Chapter 5.

In Section 7.1 we evaluate the algorithms for the objective functions of minimizing the makespan, the maximum lateness and the total completion time on artificially generated test sets. In Section 7.2 we present computation results for the practical application discussed in Chapter 6. Therein, the results obtained by the multi agent approach described in Section 6.3 for data obtained by the practitioner are compared to the actual production.

7.1 Generated test sets

In this section we evaluate the algorithms developed in Chapters 4 and 5 on several different test sets which are based on the one created by Taillard (1993) for asynchronous flow shop problems. Up to this point, the test set by Taillard has been used by various authors to evaluate exact and heuristic methods for flow shop scheduling problems. While the instances have been created for asynchronous flow shops, we can still use them to evaluate the methods discussed in this thesis. The test set of Taillard contains ten instances for each of twelve distinct combinations of number of machines and jobs, ranging from 20 jobs on 5 machines to 500 jobs on 20 machines. For each instance, the processing times of jobs on the respective machines are within the interval $[0, 100]$.

To evaluate the algorithms designed for minimizing the makespan of synchronous flow shops without machine dominance as well as for minimizing the total completion time, we used the original test set of Taillard. To test algorithms for synchronous flow shops with dominating machines and algorithms for minimizing the maximum lateness, we altered the test set as described in the corresponding section. Additionally, we generated test sets with fewer jobs and machines to test exact algorithms for which the test sets described above proved to be too difficult.

This section is structured as follows: In 7.1.1 we present results for minimizing the makespan of synchronous flow shops. Subsection 7.1.2 deals with the minimization of maximum lateness while in 7.1.3 we consider the minimization of total completion time.

All subsections follow the same structure: First, we compare the performance of exact methods derived in Chapter 4 for smaller instances. Afterwards, we concentrate on the larger instances based on the test set by Taillard. For these, we first summarize the quality of the lower bounds developed in Section 4.4. Thereafter, we evaluate the heuristics approaches described in Chapter 5. The constructive heuristics of similar procedure are evaluated in comparison to another as well as in comparison to the best solution that could be obtained by any constructive heuristic; to which we will also refer to as the best constructive solution. Additionally, we summarize the performance of all constructive heuristics over the whole test set. Afterwards, the improvement heuristics are evaluated. At the end of each section, the performance of the heuristics in comparison to the exact algorithms as well as their average deviation from the best lower bound is summarized.

All computation experiments were conducted on an Intel Core i7-2600 @ 3.40 GHz CPU with 8 GB RAM. To evaluate the mixed integer programming formulations, we used the IBM ILOG CPLEX 12.5.0.0 IP solver. All runtimes in each table are in seconds and all average deviations are given in percentage. The runtime required to obtain a heuristic solution was below one second for all constructive heuristics and all instances except for the largest instances when using the NEHA heuristic. Therefore, we mention the runtime of the constructive heuristics just for cases in which it exceeds one second and only discuss the runtime of exact algorithms and improvement heuristics.

7.1.1 Makespan

In this section we evaluate the methods to minimize the makespan of synchronous flow shop problems. As discussed in Chapter 3, minimizing the makespan is $\mathcal{NP}$-hard in the general case with three or more machines and polynomially solvable in the presence of two adjacent dominating machines. If the two dominating machines are non-adjacent, the complexity status remains open. However, the special structure of these instances allows for more efficient solution algorithms. Therefore, we distinguish between these cases and present results separately for the general synchronous flow shop and for two dominating machines.

The general case

Even for the smallest instances of the test set of Taillard, no optimal solution could be found within 30 minutes by CPLEX using the basic mixed integer formulation (4.1)-(4.6) described in Chapter 4.2. Unfortunately, a branch and bound algorithm using the matching lower bound (LB-M) performed even worse. Therefore, we generated some smaller instances to compare the two algorithms. For each combination of number of jobs and machines depicted in Table 7.1, we generated 10 instances. For each instance, the processing times of all jobs were chosen in the interval $[0, 100]$ similar to the test set of Taillard. Table 7.1 depicts the number of instances solved by CPLEX and the branch and bound algorithm within a time limit of 30 minutes as well as the average computation time required for the instances for which an optimal solution could be obtained. It can be seen that already for 15 jobs on two machines, the branch and bound algorithm can not solve

all instances to optimality within the time limit. Further, for all problem sizes, CPLEX clearly outperforms the branch and bound algorithm in terms of computation time.

		# solved		avg. time				# solved		avg. time	
m	n	CPLEX	BB	CPLEX	BB	m	n	CPLEX	BB	CPLEX	BB
2	10	10	10	0.22	0.23	3	10	10	10	0.22	1.86
2	15	10	9	8.80	130.0	3	15	10	0	17.59	–
2	20	6	0	419.41	–	4	10	10	10	0.24	5.24
2	25	2	0	1.17	–	5	10	10	10	0.27	9.18

Table 7.1: Comparison of the number of instances solved to optimality within 30 minutes and the average computation time of solved instances

We compare four lower bounds for the test set of Taillard: LB-FS lists the optimal solution value or the best known lower bound for the asynchronous flow shop problem. The values were obtained from the homepage of Taillard (cf. Taillard (2005)), the last update occurred in April, 2005. LB-GG and LB-M illustrate the two constructive lower bounds described in Section 4.4.1, i.e. the iterative adjacent Gilmore-Gomory lower bound and the matching lower bound. Finally, LB-IP presents the best lower bound obtained by CPLEX within a time limit of 30 minutes. Table 7.2 shows the average deviation in percent from the best obtained lower bound for each of the combinations of jobs and machines (10 instances each). We also depict the average deviation from the best lower bound of the best feasible solution obtained by CPLEX within 30 minutes (UB-IP). It can be seen that the gap between the best lower and upper bound grows very large with increasing problem size. The IP lower bound outperforms the constructive lower bounds for most problem sizes and for the smallest instances it always achieves the best lower bound. However, the matching lower bound LB-M performs better in comparison to the other lower bounds for larger instances. For 500 jobs on 20 machines, CPLEX was able to derive a lower bound for only 9 out of the 10 instances. The matching lower bound outperforms the Gilmore-Gomory lower bound for all problem sizes, most notably for instances where the number of jobs is not much larger than the number of machines. Surprisingly, despite the differences between synchronous and classical flow shops, the flow shop lower bound LB-FS performs very similar to the other lower bounds and in comparison to the other lower bounds offers very good results with increasing number of machines.

In the following, we discuss results of heuristic approaches for minimizing the makespan of synchronous flow shops as described in Chapter 5. First, we investigated the effect of using different weights w for the formula

$$\sum_{k=i}^{i+m-2} \left(w \cdot \max\left(0, p_{1+k-i,j} - c_k\right) + (1-w) \cdot \max\left(0, c_k - p_{1+k-i,j}\right)\right)$$

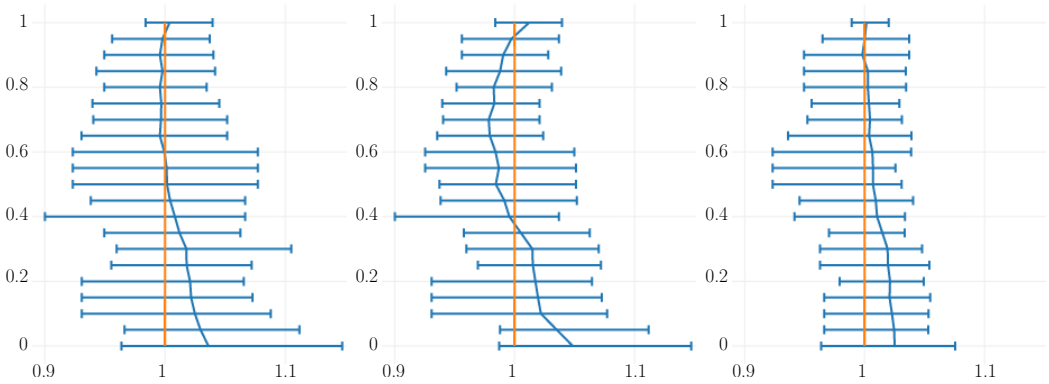
to evaluate the insertion of a job j at position i in heuristic S3W. For different weights $w = 0, 0.05, 0.1, \ldots, 1.0$, Figure 7.1 depicts the mean as well as the maximal relative deviation of the objective value of S3W from the objective value obtained by S3I.

		average deviation from best LB				
m	n	LB-FS	LB-GG	LB-M	LB-IP	UB-IP
5	20	12.14	18.66	10.62	0.00	4.87
5	50	2.99	2.98	0.00	2.34	17.04
5	100	2.14	1.89	0.00	2.22	21.55
10	20	16.73	48.62	19.29	0.00	8.29
10	50	1.69	9.92	2.59	0.06	38.86
10	100	0.55	3.64	0.52	0.12	46.91
10	200	0.75	2.24	0.08	0.67	66.42
20	20	12.02	105.29	7.53	0.00	15.21
20	50	0.39	26.75	6.13	0.48	44.85
20	100	0.33	13.08	5.14	0.53	69.9
20	200	0.03	5.37	1.85	0.45	79.05
20	500	0.27	1.96	0.12	0.15*	83.71

Table 7.2: Comparison of lower bounds on the makespan and upper bound obtained by CPLEX
*: No lower bound could be obtained for three instances

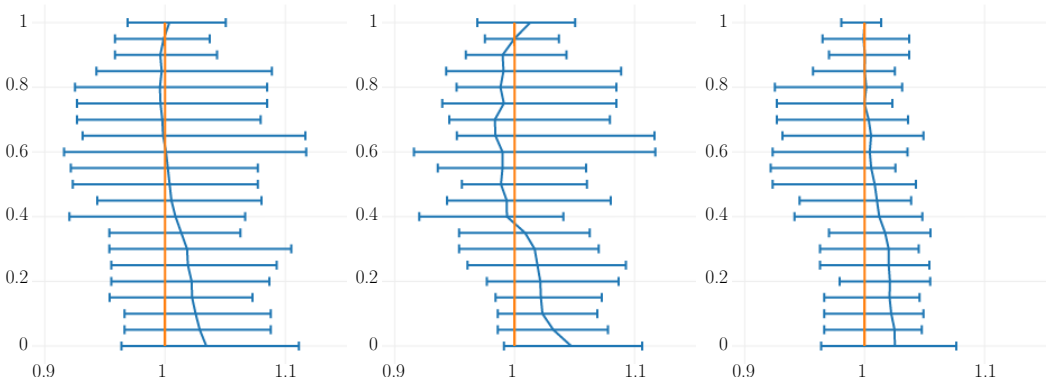
Figure 7.1 a) shows the effect of the weights over all instances of Taillard. It can be seen that for weights lower than 0.6, the mean objective value of S3W is larger than that of S3I. However, even for larger weights, the mean objective value is at most 0.5 percent lower than for S3I. Further, using weights can lead to larger improvements if the number of machines is smaller. Figures 7.1 b) and c) depict the effect of the weights for instances with five and twenty machines, respectively. It can be seen that for five machines, the mean objective value is improved by up to 1.6 percent for a weight of 0.6, while for twenty machines there is a decrease in solution quality for most weights. Figure 7.2 shows the same evaluation for using weights for S3DEW and their comparison to the objective value obtained with S3DE. The results are similar to those of S3W and S3I. In both cases, the best results over all instances are achieved by using a weight of 0.8.

Next, we compare the iterative constructive heuristics S1, S2, S3, S3I, S3DE as well as S3W and S3DEW with a weight of 0.8. Table 7.3 shows the results for these heuristics. For each set of 10 instances of the test set of Taillard we show how often each heuristic performs best in comparison to all 18 constructive heuristics discussed in this section as well as the average deviation in percent from the best result obtained by a constructive heuristic. It can be seen that heuristics S3I and S3DE clearly outperform the heuristics S1, S2 and S3 except for instances with 20 machines and 20 jobs for which heuristics S2 has the lowest average deviation from the best heuristic solution among the iterative constructive heuristics. Only for one instance was one of the original heuristics by Soylu et al. (2007) able to reach the best result of all constructive heuristics. Overall, the weighted variants S3W and S3DEW perform best. The non-weighted versions S3I and S3DE are better only for larger instances.



a) all instances b) 5 machines c) 20 machines

Figure 7.1: Effect of different weights for S3W in comparison to S3I for the instances of Taillard



a) all instances b) 5 machines c) 20 machines

Figure 7.2: Effect of different weights for S3DEW in comparison to S3DE for the instances of Taillard

Furthermore, we evaluate six variants of the NEH heuristics. In the first variant (Σ,s), the jobs are sorted by their total processing time and then iteratively inserted in the standard way as shown in Algorithm 5.4. The second variant (DE,s) employs the standard algorithm but uses the schedule obtained by the constructive heuristic S3DE as an initial sorting for the jobs. The third and fourth variant use a different evaluation function for the insertion of the jobs, similar to the weighted function used in heuristic S3W with a weight of $w = 0.8$. This is performed for both sorting methods, total processing time as well as presorting with the constructive heuristic S3DE (Σ,w and DE,w). Finally, we also test the heuristic NEHA where the insertion of all jobs is evaluated in each iteration of the NEH heuristic. This heuristic is used in combination with the standard evaluation

		# times best constructive solution							average deviation from best result						
m	n	S1	S2	S3	S3I	S3W	S3DE	S3DEW	S1	S2	S3	S3I	S3W	S3DE	S3DEW
5	20	0	0	0	2	2	3	2	17.47	11.25	7.44	3.02	1.77	2.67	2.88
5	50	0	0	0	2	6	2	7	22.55	8.77	7.04	2.45	0.99	2.14	0.41
5	100	0	0	0	0	7	0	5	28.51	14.04	7.87	3.08	0.28	2.66	0.46
10	20	0	0	0	2	1	2	1	13.78	9.98	8.68	5.14	4.54	4.73	3.25
10	50	0	0	0	2	1	1	3	13.46	6.86	4.79	2.29	2.11	2.68	1.81
10	100	0	0	0	3	2	3	2	16.22	7.96	4.29	1.08	0.57	0.80	0.64
10	200	0	0	0	3	4	4	4	19.70	10.16	4.29	0.78	0.70	0.46	0.55
20	20	1	0	0	0	1	0	1	8.62	4.91	7.11	5.69	5.15	5.35	3.97
20	50	0	0	0	0	0	0	0	10.02	4.45	6.63	3.80	4.38	3.53	4.51
20	100	0	0	0	0	0	0	0	9.95	4.58	4.55	2.57	2.66	2.49	2.43
20	200	0	0	0	3	0	4	1	9.13	4.07	1.98	0.49	1.14	0.51	0.92
20	500	0	0	0	3	0	8	0	11.24	5.88	1.86	0.41	0.96	0.04	0.70

Table 7.3: Comparison of iterative constructive heuristics for minimizing the makespan

function as well as the weighted function (s and w). The results are depicted in Table 7.4. It can be seen the incorporation of a weighted evaluation function instead of the standard evaluation does in general not lead to inferior results. Using the result obtained by S3DE as a starting solution seems more beneficial for the variant employing the weighted evaluation function. For the standard evaluation function, it is not definitive which initial sorting of the jobs leads to better results. The NEHA heuristic clearly outperforms the standard NEH heuristic. However, for the largest problem size containing 500 jobs and 20 machines, NEHA already requires a computation time of 27 seconds while all other heuristics need at most 0.3 seconds. Further, it can be seen that the NEH heuristics outperform the iterative constructive heuristics when the number of jobs is not too large in relation to the number of machines. Especially, for problem sets with 20 machines, NEH performs very well. For the largest problem sets, all NEH variants are inferior to S3I and S3DE.

Finally, we evaluate constructive heuristics that transform the problem into either an instance of a two-machine synchronous flow shop or a traveling salesman problem, i.e. the heuristics GG,CDS and TSP. We try the following three cost functions for the TSP. For two jobs i, j and $k = 1, 2, 3$, cost functions $c_k(i, j)$ calculate the cost when job j is directly succeeding job i:

$$c_1(i,j) = \sum_{k=2}^{m} \max\left(0, p_{k-1,j} - p_{ki}\right) + p_{mj}$$

$$c_2(i,j) = \sum_{k=2}^{m} \left(0.8 \max\left(0, p_{k-1,j} - p_{ki} + 0.2 \max(0, p_{ki} - p_{k-1,j})\right)\right) + 0.8 p_{mj}$$

$$c_3(i,j) = \sum_{k=2}^{m} |p_{k-1,j} - p_{ki}| + p_{mj}$$

		# times best constructive solution						average deviation from best result					
		NEH				NEHA		NEH				NEHA	
m	n	$\sum$,s	DE,s	$\sum$,w	DE,w	s	w	$\sum$,s	DE,s	$\sum$,w	DE,w	s	w
5	20	1	0	0	0	4	0	4.56	6.14	6.27	6.66	2.72	14.93
5	50	0	0	0	0	0	0	6.27	4.68	6.81	4.67	2.49	17.27
5	100	0	0	0	0	0	0	8.31	7.55	8.75	7.52	4.56	22.70
10	20	1	2	0	0	3	0	3.19	2.99	5.09	4.48	2.02	11.18
10	50	1	1	0	0	3	0	2.91	3.81	4.42	4.30	0.98	11.43
10	100	0	0	0	0	3	0	4.53	5.47	5.97	6.02	1.03	12.87
10	200	0	0	0	0	0	0	7.93	7.24	9.67	8.21	4.34	16.94
20	20	1	3	3	0	2	0	2.30	2.35	2.65	2.55	1.51	6.67
20	50	1	0	0	1	8	0	1.77	2.74	2.71	3.18	0.08	6.45
20	100	0	0	0	0	10	0	2.70	3.00	4.54	3.62	0.00	7.35
20	200	0	0	0	0	4	0	3.25	3.31	4.42	4.00	0.31	7.71
20	500	0	0	0	0	0	0	5.67	6.08	6.62	5.99	3.00	9.84

Table 7.4: Comparison of NEH variants for minimizing the makespan

Similar to S3, the first cost function $c_1(i, j)$ calculates the increase in cycle time that incurs when job j is scheduled after job i. Cost function $c_2(i, j)$ also takes into account the idle time that is caused by job i, similar to S3W with a weight of 0.8. Finally, cost function $c_3(i, j)$ sums up the idle time that occurs for any of the two jobs when job j is scheduled after i. Table 7.5 shows an evaluation of the transformation heuristics. Again we show how often each heuristic performs best in comparison to the other constructive heuristics. It can be seen that only for instances with a large number of machines and not too many jobs in relation to the number of machines (20 machines and 20 or 50 jobs, respectively), the GG heuristic performs better than the transformation into a traveling salesman problem. In general, using the cost function c_1 outperforms the two other cost functions and for one instance it even reaches the best solution of all constructive heuristics. However, in general all transformation heuristics perform worse than the best iterative constructive heuristics and all variants of NEH except for the weighted variant of NEHA.

Figure 7.3 depicts how often each constructive heuristic reaches the best constructive solution as well as how often it is the only heuristic to reach a best solution. Further, Figure 7.4 shows the average deviation in percent of each constructive heuristic from the best constructive solution. It can be seen that regular NEHA using the standard evaluation function for inserting jobs, reaches the best heuristic solution most often. Also, the other NEH heuristics except for the weighted NEHA variant, are able to find the best heuristic solution in at least one case. There is no clearly superior variant, they only vary slightly in their average deviation with the variants employing the non-weighted evaluation function being slightly better. As discussed above, the NEH heuristics perform best for smaller problem sizes while the iterative heuristics S3I and S3DE as well as their weighted counterparts perform better for larger instances. Similar to the NEH variants, no distinct winner can be determined among these four variants. Overall, the iterative heuristics S3I

and S3DE as well as the NEHA heuristic with the regular evaluation function achieve the best results with an average deviation of 1.88 % for the weighted version of S3DE and 1.92 % for the NEHA heuristic. Heuristics S1 and TSP with the cost function c_1 only perform best for a single instance, respectively. However, their average deviation from the best constructive solution is very large. Overall, the transformation heuristics and the heuristics S1,S2 and S3 perform very bad in comparison to the other heuristics.

		# times best constructive solution					average deviation from best result				
				TSP					TSP		
m	n	GG	CDS	c_1	c_2	c_3	GG	CDS	c_1	c_2	c_3
5	20	0	0	1	0	0	8.86	10.37	4.38	7.51	6.95
5	50	0	0	0	0	0	15.74	18.24	5.98	7.98	9.40
5	100	0	0	0	0	0	20.94	24.69	9.01	10.57	10.76
10	20	0	0	0	0	0	6.44	6.84	5.99	6.32	7.13
10	50	0	0	0	0	0	10.20	10.47	5.88	7.68	5.91
10	100	0	0	0	0	0	13.16	14.17	5.66	6.68	6.99
10	200	0	0	0	0	0	17.63	18.47	8.41	8.85	8.97
20	20	0	0	0	0	0	2.88	3.79	5.10	5.34	5.22
20	50	0	0	0	0	0	6.09	7.02	5.58	6.46	7.73
20	100	0	0	0	0	0	7.31	7.44	6.68	7.20	7.65
20	200	0	0	0	0	0	7.64	7.84	5.96	6.38	6.42
20	500	0	0	0	0	0	10.26	10.54	7.83	8.13	8.12

Table 7.5: Comparison of transformation heuristics for minimizing the makespan

In the following, we will discuss improvement heuristics for minimizing the makespan of synchronous flow shops. First, we consider improvement heuristics that mostly rely on adjacent pairwise interchange, i.e. the heuristics (I2) as well as variants of RACE which iteratively apply the heuristics RACS, I1 and SU as long as an improvement can be achieved. To evaluate the relative performance of the improvement heuristics we applied them to the same starting solution and compared the average improvement. Table 7.6 shows how often each improvement heuristic achieves the best improvement as well as the average improvement in percent of the objective value when using the result of S3DE as a starting solution. As can be seen, all heuristics perform very similar. I2 on average achieves the best results, followed by the iterative execution of the SU heuristic. For different starting solutions, the relative performance of the improvement heuristic was similar.

Next, we evaluate the metaheuristics when applying the swap and shift neighborhoods. For both neighborhoods we apply a tabu search, simulated annealing and an iterative improvement procedure. In each iteration we evaluate the neighborhood of the current solution given by a sequence $\sigma = (\sigma_1, \ldots, \sigma_n)$. Within iterative improvement, we start with evaluating the swaps of the job σ_1 with all other jobs (equivalently for the shift operation we evaluate shifts to all other positions). If any of the swaps (shifts) improve the objective value, we execute a move which improves the objective value the most and continue with

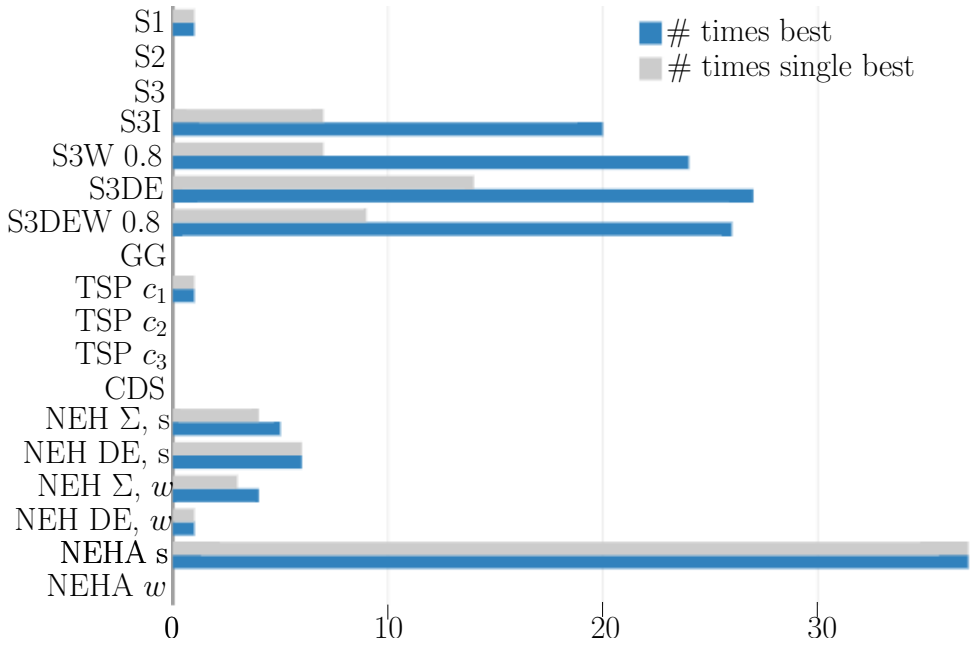


Figure 7.3: Comparison of constructive heuristics for minimizing the makespan

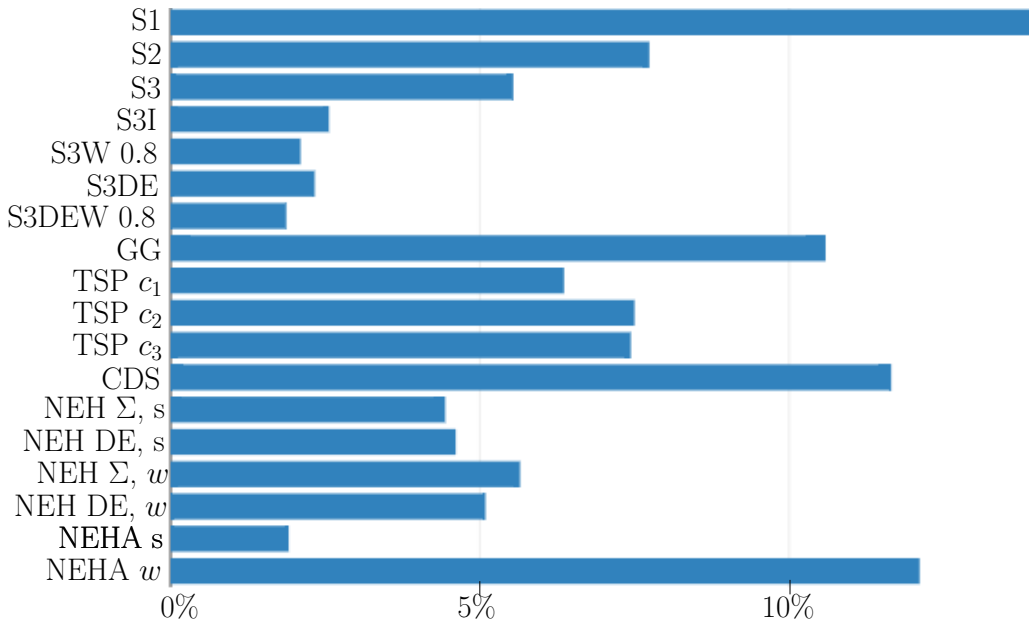


Figure 7.4: Average deviation of constructive heuristics from best obtained solution for minimizing the makespan

		# times best				average improvement of starting solution			
			RACE			I2	RACE		
m	n	I2	RACS	I1	SU		RACS	I1	SU
5	20	8	6	9	10	1.11	0.80	1.12	1.46
5	50	6	5	4	7	0.91	0.45	0.82	0.94
5	100	8	6	5	9	0.41	0.34	0.30	0.44
10	20	8	5	4	7	1.78	1.22	1.07	1.76
10	50	8	5	6	7	1.00	0.62	0.93	1.02
10	100	8	9	8	8	0.24	0.20	0.16	0.17
10	200	10	9	4	7	0.25	0.20	0.11	0.17
20	20	7	4	5	7	1.82	1.43	1.73	1.57
20	50	5	7	0	2	0.78	0.88	0.49	0.57
20	100	6	5	5	5	0.52	0.48	0.44	0.44
20	200	9	5	5	6	0.23	0.21	0.18	0.18
20	500	10	7	5	8	0.09	0.07	0.07	0.08

Table 7.6: Comparison of improvement heuristics for minimizing the makespan

the next iteration of the iterative improvement procedure. If no move would lead to an improvement, we evaluate all swap (shift) moves of job σ_2 and iteratively continue with jobs σ_i for all $i = 1, \ldots, n$ until we evaluated all possible moves. If for any i there exists a move which improves the objective value, we execute a move which improves the objective value the most and continue with the next iteration of the iterative improvement procedure. If no improvement can be found in the whole neighborhood, the iterative improvement procedure terminates.

In the tabu search, we define the following tabu criterion for the swap neighborhood: For a sequence $\sigma = (\sigma_1, \ldots, \sigma_n)$, when swapping two jobs in positions i, j with $i < j$ to obtain a sequence $\sigma' = (\sigma'_1, \ldots, \sigma'_n)$ we add the pairs $(\sigma'_i, \sigma'_{i+1})$ and $(\sigma'_{j-1}, \sigma'_j)$. The pairs denote the new successor of σ_j and the new predecessor of job σ_i after the move. A swap is tabu, when (σ_j, σ_{i+1}) or (σ_{j-1}, σ_i) is in the tabu list. Similarly, for the shift neighborhood we define the following tabu criterion: When shifting the job in position i to position j to obtain a sequence $\sigma' = (\sigma'_1, \ldots, \sigma'_n)$ we add the pair $(\sigma'_{j-1}, \sigma'_j)$ if $i < j$ and $(\sigma'_j, \sigma'_{j+1})$ if $i > j$. Again, the pairs denote the new predecessor of job σ_i in case of a shift to the right or a the new sucessor of job σ_i in case of a swap to the left. A shift is tabu, when (σ_j, σ_i) (if $i < j$) or (σ_i, σ_j) (if $i > j$) is in the tabu list. We use the successor for a shift to the left and a predecessor for a shift to the right since these always exist, while when shifting a job to the start (or end) of the schedule, it does not have a predecessor (successor). The same holds for swap moves. This tabu criterion does not forbid cycles as a solution can be visited again. However, in this case another move has to be executed than at the first time the solution was reached. Similar to the iterative improvement algorithm, we evaluate the swaps of job σ_1 with all jobs (equivalently for the shift operation we evaluate shifts to all other positions). If no non-tabu move leads to an improved objective value, we continue

with the other positions. Otherwise, we execute a best non-tabu move. If no non-tabu move that improves the current objective value can be found in the whole neighborhood, we choose the non-tabu move which decreases the objective value the least. We apply an aspiration criterion, i.e. whenever we find a move that would lead to a solution with a better objective value than the current best solution, we execute the move even if it is tabu. We use a tabu list of size 100 and stop the tabu search either after 100 iterations in which we found no new best solution or after 30 minutes.

For the simulated annealing approach, Brockmeyer (2014) evaluated several annealing functions. For each iteration t, we make a neighborhood move from a solution with objective value o^{old} to a solution with objective value o^{new} with probability

$$\max\left(1, \exp\left(\frac{\left(o^{\text{new}} - o^{\text{old}}\right)}{50\frac{cos\,\frac{t}{30}}{t}}\right)\right).$$

We stop the simulated annealing algorithm if we did not find a new best solution for 10.000 iterations or after 30 minutes.

Table 7.7 summarizes the results of the metaheuristics when using the swap and shift neighborhoods with the described attributes. Analogous to the other improvement heuristics, we compare the improvement achieved when using the result of S3DE as a starting solution. We depict the average improvement in percent achieved by the metaheuristics in comparison to the starting solution. Further, we show the average computation time in seconds. It can be seen that for smaller instances, all heuristics have a very low runtime while achieving better improvements in comparison to the improvement heuristics depicted in Table 7.6. For all heuristics and all problem sizes, using the swap neighborhood clearly outperforms using the shift neighborhood. In general, all heuristics using the same neighborhood achieve a similar improvement of the constructive solution. While the tabu search leads to slightly better results than simulated annealing and iterative improvement, the latter two require a much lower runtime. Again, the metaheuristics perform similar for different starting solutions.

Finally, we compare the heuristics to the results obtained by using CPLEX with the mixed integer programming formulation. Further, we determine the gaps between the best obtained solutions and the best lower bounds for the instances of Taillard. Table 7.8 shows how often the best solution obtained by a constructive heuristic was better than the upper bound obtained by CPLEX within 30 minutes. The same is depicted for the best solution obtained by an improvement heuristic. For this, the improvement heuristics were executed on various starting solutions and for each instance we use the best result achieved over all runs. While CPLEX outperforms the heuristics for smaller instances, for larger instances even the constructive heuristics lead to better results. Unfortunately, the gaps are very large for all problem sizes. This seems to be mostly caused by the overall bad quality of the lower bounds on the makespan for the synchronous flow shop.

		avg. impr. of starting solution						average computation time					
		Tabu		SA		II		Tabu		SA		II	
m	n	sw	sh	sw	sh	sw	sh	sw	sh	sw	sh	sw	sh
5	20	5.87	3.94	5.85	3.39	5.29	3.53	0.0	0.0	0.0	0.0	0.0	0.0
5	50	4.07	2.43	4.08	2.12	3.73	2.04	0.1	0.2	0.0	0.0	0.0	0.0
5	100	2.01	0.83	1.94	0.69	1.90	0.79	1.1	1.6	0.0	0.0	0.0	0.0
10	20	8.05	5.43	7.80	5.20	7.08	4.88	0.0	0.0	0.0	0.0	0.0	0.0
10	50	6.12	2.89	5.97	3.01	5.48	2.60	0.3	0.4	0.0	0.0	0.0	0.0
10	100	3.39	1.27	3.20	1.36	3.12	1.16	2.4	2.9	0.0	0.0	0.0	0.0
10	200	2.13	0.62	1.98	0.59	1.99	0.61	21.6	21.3	0.1	0.0	0.1	0.1
20	20	2.41	2.41	2.43	2.43	1.87	1.87	0.0	0.1	0.0	0.0	0.0	0.0
20	50	7.91	2.49	7.57	3.03	6.42	2.61	0.7	1.0	0.0	0.0	0.0	0.0
20	100	4.68	1.85	4.61	1.91	4.44	1.79	6.2	6.5	0.1	0.1	0.0	0.1
20	200	3.43	1.27	3.28	1.24	3.27	1.29	44.4	45.1	0.2	0.1	0.3	0.3
20	500	1.42	0.31	1.16	0.16	1.36	0.27	1090.3	635.5	0.6	0.1	1.6	2.9

Table 7.7: Comparison of metaheuristics employing the swap (sw) and shift (sh) neighborhoods for minimizing the makespan

		# times better than CPLEX		average gap from best LB		
m	n	constr.	impr.	constr.	impr.	CPLEX
5	20	0	0	14.59 %	7.03 %	4.87 %
5	50	0	2	24.86 %	17.79 %	17.04 %
5	100	3	10	23.27 %	18.52 %	21.55 %
10	20	0	1	17.62 %	9.71 %	8.29 %
10	50	0	6	45.88 %	36.83 %	36.86 %
10	100	5	10	46.10 %	37.46 %	46.91 %
10	200	10	10	44.66 %	38.43 %	66.42 %
20	20	0	0	22.96 %	16.78 %	15.21 %
20	50	0	7	52.89 %	44.16 %	44.58 %
20	100	10	10	58.63 %	50.15 %	69.90 %
20	200	10	10	66.06 %	56.51 %	79.05 %
20	500	10	10	66.04 %	58.25 %	83.71 %

Table 7.8: Number of times the constructive and improvement heuristics perform better than CPLEX, and gaps for minimizing the makespan

Two dominating machines

To evaluate the algorithms for minimizing the makespan of synchronous flow shops with two dominating machines, we slightly alter the instances by Taillard (1993) by setting the processing times of all jobs on all machines to zero except for the first and last machine. We include all combinations of number of machines and jobs except for the instances with 20 machines and 20 jobs as for them each subsequence of a resulting schedule would contain at most two jobs.

In Section 4.2.2 we presented a mixed integer linear program (4.11)-(4.21) for the synchronous flow shop problem with two dominating machines. Table 7.9 shows the performance of CPLEX on the generated instances when using the basic mixed integer formulation (4.1)-(4.6) (denoted as IP-I) as well as the formulation specialized on two dominating machines (denoted as IP-II) for a time limit of 30 minutes. For each combination of m and n, we show how many of the 10 instances CPLEX was able to solve to optimality using the respective IP formulation. Further, for both formulations we depict the average deviation in percent from the best feasible solution obtained using the two formulations. Note that for larger instances with 20 machines, CPLEX was able to find a feasible solution when using formulation (4.11)-(4.21) in only 9 cases for 200 jobs or even no case at all for 500 jobs. Further, we also depict the average computation time required in case an instance was solved to optimality. The results show that using the specialized formulation (4.11)-(4.21) is clearly superior to using the standard formulation (4.1)-(4.6) for smaller problem sizes, solving a lot more instances in very short time. However, for larger sizes it becomes difficult to even obtain feasible solutions. While we were not able to solve even a single instance to optimality within 30 minutes in the general case, 78 out of the 110 instances can be solved by using the specialized formulation in the case of two dominating machines.

		# solved		avg. dev. fr. best UB		avg. time	
m	n	IP-I	IP-II	IP-I	IP-II	IP-I	IP-II
5	20	10	10	0.00	0.00	17.58	3.79
5	50	0	10	0.07	0.00	–	1.55
5	100	0	10	0.39	0.06	–	501.38
10	20	10	10	0.00	0.00	5.35	0.10
10	50	1	8	0.34	0.00	8.62	11.08
10	100	0	10	0.37	0.00	–	94.98
10	200	0	0	0.68	2.70	–	–
20	50	3	10	0.00	0.00	61.10	2.01
20	100	0	9	0.36	0.00	–	94.38
20	200	0	0	0.19	3.94*	–	–
20	500	0	0	0.00	–**	–	–

Table 7.9: Comparison for the two MIP formulations for two dominating machines
*: Only 9 feasible solutions were obtained
**: No feasible solution was obtained for any of the ten instances

Table 7.10 depicts the average deviation of various lower bounds from the best obtained lower bound. The lower bound reached within 30 minutes using formulation (4.1)-(4.6) is denoted by LB IP-I, the lower bound obtained within 30 minutes with formulation (4.11)-(4.21) is denoted by LB IP-II. Furthermore, the Gilmore-Gomory lower bound (LB-GG) as well as the matching lower bound (LB-M) are presented. The lower bound obtained by the formulation (4.11)-(4.21) outperforms all other lower bounds and delivers the best results for all instances. The Gilmore-Gomory lower bound performs very similar to the matching lower bound. For smaller problem sizes, the two constructive lower bounds lead to very poor results but the performance increases for larger instances. The table shows the average deviation from the best lower bound as well as the average deviation of the best upper bound obtained by CPLEX using either of the two mixed inter programming formulations (UB-IP) from the best obtained lower bound. It can be seen that the gaps are a lot smaller than in the general case without machine dominance.

		average deviation from best LB				
m	n	LB-GG	LB-M	LB IP-I	LB IP-II	UB-IP
5	20	4.81	4.83	0.00	0.00	0.00
5	50	0.54	0.55	1.81	0.00	0.00
5	100	0.16	0.16	1.23	0.00	0.03
10	20	20.82	20.94	0.00	0.00	0.00
10	50	2.84	2.84	1.08	0.00	0.03
10	100	0.81	0.81	0.49	0.00	0.00
10	200	0.22	0.22	0.77	0.00	1.68
20	50	14.09	14.09	0.30	0.00	0.00
20	100	3.49	3.49	0.63	0.00	0.02
20	200	0.98	0.98	0.71	0.00	1.75
20	500	0.17	0.17	0.67	0.00	3.50

Table 7.10: Comparison of lower bounds on the makespan for two dominating machines

In the following, we evaluate the heuristics discussed in Chapter 5 applied to these test instances. First, we start by comparing the constructive heuristics S1, S2, S3, S3I and S3DE as well as weighted variants of S3 and S3DE with a weight of $w = 0.8$. Table 7.11 summarizes the results, showing for each heuristic how often it obtains the best result of all constructive heuristics and the average deviation from the best objective value obtained by a constructive heuristic. The results are very similar to the case without machine dominance. Again, S3DE leads to the best overall results which can be slightly improved by adding a weight to the evaluation function. The S3DE heuristics perform relatively better compared to the S3I heuristics than in the general case without machine dominance where both heuristics achieved more similar results. Further, the weighted variant of S3DE performs even better than in the general case.

Table 7.12 summarizes the results for the variants of the NEH heuristic that were also used for the instances without machine dominance. Analogous to the case without

		# times best constructive solution							average deviation from best result						
m	n	S1	S2	S3	S3I	S3W	S3DE	S3DEW	S1	S2	S3	S3I	S3W	S3DE	S3DEW
5	20	0	0	0	1	3	2	6	19.00	16.70	14.40	5.65	3.40	2.12	1.44
5	50	0	0	0	0	1	0	7	24.49	19.08	12.47	5.33	3.50	2.46	0.49
5	100	0	0	0	0	0	2	7	27.38	21.48	11.11	2.44	2.03	0.71	0.45
10	20	0	3	0	1	1	6	6	10.50	5.81	15.96	5.61	4.63	1.49	1.49
10	50	0	0	0	0	0	4	9	20.03	16.32	16.66	5.81	6.12	0.58	0.09
10	100	0	0	0	0	0	3	7	23.02	20.76	13.24	4.15	4.13	0.79	0.27
10	200	0	0	0	0	1	0	9	26.95	24.33	10.25	2.52	2.08	0.32	0.04
20	50	0	0	0	0	0	6	9	13.17	8.58	18.47	4.75	4.85	0.18	0.00
20	100	0	0	0	0	0	5	5	20.11	16.62	17.51	6.47	6.09	0.30	0.45
20	200	0	0	0	0	0	2	8	25.16	22.49	12.82	5.70	5.60	0.75	0.12
20	500	0	0	0	0	0	1	9	29.59	28.00	10.02	2.94	2.70	0.28	0.02

Table 7.11: Comparison of the iterative constructive heuristics for two dominating machines

		# times best constructive solution						average deviation from best result					
		NEH				NEHA		NEH				NEHA	
m	n	$\sum$,s	DE,s	$\sum$,w	DE,w	s	w	$\sum$,s	DE,s	$\sum$,w	DE,w	s	w
5	20	0	0	0	0	0	0	13.87	11.72	18.09	15.99	14.43	26.80
5	50	0	0	0	0	0	0	9.96	11.62	15.06	16.06	9.04	29.28
5	100	0	0	0	0	0	0	9.01	10.09	11.87	13.62	7.51	24.55
10	20	0	0	0	0	0	0	17.29	9.44	20.30	13.08	17.25	26.76
10	50	0	0	0	0	0	0	14.92	10.71	26.02	18.70	13.88	34.57
10	100	0	0	0	0	0	0	14.39	12.38	25.68	19.86	10.97	35.42
10	200	0	0	0	0	0	0	12.82	11.73	23.13	19.05	9.93	34.37
20	50	0	0	0	0	0	0	20.60	5.80	27.34	19.71	20.50	31.11
20	100	0	0	0	0	0	0	21.75	11.75	32.85	22.12	20.53	37.95
20	200	0	0	0	0	0	0	19.50	17.22	32.67	21.72	16.26	39.22
20	500	0	0	0	0	0	0	16.54	15.09	32.09	20.92	13.06	38.48

Table 7.12: Comparison of NEH heuristics for two dominating machines

machine dominance, the non-weighted versions clearly outperform their weighted counterparts. However, in difference to the general case, all NEH variants perform much worse than the iterative constructive heuristics S3I and S3DE. None of the NEH heuristics was able to reach the best constructive solution for any instance. All variants required less than one second for all instances except for the largest set containing 500 jobs on 20 machines for which NEHA needed 27 seconds, analogous to the general case.

Next, we evaluate the performance of the transformation heuristics. Table 7.13 summarizes the results for each of the heuristic GG, CDS as well as the TSP heuristic with the three cost functions c_1,c_2 and c_3 as defined for the general case without dominating machines. The Gilmore-Gomory heuristics outperforms all other transformation heuristics

and achieves the best results of all transformation heuristics. Further, the Gilmore-Gomory heuristic is the only transformation heuristic to perform well compared to the best iterative construction heuristics. However, it is still outperformed by S3DE and its weighted counterpart. The solutions obtained by all other transformations deviate a lot from the best solutions.

		# times best constr. sol.					average deviation from best result				
				TSP					TSP		
m	n	GG	CDS	c_1	c_2	c_3	GG	CDS	c_1	c_2	c_3
5	20	2	0	0	0	0	4.05	21.78	20.51	19.35	23.33
5	50	2	0	0	0	0	2.43	27.03	29.35	23.89	26.17
5	100	2	0	0	0	0	0.94	27.35	29.15	26.32	28.48
10	20	1	0	0	0	0	5.04	14.23	15.53	13.17	15.62
10	50	0	0	0	0	0	5.23	25.80	24.65	24.37	23.68
10	100	0	0	0	0	0	3.48	28.37	27.30	27.41	28.40
10	200	0	0	0	0	0	1.72	29.09	28.76	28.92	28.61
20	50	0	0	0	0	0	6.88	19.46	20.45	20.13	20.46
20	100	0	0	0	0	0	7.02	26.68	27.26	26.77	26.77
20	200	0	0	0	0	0	5.05	29.66	30.43	29.32	29.54
20	500	0	0	0	0	0	2.14	30.56	31.38	30.95	31.35

Table 7.13: Comparison of transformation heuristics for two dominating machines

Finally, as we did for the general case, Figure 7.5 depicts how often each constructive heuristic reaches the best constructive solution as well as how often it is the only heuristic to reach a best solution. Figure 7.6 shows the average deviation of each constructive heuristic from the best constructive solution. The result is very different from the case with no machine dominance. The NEH heuristics perform very poorly for the test sets with two dominating machines. While the NEHA heuristic only had an average deviation of 1.88 % in the former case, it is 13.11 % for two dominating machines. Further, for no instance was any NEH heuristic able to find the best heuristic solution. In comparison, the S3DE heuristic and its weighted counterpart perform even better than in the general case. At least one of the two heuristics finds the best heuristic solution for 98 of the 110 test instances. Also, the average deviation from the best heuristic result is only 0.41 % for the weighted version and 0.83 % for the non-weighted version. The S3I heuristics perform slightly worse than the S3DE heuristics but still only with a small seviation. The GG heuristic also performs well. While the S1 and S2 heuristic have a very large deviation from the best objective value for almost all problem sizes, S2 manages to deliver the best solution in 3 instances with 20 jobs on 10 machines.

In the following, we discuss the effect of the improvement heuristics for the test set based on Taillard with two dominating machines. As the metaheuristics based on the api neighborhood were clearly outperformed by the metaheuristics based on the swap and shift neighborhood, we only evaluate the tabu search, simulated annealing approach and iterative improvement procedure with these two neighborhoods. As parameters for these

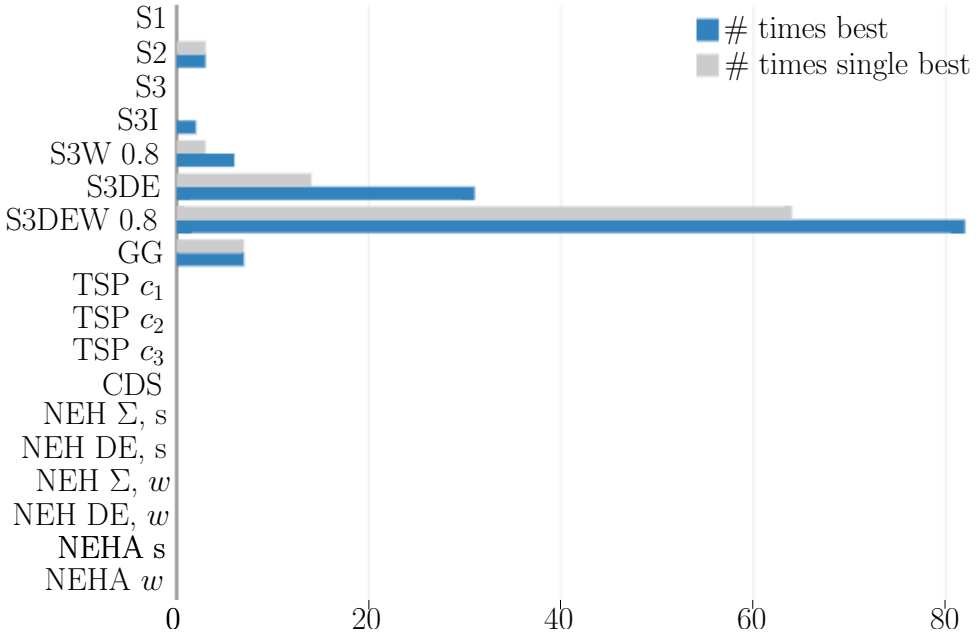


Figure 7.5: Comparison of constructive heuristics for two dominating machines

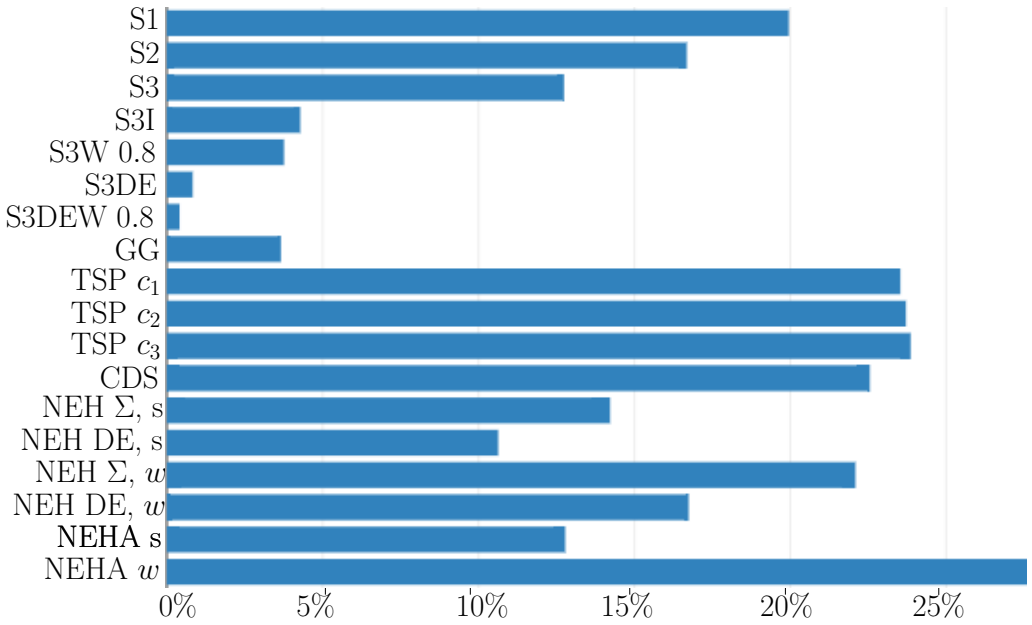


Figure 7.6: Average deviation of constructive heuristics from best obtained solution for two dominating machines

metaheuristics we used the same as in the general case. As a starting solution we used the solution obtained by the weighted S3DE heuristic. For each heuristic we depict the average improvement in percent from the starting solution as well as the average runtime in seconds. Similar to the general case, the swap neighborhood performs better than the shift neighborhood. The results are analogous to the problem without machine dominance. All metaheuristics perform very similar for all problem sizes. The tabu search performs slightly better than the simulated annealing and iterative improvement procedures but requires a longer computation time, up to 975 seconds on average for the largest instances.

		avg. impr. of starting solution						average computation time					
		Tabu		SA		II		Tabu		SA		II	
m	n	sw	sh	sw	sh	sw	sh	sw	sh	sw	sh	sw	sh
5	20	5.87	3.94	5.85	3.39	5.29	3.53	0.0	0.0	0.0	0.0	0.0	0.0
5	50	4.07	2.43	4.08	2.12	3.73	2.04	0.1	0.2	0.0	0.0	0.0	0.0
5	100	2.01	0.83	1.94	0.69	1.90	0.79	1.1	1.5	0.0	0.0	0.0	0.0
10	20	8.05	5.43	7.80	5.20	7.08	4.88	0.0	0.0	0.0	0.0	0.0	0.0
10	50	6.12	2.89	5.97	3.01	5.48	2.60	0.3	0.5	0.0	0.0	0.0	0.0
10	100	3.39	1.27	3.20	1.36	3.12	1.16	2.6	3.2	0.0	0.0	0.0	0.0
10	200	2.13	0.62	1.98	0.59	1.99	0.61	17.2	21.4	0.1	0.0	0.0	0.1
20	50	7.91	2.49	7.57	3.03	6.42	2.61	0.9	0.9	0.0	0.0	0.0	0.0
20	100	4.68	1.85	4.61	1.91	4.44	1.79	4.7	7.1	0.1	0.0	0.0	0.1
20	200	3.43	1.27	3.28	1.24	3.27	1.29	48.6	50.5	0.2	0.1	0.2	0.3
20	500	1.42	0.31	1.16	0.16	1.36	0.27	975.0	586.5	0.3	0.1	0.9	1.8

Table 7.14: Comparison of improvement heuristics for two dominating machines

In addition to these heuristics we also evaluate the tabu search approach T2D that was developed especially for synchronous flow shops with two dominating machines in Section 5.2.2. In Table 7.15, we show the average improvement of the best constructive solution achieved by T2D in comparison to the metaheuristics that are not specialized on problems with two dominating machines. T2D performs slightly better than the other heuristics for instances with only 5 machines. Additionally, for the largest instances with 20 machines and 500 jobs it delivers the best results while requiring a much lower runtime than the standard tabu search. Only for instances with a large number of machines and few jobs in comparison to the number of machines, the general metaheuristics perform slightly better.

Table 7.16 shows how well the constructive, the standard improvement heuristics and the specialized tabu search perform in comparison to CPLEX using the mixed integer formulations within a time limit of 30 minutes as well as the average gaps. The table depicts how often the optimality of the best solution could be verified, i.e. how often it was equal to the best lower bound. As can be seen, CPLEX finds an optimal solutions most often, especially for smaller instances. For one instance with 5 machines and 100 jobs for which CPLEX was not able to find an optimal solution within the time time limit, an optimal solution was found by T2D. Also, either T2D or the standard tabu search were able to find optimal solutions for three instances of 10 machines and 200 jobs for

		avg. impr. of best constr. sol.		
m	n	T2D	best general	runtime T2D
5	20	6.02	5.71	0.05
5	50	4.10	3.93	0.62
5	100	2.18	1.98	4.37
10	20	7.14	7.53	0.06
10	50	5.75	5.74	0.50
10	100	3.38	3.27	3.13
10	200	2.22	2.09	20.06
20	50	7.07	7.33	0.60
20	100	4.41	4.45	2.89
20	200	3.31	3.30	15.68
20	500	1.44	1.40	182.24

Table 7.15: Comparison of improvement heuristics for two dominating machines

which CPLEX failed as well. T2D slightly outperforms the other improvement heuristics in terms of optimal solutions found for all except three sets of instances. With the methods developed in this work, we are able to derive very good lower and upper bounds on the optimal value of the makespan of synchronous flow shop problems with two dominating machines. The gaps are very small, especially when comparing them to the large gaps for the general case. Even for the largest instances, the improvement heuristics managed to obtain a solution that is only 0.04 % larger than the best lower bound.

		# Verified Optimum				average gap from best LB			
m	n	constr.	impr.	T2D	IP	constr.	impr.	T2D	IP
5	20	0	3	6	10	4.39 %	0.44 %	0.11 %	0.00 %
5	50	0	1	9	10	1.82 %	0.19 %	0.01 %	0.00 %
5	100	0	0	5	9	1.54 %	0.22 %	0.03 %	0.03 %
10	20	0	6	4	10	6.85 %	0.20 %	0.63 %	0.00 %
10	50	0	2	1	8	6.21 %	0.61 %	0.60 %	0.03 %
10	100	0	1	2	10	2.79 %	0.20 %	0.09 %	0.00 %
10	200	0	1	2	0	1.99 %	0.17 %	0.04 %	1.68 %
20	50	0	1	0	10	8.26 %	0.47 %	0.75 %	0.00 %
20	100	0	1	1	9	4.76 %	0.37 %	0.41 %	0.02 %
20	200	0	0	0	0	2.83 %	0.17 %	0.16 %	1.75 %
20	500	0	0	0	0	1.22 %	0.08 %	0.04 %	3.50 %

Table 7.16: Number of verifiable optimal solutions and average gaps for two dominating machines

7.1.2 Maximum lateness

In this section we evaluate the methods to minimize the maximum lateness. During the evaluation we experienced that the distribution of due dates has a great impact on the runtime and the overall performance of our algorithms. As this effect can already be observed for small instances, we generated a test set containing instances with varying distribution of due dates. To generate an instance, we specified the number of machines m , the number of jobs n as well as an upper limit $\bar{p}$ on the processing time and an upper limit $\bar{d}$ on the due dates. For each job and each machine we randomly chose a processing time in the interval $[0, \bar{p}]$ and for each job we randomly chose a due date in the interval $\left[1, \bar{d}\right]$. Similar to the test set of Taillard, for each combination of m,n,$\bar{p}$ and $\bar{d}$ we generated 10 instances. In Table 7.17 we show a comparison between the results of the branch and bound algorithm with the lower bound described in Section 4.4.2 and the results obtained by using CPLEX on the mixed integer formulation described in Section 4.2 (denoted by IP in the table) when using both approaches with a time limit of 30 minutes. Depicted are the number of solved instances out of 10 for each generated combination of m,n,$\bar{p}$ and $\bar{d}$ as well as the average runtime in seconds for solved instances. It can be seen that for instances where the due dates are only distributed over a very small interval, both algorithms perform worse in comparison to instances in which due dates are distributed over a larger interval. The smaller the intervals are in comparison to the number of jobs and their processing times, the more similar the problem of minimizing the maximum lateness becomes to the problem of minimizing the makespan. Analogous to Section 7.1.1, the IP solver is superior to the branch and bound algorithm in these cases. However, the larger the interval over which the due dates are distributed, the better the branch and bound algorithm performs in comparison to CPLEX. Further, if the due dates are distributed over a very large interval, both algorithms obtain optimal solutions in very low computation time even for the larger test instances.

				#		avg. time						#		avg. time	
m	n	$\bar{p}$	$\bar{d}$	IP	BB	IP	BB	m	n	$\bar{p}$	$\bar{d}$	IP	BB	IP	BB
2	10	40	100	10	10	0.29	0.07	2	30	40	900	8	8	203.45	23.27
2	15	40	100	10	7	10.174	351.10	2	30	400	12000	10	10	132.49	12.95
2	15	40	300	10	10	2.47	9.95	5	10	40	100	10	10	0.95	0.15
2	20	40	100	0	0	–	–	5	15	40	100	10	5	149.58	541.53
2	20	40	400	10	9	169.06	167.32	5	15	40	300	10	10	23.93	8.47
2	20	400	8000	10	10	6.22	0.10	5	20	40	400	4	5	1049.08	636.49
2	25	40	100	1	0	91.07	–	5	25	40	875	8	10	165.94	75.50
2	25	400	10000	10	10	0.67	0.01	5	25	400	10000	10	10	32.54	6.05

Table 7.17: Comparison of the number of instances solved to optimality within 30 minutes and the average computation time of solved instances

Since the original test set of Taillard only contains processing times, we altered the test set by specifying due dates for all jobs. As described above, the difficulty of the problem depends heavily on the distribution of due dates. We were neither interested in too tightly

distributed due dates that turn the problem into one of minimizing the makespan nor in too far spread due dates that make the problem too easy. Therefore, we randomly chose the due dates from the interval $[1, 100n]$. Again, for each combination of m and n we generated 10 instances with the same processing times of all operations as in the original instances by Taillard and due dates as described above. For all but one instance with 20 jobs, the branch and bound algorithm was able to obtain an optimal solution. Further, even four instances with 5 machines and 50 jobs could be solved to optimality by the branch and bound algorithm. CPLEX using the mixed integer programming formulation described in Section 4.2 was also able to solve all but one instance with 20 jobs to optimality, but none of the other instances.

Unfortunately, for all other problem sets, the lower bounds obtained by CPLEX after 30 minutes as well as the lower bound LB-1D derived in Section 4.4.2 perform extremely bad in comparison to the best feasible solutions obtained by either CPLEX or the branch and bound algorithm. The best obtained feasible solution is at least 100 % larger than the lower bound obtained by 1DOM even for 5 machines and 50 jobs and increasing to well over 1000 % for larger instances (with average of absolute values being 437 and 6070, respectively). The lower bound obtained by CPLEX performs even much worse for all instances. We omit the table comparing the lower bounds over all class of instances. Unfortunately, the branch and bound algorithm achieves even worse upper bounds than CPLEX for the instances in which it does not return an optimal solution, except for the remaining instances with 5 machines and 50 jobs. Also, it does not deliver a global lower bound.

In the following, we evaluate the constructive heuristics 1DOM and BW discussed in Chapter 5.1. We use two variants of BW. For both variants, BW is executed iteratively 10 times with different threshold values and approximations of the makespan. In each case, we first sort the jobs in EDD order and use the makespan of the resulting schedule and the maximum lateness of this schedule reduced by one as the first approximation value for the makespan and the first threshold value for BW. Further, the threshold value for each following execution of BW is the hitherto best found maximum lateness reduced by one. In the first variant, BW-I, the makespan of the schedule obtained by the previous execution of BW is used as the approximation of the makespan for the following execution. In each iteration of BW-I, when the set of feasible jobs is not empty, we choose a job to be scheduled which maximizes the sum of the cycle times of the last $m - 1$ by the largest amount (similar to choosing the job with the largest processing time in the case with one dominating machine). If the set of feasible jobs is empty, we choose a remaining job with maximum due date. The second variant, BW-II, uses a more randomized approach. As the approximation value of the makespan for the execution of the next iteration we choose $(0.5 + r)C$ where C is the makespan of the schedule obtained by the last iteration and r is a real random number uniformly chosen from the continuous interval $[0, 1]$. Also, in BW-II we always choose a random job from the set of feasible jobs. If the set of feasible jobs is empty, we choose a random remaining job. Table 7.18 shows the results for these heuristics. We also depict the results obtained by scheduling the jobs in EDD order. As in the previous section, we show how often each heuristic performs best in comparison to the other constructive heuristics as well as the average deviation from the

best obtained constructive solution. Both BW heuristics perform best in comparison to the other heuristics. For smaller instances, BW-II is better than BW-I while for the larger instances the deterministic algorithm BW-I is superior. Simply scheduling the jobs in EDD order on average leads to very bad results in comparison, especially for instances with only 5 jobs. The 1DOM heuristic performs very bad, especially for large instances.

		# times best constructive solution				average deviation from best result			
m	n	EDD	BW-I	BW-II	1Dom	EDD	BW-I	BW-II	1Dom
5	20	0	1	2	0	33.20	16.81	10.27	128.36
5	50	0	0	1	0	22.53	15.85	18.82	348.80
5	100	0	1	3	0	17.98	10.90	12.55	858.63
10	20	0	0	4	0	8.88	6.99	2.75	46.38
10	50	0	0	4	0	9.28	6.66	4.58	165.99
10	100	0	1	1	0	17.88	9.03	14.66	333.88
10	200	1	2	1	0	10.32	7.17	7.60	822.31
20	20	0	2	1	0	4.19	3.33	1.89	15.17
20	50	0	1	1	0	4.49	3.42	2.64	76.51
20	100	0	4	0	0	5.02	2.60	4.21	174.47
20	200	0	2	0	0	10.24	5.74	8.23	386.53
20	500	1	1	1	0	11.37	5.65	9.85	977.37

Table 7.18: Comparison of the iterative constructive heuristics for minimizing the maximum lateness

Next, we evaluate the NEH heuristics. As an initial sorting for the standard NEH heuristic we choose the EDD order of jobs. Similar to Section 7.1.1 we evaluate two different evaluation functions for the insertion of jobs. In the first case (denoted by s in Table 7.19), for each insertion of a job we only consider the maximum lateness of the partial schedules and insert the job into a position which yields the minimal maximum lateness. In the second case (denoted by Σ), if two partial schedules achieve the same maximum lateness, we choose to insert the job into a position which yields the minimum sum of lateness $\sum L_j$. Both insertion functions are tested for NEH and NEHA, respectively. Table 7.19 shows the result for the NEH heuristics. It can be seen that the NEHA heuristic is clearly superior to the standard NEH heuristic, especially for larger instances. Only for smaller instances, NEH is sometimes able to reach the best constructive solution, yet the average deviation from the best constructive solution is very large. Further, it can be seen that using the second evaluation functions has little impact on the performance of the NEHA heuristic while for the standard NEH variant the introduction of the second criterion leads to a clear improvement.

To compare the constructive heuristics, Figure 7.7 depicts how often each constructive heuristic reaches the best solution as well as how often it is the only heuristic to reach a best solution. Figure 7.8 shows the average deviation of each constructive heuristic from the best constructive solution. We omit the results of the 1DOM heuristic as its average deviation from the best solution is 361.20 % and for no instance obtains the best

		# times best constr. sol.				avg. dev. from best result			
		NEH		NEHA		NEH		NEHA	
m	n	s	Σ	s	Σ	s	Σ	s	Σ
5	20	0	2	5	5	35.30	35.03	3.96	3.96
5	50	2	1	6	6	33.36	53.54	2.38	2.38
5	100	1	0	5	4	96.84	64.55	1.59	1.60
10	20	2	2	4	4	11.25	10.16	2.98	2.98
10	100	0	1	5	5	31.29	26.16	1.49	1.49
10	200	0	0	10	8	40.94	21.07	0.00	0.61
10	50	0	0	9	9	62.02	56.15	0.20	0.20
20	20	3	3	4	4	5.54	5.09	0.85	0.65
20	50	0	0	7	8	9.46	5.73	0.14	0.16
20	100	0	0	6	6	32.50	13.22	0.38	0.38
20	200	0	0	5	8	53.14	18.45	0.76	0.15
20	500	0	0	7	5	82.32	40.07	0.60	0.65

Table 7.19: Comparison of NEH variants for minimizing the maximum lateness

constructive solution. It can be seen that NEHA outperforms all other heuristics with an average deviation from the best constructive solution of 1.28 % for the standard insertion procedure and 1.27 % when using the second criterion.

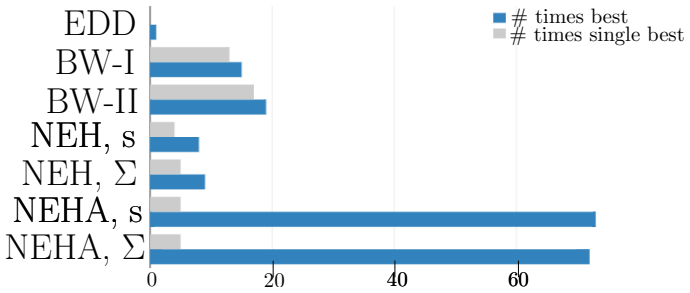


Figure 7.7: Comparison of constructive heuristics for minimizing the maximum lateness

To evaluate the metaheuristics we chose the same setting as in Section 7.1.1. We ran each metaheuristic using the result obtained by the standard version of NEHA as a starting solution. In Figure 7.20 we depict the average improvement of the starting solution in percent. Further, we depict the average computation time. The tabu search and iterative improvement procedure perform very similar and in general lead to better results than the simulated annealing procedure. Only for 5 machines and 20 jobs is the simulated annealing procedure able to outperform the other metaheuristics. Similar as for the other objective functions, the iterative improvement procedure performs at least as well as the tabu search for larger instances while requiring a way smaller runtime. In comparison to minimizing

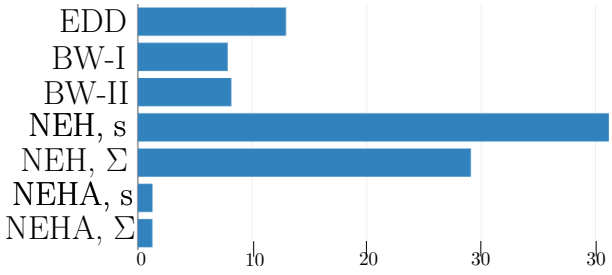


Figure 7.8: Average deviation of constructive heuristics from best obtained solution for minimizing the maximum lateness

the makespan, all heuristics require a larger runtime. This is most likely caused by the larger time requirement to evaluate the change in maximum lateness when making changes to a solution in comparison to the time required to evaluate the change of makespan or the total completion time. Again, using other starting solutions lead to the same relative performance between the metaheuristics.

		avg. impr. of starting solution						average computation time					
		Tabu		SA		II		Tabu		SA		II	
m	n	sw	sh	sw	sh	sw	sh	sw	sh	sw	sh	sw	sh
5	20	11.82	14.65	17.51	16.79	9.67	8.26	0.0	0.1	0.0	0.0	0.0	0.0
5	50	11.24	5.93	8.22	6.16	8.26	6.34	0.5	1.1	0.0	0.0	0.0	0.0
5	100	6.23	4.82	0.97	1.24	4.06	4.61	4.1	7.7	0.0	0.0	0.1	0.4
10	20	8.31	5.35	7.21	5.98	7.17	3.99	0.1	0.1	0.0	0.0	0.0	0.0
10	50	7.97	5.19	6.71	4.09	6.46	5.08	1.1	1.8	0.0	0.0	0.0	0.1
10	100	9.26	7.18	3.20	2.95	9.77	5.19	6.8	13.5	0.0	0.0	0.2	1.0
10	200	9.03	7.64	0.03	0.30	8.69	6.81	55.5	101.4	0.0	0.0	1.5	7.8
20	20	3.87	3.19	3.37	2.39	3.66	2.79	0.2	0.3	0.0	0.0	0.0	0.0
20	50	6.01	3.99	4.73	2.77	5.16	3.18	2.2	3.5	0.0	0.0	0.1	0.2
20	100	6.49	5.28	3.74	0.99	6.29	4.88	14.5	24.1	0.0	0.0	0.5	1.9
20	200	4.70	3.34	0.60	0.44	4.99	2.56	132.5	181.2	0.0	0.0	4.1	24.0
20	500	9.58	4.91	0.14	0.08	9.95	5.68	1719.3	1800.0	0.1	0.1	74.8	643.3

Table 7.20: Comparison of metaheuristics employing the swap (sw) and shift (sh) neighborhoods for the benchmark instances of Taillard

Finally, we compare the heuristics to the exact methods and depict the gaps between the best obtained solutions and the best lower bounds in Figure 7.21. We show, how often the best constructive solution and the best solution obtained by an improvement heuristic (using various starting solutions) deliver a better result than the upper bound obtained by CPLEX within 30 minutes. For one instance of 5 machines and 20 jobs, the BW-II heuristic was able to obtain an optimal objective value which could be verified. For none of the other instances could any of the heuristics find a solution that could be verified

as optimal. It can be seen that the constructive and improvement heuristics achieve way better results than CPLEX for all problem sizes except the smallest ones containing 20 jobs which can be solved to optimality by CPLEX. For larger instances, the gap between the CPLEX upper bound and the best lower bound is well over 1000 %. While the gaps for the heuristics are still large as well, this may again be attributed to the bad performance of the lower bounds.

		# times better than CPLEX		average gap from best LB		
m	n	constr.	impr.	constr.	impr.	CPLEX
5	20	0	0	42.05 %	2.70 %	0.00 %
5	50	7	10	78.53 %	37.74 %	122.45%
5	100	10	10	95.44 %	68.00 %	1931.58%
10	20	0	0	11.44 %	0.13 %	0.00 %
10	50	8	10	113.09%	88.81 %	152.62%
10	100	10	10	163.28%	122.35%	1570.20%
10	200	10	10	119.08%	87.29 %	»1000 %
20	20	0	0	17.19 %	10.19 %	7.86 %
20	50	10	10	98.06 %	82.25 %	187.82 %
20	100	10	10	87.38 %	69.61 %	646.01 %
20	200	10	10	108.33%	88.06 %	»1000 %
20	500	10	10	142.42%	105.94%	»1000 %

Table 7.21: Number of times the constructive and improvement heuristics perform better than CPLEX, and gaps for minimizing the maximal lateness

7.1.3 Total completion time

In this section we evaluate the methods to minimize the total completion time using the test set of Taillard. Analogous to the study of minimizing the makespan of general synchronous flow shop problems in Section 7.1.1, neither CPLEX using the mixed integer programming formulation described in Section 4.2 nor our branch and bound algorithm were able to obtain an optimal solution for any instance of the test set within a time limit of 30 minutes. For two of the largest instances containing 20 machines and 500 jobs, CPLEX was not even able to obtain feasible solutions. To test the integer programming approach versus our branch and bound algorithm, we created a test set of smaller instances, ranging from 10 to 25 jobs on 2 to 5 machines. For each combination of machines and jobs, ten instances were generated. The processing times on all machines were chosen uniformly in the interval $[0, 100]$. As can be seen in Table 7.22, the branch and bound approach using the SPT lower bound outperforms CPLEX in time and number of solved instances. CPLEX was not able to solve any instance with 15 or more jobs to optimality, while the branch and bound algorithm solved all instances with 15 or fewer jobs in very short time. Only for instances with 20 or more jobs the algorithm did not reach an optimal solution within the time limit on several occasions but was still able to solve some of the instances. While the

branch and bound algorithm was able to solve 3 of the additionally generated instances with 5 jobs and 20 jobs to optimality (and even was able to solve instances with 25 jobs and 2 machines), we could not obtain solutions for the instances of the same size contained in the test set of Taillard.

		# solved		avg. time	
m	n	IP	BB	IP	BB
2	10	10	10	11.52	0.01
2	15	0	10	–	4.05
2	20	0	10	–	76.6
2	25	0	4	–	619.67
3	25	0	0	–	–
5	10	10	10	44.12	0.01
5	15	0	10	–	15.09
5	20	0	3	–	633.93

Table 7.22: Comparison of the number of instances solved to optimality within 30 minutes and the average computation time of solved instances

In Table 7.23 we compare three lower bounds for the test set of Taillard: LB-SPT presents the lower bound obtained by relaxing the problem into a synchronous flow shop problem with a single dominating machine and applying the SPT rule as described in Section 4.4.3. The results achieved by the parallel machine bound are denoted by LB-P, while LB-IP denotes results for the lower bound obtained by CPLEX after a time limit of 30 minutes. Unfortunately, for the largest problem sizes (60 instances in total), CPLEX could not obtain a lower bound within the time limit for any of the instances. Further, we depict the average deviation from the best lower bound of the best feasible solution found by CPLEX (UB-IP). For larger instances, LB-P is clearly superior to the SPT bound while for smaller instances, the latter performs better in comparison. We omit the results obtained by the branch and bound algorithm as it was not designed to compute global lower bounds and its upper bounds for instances it did not solve to optimality were clearly inferior to the ones derived by CPLEX.

In the following, we discuss constructive heuristics for minimizing the total completion time of synchronous flow shops. First, we evaluate three different methods of sorting the jobs according to their processing times: The standard sorting in order of non-decreasing total processing time $\sum_{i=1}^{m} p_{ij}$ as well as two weighted versions. In the first weighted version, $\sum_{i=1}^{m} \frac{i}{m} p_{ij}$, the influence of the processing times of a jobs operation increases with the machine index, i.e. the larger the processing times of the operations on the later machines, the larger the weight of the job. In the second weighted version, $\sum_{i=1}^{m} \frac{m-i+1}{m} p_{ij}$ the processing times of the operations on the earlier machines have a larger influence. Finally, we also evaluate the 1dom heuristic. Table 7.24 summarizes the results. It can be seen that sorting the jobs according to the first weighted variant overall leads to the best results in comparison to other sorting methods. The 1dom heuristic performs very bad in general, only for the smallest instances with 20 machines and 20 jobs it is superior to

		average deviation from best LB			
m	n	LB-SPT	LB-P	LB-IP	UB-IP
5	20	1.44	4.59	92.86	40.22
5	50	7.02	0.00	–	47.88
5	100	16.01	0.00	–	63.03
10	20	1.25	10.24	46.66	68.72
10	50	2.57	2.45	–	82.42
10	100	10.65	0.00	–	93.46
10	200	18.60	0.00	–	106.14
20	20	0.00	11.31	76.22	106.06
20	50	0.00	8.09	–	100.51
20	100	1.39	3.62	–	115.65
20	200	12.71	0.00	–	116.24
20	500	25.08	0.00	–	109.32*

Table 7.23: Comparison of lower bounds or minimizing the total completion time
*: No feasible solution were obtained for two instances

the other methods. The total completion times of the schedules obtained by the sorting procedures on average deviate a lot from the best heuristic results. For no instance does simply sorting the jobs lead to the best constructive solution.

		# times best constructive solution				average deviation from best result			
m	n	p_{ij}	$\frac{i}{m}p_{ij}$	$\frac{m-i+1}{m}p_{ij}$	1dom	p_{ij}	$\frac{i}{m}p_{ij}$	$\frac{m-i+1}{m}p_{ij}$	1dom
5	20	0	0	0	0	17.64	16.28	22.34	30.87
5	50	0	0	0	0	25.25	24.50	29.93	41.03
5	100	0	0	0	0	28.07	29.67	32.12	41.41
10	20	0	0	0	0	16.02	12.27	18.47	19.51
10	50	0	0	0	0	20.34	18.16	22.80	24.07
10	100	0	0	0	0	20.96	21.53	25.05	29.07
10	200	0	0	0	0	25.46	25.67	28.33	32.98
20	20	0	0	0	0	9.21	8.15	9.63	6.19
20	50	0	0	0	0	13.00	11.25	15.32	16.36
20	100	0	0	0	0	14.95	14.21	16.83	17.82
20	200	0	0	0	0	17.03	16.42	19.26	20.27
20	500	0	0	0	0	19.92	19.53	20.96	22.16

Table 7.24: Comparison of the sorting procedures for minimizing the total completion time

Next, we evaluate the performance of some of the iterative construction heuristics that were originally designed for minimizing the makespan; S3, S3I, S3DE, S3W and S3DEW. As described in Section 5.1, we alter heuristic S3 such that in each iteration it inserts a job at the end of the schedule which leads to the minimal total completion time for the

partial schedule. In S3I we add another criterion; in case that two jobs lead to the same total completion time we add a job for which the sum of idle times that would occur for the operations of the job to be scheduled is lowest. S3IW and S3DEW use a weighted sum of the two values, similar to Section 7.1.1 we choose a weight of $w = 0.8$ for our evaluation. Table 7.25 summarizes the results. The results differ vastly from the ones obtained for these heuristics when minimizing the makespan. S3DE as well as its weighted counterpart, which were the best heuristics for minimizing the makespan, perform particularly poor in comparison to the other heuristics. Overall, S3W performs best, especially for larger instances. S3 also performs better in comparison to S3I for minimizing the total completion time, only for larger instances on 20 machines S3I is slightly superior.

		# times best constructive solution					avg. deviation from best result				
m	n	S3	S3I	S3IW	S3DE	S3DEW	S3	S3I	S3W	S3DE	S3DEW
5	20	5	0	1	0	0	1.62	2.97	1.97	12.78	13.08
5	50	1	0	1	0	1	1.58	3.56	2.21	12.40	10.75
5	100	6	0	1	0	0	0.37	3.64	2.57	25.41	21.68
10	20	4	0	4	0	0	1.10	1.52	0.75	7.82	7.27
10	50	0	1	4	0	0	1.43	0.99	0.64	11.77	11.26
10	100	2	3	5	0	0	0.52	0.65	0.33	11.35	11.44
10	200	6	0	1	0	0	0.27	1.11	0.33	12.06	12.43
20	20	3	1	2	0	0	0.71	1.29	1.10	6.87	5.64
20	50	1	3	4	0	0	0.96	0.53	0.44	11.57	12.19
20	100	0	1	6	0	0	0.69	0.34	0.16	12.28	11.65
20	200	0	2	7	0	0	0.45	0.34	0.12	11.06	11.18
20	500	0	2	8	0	0	0.43	0.24	0.01	13.19	12.40

Table 7.25: Comparison of the constructive heuristics for minimizing the total completion time

Table 7.26 depicts the performance of the NEH heuristics. For the NEH procedure that uses a presorting of the jobs, we use two variants. The first variant (denoted as Σ) initially sorts the jobs in non-decreasing order of weighted total processing time $\sum_{i=1}^{m} \frac{i}{m} p_{ij}$ which was the best sorting procedure as determined above. The second variant (denoted as S3) uses the schedule obtained by the constructive heuristic S3 as a sorting for the jobs. As for the other heuristics, we present how often each variant is the best in comparison to the other constructive heuristics and its average deviation from the best constructive solution. As can be seen, sorting the jobs in the order obtained by the iterative heuristic, clearly outperforms the variant where the jobs are initially sorted in non-decreasing order of total processing time, especially for larger instances. NEHA delivers the best average deviation from the best constructive solution except for the smallest instances. For the largest instance with 20 machines and 500 jobs the NEHA is only slightly superior to the standard NEH procedure using the S3 presorting. However, in this case, NEHA requires an average runtime of 105 seconds while NEH finishes within 2.6 seconds on average. All

other computation times for all other problem sizes are under one second for all NEH variants.

		# times best constr. sol.			average objective value		
m	n	Σ	S3	NEHA	Σ	S3	NEHA
5	20	0	0	4	5.97 %	2.08 %	1.25 %
5	50	0	1	6	8.42 %	2.76 %	1.26 %
5	100	0	0	3	8.58 %	3.48 %	1.23 %
10	20	0	1	1	6.66 %	2.41 %	2.48 %
10	50	0	2	3	8.83 %	3.69 %	0.88 %
10	100	0	0	0	10.66 %	3.14 %	0.77 %
10	200	0	0	3	13.75 %	3.08 %	0.38 %
20	20	0	2	2	3.55 %	1.65 %	1.74 %
20	50	0	1	1	6.95 %	3.59 %	1.16 %
20	100	0	1	2	8.29 %	4.59 %	0.99 %
20	200	0	1	0	11.43 %	2.86 %	0.97 %
20	500	0	0	0	14.00 %	0.98 %	0.55 %

Table 7.26: Comparison of the NEH heuristics for minimizing the total completion time

To summarize the evaluation of the constructive heuristics, Figure 7.9 depicts how often each constructive heuristic reaches the best constructive solution. It never occurred that a best solution was achieved by two distinct algorithms for any instance. Figure 7.10 shows the average deviation of each constructive heuristic from the best constructive solution. It can be seen that all variants of S3, except for S3DE and its weighted counterpart, perform very similar and offer very good results. Similar to minimizing the makespan and minimizing the maximum lateness, NEHA also reaches a very low average deviation from the best objective value of 1.14 %. However, opposed to the other two objective functions it does not yield the lowest average deviation as S3 and SEW are superior with an average deviation of 0.84 % and 0.89 %, respectively.

Again, we chose the same setting to evaluate the metaheuristics as for minimizing the makespan and the maximum lateness. This time, each metaheuristic was started with the starting solution obtained by the S3 heuristic. Table 7.27 summarizes the relative performance of the metaheuristics. The results are very similar to the ones obtained for minimizing the makespan. Again, using the swap neighborhood leads to a better performance than using the shift neighborhood and the tabu search leads to the best average improvement of the solution. However, especially for larger problem sizes, the results obtained by simulated annealing and iterative improvement are very close to the ones of the tabu search, while these two procedures require a way lower computation time.

Finally, we once again compare the heuristics to the exact algorithms and determine the gaps between the best obtained solutions and the best lower bounds. Table 7.28 shows, how often the best solution obtained by a constructive or the best result obtained by the improvement heuristic was better than the upper bound obtained by CPLEX within

30 minutes. Only for the smallest instances, CPLEX achieves the best results. For all instances with 50 or more jobs, already the best constructive heuristic outperforms the IP solution. Unfortunately, once again, the gaps are very large for all problem sizes. Further, the improvement heuristics only manage to improve very little on the best constructive solution, especially for the largest instances. Similar to the other objective functions, the reason behind this most likely lies in the overall bad quality of the lower bounds.

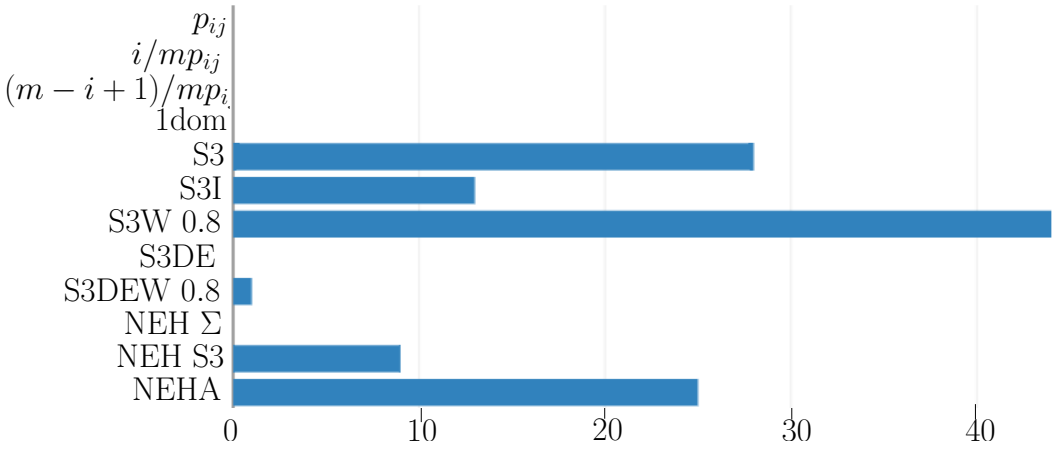


Figure 7.9: Comparison of constructive heuristics for minimizing the total completion time

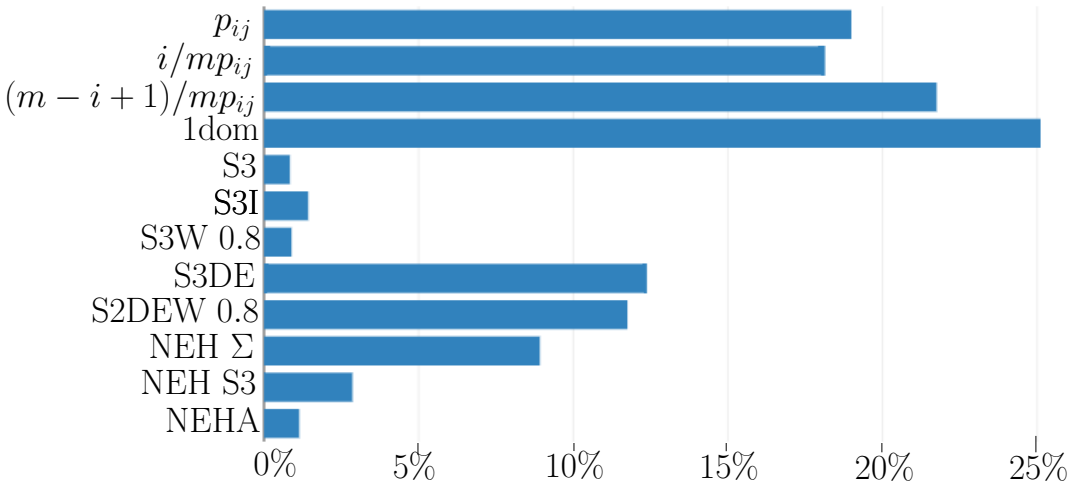


Figure 7.10: Average deviation of constructive heuristics from best obtained solution for minimizing the total completion time

		avg. impr. of starting solution						average computation time					
		Tabu		SA		II		Tabu		SA		II	
m	n	sw	sh	sw	sh	sw	sh	sw	sh	sw	sh	sw	sh
5	20	2.31	1.37	1.68	1.68	1.73	1.70	0.0	0.0	0.0	0.0	0.0	0.0
5	50	2.19	0.39	1.32	0.38	1.05	0.35	0.4	0.5	0.0	0.0	0.0	0.0
5	100	1.69	0.19	1.25	0.19	1.22	0.18	2.4	3.8	0.0	0.0	0.0	0.0
10	20	2.49	1.24	1.49	0.89	1.71	0.98	0.0	0.0	0.0	0.0	0.0	0.0
10	50	1.66	0.49	1.14	0.52	1.17	0.44	0.3	0.4	0.0	0.0	0.0	0.0
10	100	0.88	0.08	0.38	0.06	0.48	0.07	1.9	2.8	0.0	0.0	0.0	0.0
10	200	0.53	0.04	0.35	0.03	0.41	0.04	13.4	20.2	0.0	0.0	0.0	0.2
20	20	3.02	1.09	1.26	0.71	1.90	1.02	0.1	0.1	0.0	0.0	0.0	0.0
20	50	1.64	0.37	0.64	0.26	0.85	0.37	1.5	1.7	0.0	0.1	0.0	0.0
20	100	0.94	0.16	0.51	0.15	0.53	0.15	5.5	5.2	0.1	0.0	0.0	0.1
20	200	0.43	0.02	0.21	0.01	0.25	0.02	32.0	36.0	0.1	0.1	0.2	0.4
20	500	0.23	0.05	0.02	0.04	0.13	0.05	835.7	616.3	0.1	0.1	1.3	4.3

Table 7.27: Comparison of metaheuristics employing the swap (sw) and shift (sh) neighborhoods for minimizing the total completion time

		# times better than CPLEX		average gap from best LB		
m	n	constr.	impr.	constr.	impr.	CPLEX
5	20	1	4	42.39 %	40.22 %	40.22 %
5	50	10	10	39.10 %	37.38 %	47.88 %
5	100	10	10	34.20 %	32.13 %	63.03 %
10	20	2	4	71.32 %	68.81 %	68.72 %
10	50	10	10	67.50 %	66.29 %	82.42 %
10	100	10	10	61.22 %	60.08 %	93.46 %
10	200	10	10	54.21 %	53.53 %	106.14 %
20	20	0	4	111.16%	105.64%	106.06 %
20	50	10	10	86.96 %	85.04 %	100.51 %
20	100	10	10	85.20 %	84.62 %	115.65 %
20	200	10	10	79.76 %	79.73 %	116.24 %
20	500	10	10	71.22 %	71.22 %	109.32 %

Table 7.28: Number of times the constructive and improvement heuristics outperform CPLEX, and gaps for minimizing the total completion time

7.2 Practical application

In this section we will present computation results for the real world application discussed in Chapter 6. We tested the A-team approach described in Section 6.3 on a test set given to us by the practitioner consisting of order pools for 18 work days during the month of September 2012. For each day the practitioner gave us the respective product, volume and due date of all orders that had sufficient supply of raw material and thus were ready for production. The time frame to be scheduled on each day was given as well, usually ranging between 16 and 24 hours. The gluing and insertion times for each product were determined by the practitioner (with a mean of 50 seconds for each insertion and gluing process) and the changeover time s was set to six minutes. For each day d, Table 7.29 shows the number of orders n and the total volume $\sum v_j$. The numbers of orders and individual jobs that are due on that day (i.e. have a due date smaller or equal to d) are given, both in absolute number and as a sum of the penalty values $\sum (\max(0, d - d_j + 1))^2$, given in brackets. This value relates to the penalty an order receives if it can not be completed on day d.

d	n	$\sum v_j$	due orders	due jobs	d	n	$\sum v_j$	due orders	due jobs
1	168	11498	43 (472)	3688 (370111)	10	163	9579	36 (441)	2891 (581633)
2	186	12359	55 (445)	4993 (515364)	11	161	10997	41 (155)	3740 (254584)
3	193	11656	38 (189)	2522 (91146)	12	151	12246	35 (146)	3773 (307567)
4	187	10601	67 (649)	5155 (1060704)	13	148	10464	46 (181)	4065 (320490)
5	171	12623	52 (578)	4559 (834862)	14	129	8553	40 (150)	3344 (181984)
6	148	7284	30 (251)	2634 (96646)	15	123	7361	60 (269)	3574 (246020)
7	208	11983	68 (392)	4154 (395420)	16	143	8764	53 (433)	2756 (453874)
8	181	9920	48 (267)	3298 (232623)	17	162	9024	74 (358)	3854 (329135)
9	180	10013	45 (219)	3447 (271147)	18	193	12554	69 (360)	3739 (308618)

Table 7.29: Test set given to us by the practitioner

In Table 7.30 and Figure 7.11 the results of the A-Team algorithm and the actual production are compared. The A-Team results were obtained from running the algorithm for a total of one hour (again, on an Intel i7 8-Core 3.4 GHz and 8 GB RAM) and taking the best found solution. We chose one hour because this is the time allowed by the practitioner in daily production planning. In comparison, prior to the implementation of the algorithm, a human expert was allowed up to four hours of planning. The actual production numbers were given to us by the practitioner, however, we did not obtain the actual production plan (i.e. assignments to stations and cycles). For each day we give the number of orders that were due on this day but could not be completed on time in the A-Team algorithm and the actual production, respectively. In brackets we also give the sum of the respective penalties for late orders and products. Furthermore, the total number of jobs is presented.

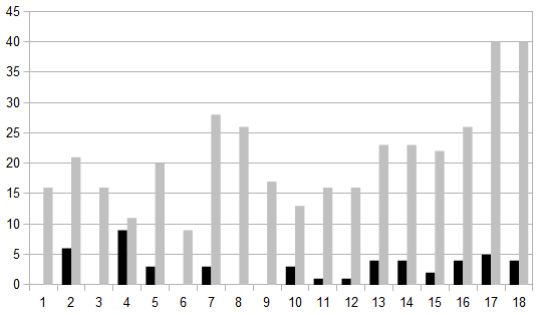
It can be seen that the results of the A-Team algorithms are always better than the actual production in terms of producing due orders on time. In all but four cases the fine planning's total production is also larger than the actual production on the respective day, however in most cases the difference is very small. This shows that the practitioner while being capable of producing "enough" tends to spend time producing "wrong" (i.e. non-critical) jobs. One reason may be a sub-optimal assignment of gluing forms and stations

	A-Team					actual production				
day	late orders		late jobs		prod.	late orders		late jobs		prod.
1	0	(0)	0	(0)	5295	16	(208)	1169	(52907)	4564
2	6	(23)	748	(3760)	5499	21	(276)	1931	(79560)	5359
3	0	(0)	0	(0)	5349	16	(133)	607	(4384)	4959
4	9	(9)	1113	(1113)	5175	11	(80)	801	(3339)	5930
5	3	(83)	929	(52209)	5282	20	(293)	2136	(57774)	5447
6	0	(0)	0	(0)	5796	9	(125)	578	(4266)	5672
7	3	(3)	372	(372)	5862	28	(208)	1334	(15134)	5705
8	0	(0)	0	(0)	6349	26	(127)	1839	(18902)	5135
9	0	(0)	0	(0)	6373	17	(59)	1128	(3633)	6509
10	3	(3)	606	(606)	5440	13	(73)	955	(2575)	5298
11	1	(4)	400	(1600)	5313	16	(76)	1602	(6594)	4911
12	1	(9)	420	(3780)	5415	16	(78)	1496	(10359)	5157
13	4	(13)	702	(2268)	5106	23	(112)	2075	(14549)	4078
14	4	(4)	721	(721)	3897	23	(73)	1683	(4542)	2945
15	2	(2)	270	(270)	4984	22	(213)	2129	(14069)	4645
16	4	(28)	170	(570)	4161	26	(215)	798	(7198)	4739
17	5	(11)	516	(1295)	4602	40	(225)	1767	(6142)	4333
18	4	(7)	474	(864)	4424	40	(245)	1722	(7483)	3948

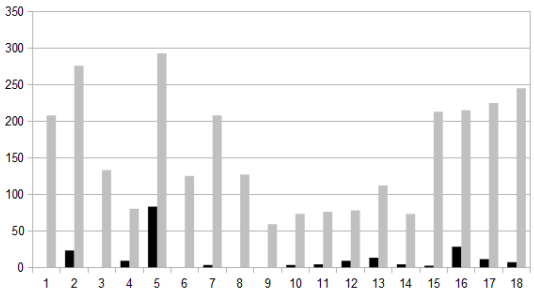
Table 7.30: Comparison of A-Team algorithm and actual production

that leads to situations in which it is only possible to produce non-critical orders. In such a case it seems reasonable that the practitioner tries to maximize its throughput and thus is able to achieve a similar production. Changing the hierarchies of the optimization criteria in such a way that maximization of total production is of highest priority, however, shows that the possible output of the production unit is much larger.

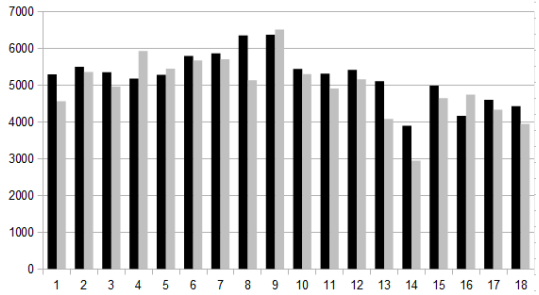
As noted above the A-Team algorithm is allowed to run for one hour. Figure 7.12 shows a representative example (the instance for day $d = 1$) of the relative evolution of the best solution value over time. For every three minutes the number of late orders and the total production volume are presented. As can be seen, the best starting solution already results in a reasonable schedule and the number of late orders decreases quickly to the final solution value. In many cases good schedules can be obtained well before the end of the A-Team procedure. The number of completed jobs, however, slightly increases over the total course of the runtime of the algorithm. Table 7.31 shows the number of minutes after which the best solution's value of the first criterion did no longer improve. Also the total production volume of this solution is compared to the total production volume of the final best solution by stating its ratio. While in some cases the best value for the first criterion was found only late within the one hour time limit, in two thirds of the cases it did not improve further after at most half an hour. Thus, while reasonable schedules can be found very early, a larger runtime nevertheless leads to a steady improvement.



a) late orders

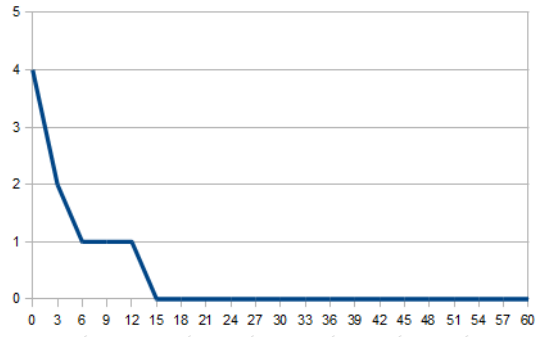


b) penalties for late orders

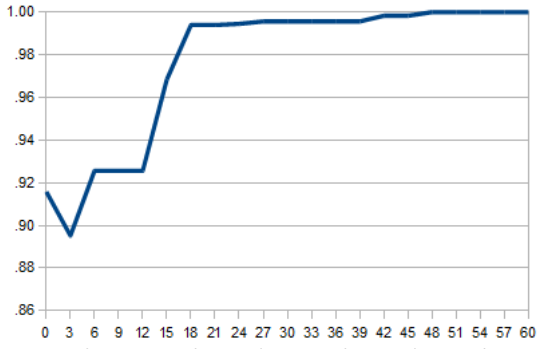


c) total production

Figure 7.11: Comparison of A-Team algorithm (dark bars) and actual production (light bars) per day



a) late orders



b) total production volume (in % of the final solution value)

Figure 7.12: Evolution over the course of the A-Team algorithm of the number of late orders and total production volume

day	min	perc	day	min	perc	day	min	perc	day	min	perc
1	14	0.98	2	17	0.93	3	1	0.93	4	39	0.97
5	52	0.97	6	26	0.97	7	55	0.98	8	1	0.96
9	28	0.92	10	2	0.95	11	53	0.98	12	56	0.96
13	8	0.98	14	27	0.97	15	16	0.96	16	1	0.96
17	55	0.99	18	18	0.97						

Table 7.31: Minutes after which the best solution value of the primary criterion was reached and percentage of the total production volume at that point

Chapter 8

Conclusion

In this thesis we offered a comprehensive overview on flow shops with synchronous movement. In this chapter we summarize the results and demonstrate the impact of the thesis. We also lay out possibilities for future work on this topic.

Thesis summary

Chapter 2 presents a thorough introduction into synchronous flow shop scheduling. The problem is compared to classical flow shop scheduling as well as to the related no-wait and blocking constraint. A generalized notion of machine dominance is introduced and embedded into the existing concepts that are prevalent in the literature. The distinctive features of synchronous flow shops with one or two dominating machines are discussed. Further, several extensions to synchronous flow shops, motivated by practical application, are presented.

In Chapter 3, an intensive study of complexity of various synchronous flow shop problems was conducted. We showed that the general synchronous flow shop problem is strongly $\mathcal{NP}$-hard for three or more machines when considering the makespan objective and strongly $\mathcal{NP}$-hard for two or more machines for all other objective functions discussed. Further, we derived several results for special cases with machine dominance.

In Chapter 4 two exact methods for obtaining optimal solutions for synchronous flow shop problems are described. The general problem as well as all special cases and extensions presented in the thesis are formulated as mixed integer linear programs. Further, several lower bounds are derived for the objective functions of minimizing the makespan, the maximum lateness and the total completion time to be used in a branch and bound algorithm.

Chapter 5 offers many heuristic approaches to solve synchronous flow shop problems. Various constructive as well as improvement heuristics are presented. Further, an asynchronous team approach is proposed to be used for extensions of synchronous flow shop problems that can not be solved by the standard procedures.

Chapter 6 demonstrates a practical application of a synchronous flow shop problem. Therein, a project in cooperation with a subcontractor for kitchen installers is described. The presented production system resembled a synchronous flow shop with two dominating

machines and cyclic changeovers. The work process was explained in detail and defined within the parameters of synchronous flow shop scheduling defined in this thesis. Because of the richness of the problem we concentrated on the asynchronous team approach to obtain solutions.

Chapter 7 offers an extensive computational study of the approaches discussed in this thesis. The exact algorithms and heuristics were tested on the well-known benchmark by Taillard, data supplied by our industrial partner as well as randomly generated test instances. The obtained results showed that the algorithms described in this thesis outperform the heuristics hitherto found in the literature for minimizing the makespan of synchronous flow shops. For minimizing the maximum lateness and the total completion time, we are the first to offer computational tests. The results of the asynchronous team approach show a noticeable improvement to the actual planning done by the practitioner.

Thesis impact

This thesis presents a thorough introduction and analysis of scheduling of flow shops with synchronous movement and systematically embeds the concepts into the flow shop scheduling framework.

The thesis offers an exhaustive study of complexity and settles the status of a large amount of the discussed problems. The complexity results for the basic synchronous flow shop and for special cases with dominating machines have already been published in Waldherr and Knust (2015).

The thesis contains a comprehensive computational study, portraying the good performance of the developed exact and heuristic methods for synchronous flow shop problems. For minimizing the makespan we were able to propose heuristics that outperform the heuristics found in the literature. In special cases with two non-adjacent dominating machines we were able to present a new mathematical programming formulation that achieves better results than the known formulation.

The two-stage planning approach discussed in Chapter 6 containing the multi agent framework proposed to solve the synchronous flow shop problem is still in operation at our industrial partner. The full description of the approach has been published in Waldherr and Knust (2014).

Due to the extensive study of the problem as well as discussions with practitioners, several extensions and practical applications were identified that spawn various open questions which can be investigated in the future.

Future work

Future work can be segmented into several different directions. For one, there are still open questions regarding more complicated objective functions that have not been discussed in more detail within this thesis. For instance, the complexity of synchronous flow shops is still open even for a single dominating machine when considering the objective functions of minimizing the total weighted completion time, $\sum w_j C_j$, or minimizing the number of late jobs, $\sum U_j$. Both objective functions are well established for classical flow shop and

well motivated by practical applications. Similar to the objective functions of minimizing the total unweighted completion time, $\sum C_j$ and the maximum lateness of jobs, $L_{\max}$, branching methods and heuristics discussed in this work could lead to good results for synchronous flow shops with dominating machines as well as general cases. Also, the complexity status of minimizing the makespan for two non-adjacent dominating machines remains open.

An important direction of future research is the development of better lower bounds for all objective functions. These are required to better evaluate the quality of the heuristic algorithms and to improve the performance of the branch and bound algorithms.

Further questions arise by deeper examination of the extensions for the synchronous flow shop. While the asynchronous team framework described within this thesis can be well adapted for all extensions of the synchronous flow shop, more specially designed algorithms may offer better results. It might be interesting to further investigate the effect of idle jobs in synchronous flow shop and how they can be exploited to improve the objective values for various objective functions. Also, the effects of job splitting offer a wild field to explore. Various possibilities to incorporate splitting in production processes may allow for better results.

Another possible direction of research emerges by considering alternative production systems instead of flow shops in which synchronous movement can take place. For instance, in open shop scheduling, each job also has to be processed on all available machines but the sequence of operations is not fixed like in a flow shop. In job shop scheduling, in comparison, each job may have its individual sequence that has to be adhered to. Also, parallel machine environments might be of interest. Therein, each job can be completed fully on each of the available machines which may have distinct processing speeds.

Bibliography

J.O. Achugbue and F.Y. Chin. Complexity and solutions of some three-stage flow shop scheduling problems. *Mathematics of Operations Research*, 7(4):532–544, 1982.

I. Adiri and D. Pohoryles. Flowshop/no-idle or no-wait scheduling to minimize the sum of completion times. *Naval Research Logistics Quarterly*, 29(3):495–504, 1982.

R. Akkiraju, P. Keskinocak, S. Murthy, and F. Wu. An agent-based approach for scheduling multiple machines. *Applied Intelligence*, 14(2):135–144, 2001.

J. Blazewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

J.D. Blocher and D. Chhajed. The customer order lead-time problem on parallel machines. *Naval Research Logistics (NRL)*, 43(5):629–654, 1996.

N. Boysen, M. Fliedner, and A. Scholl. Assembly line balancing: which model to use when? *International Journal of Production Economics*, 111:509–528, 2008.

N. Boysen, M. Fliedner, and A. Scholl. Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192:349–373, 2009.

S. Brockmeyer. *Synchrone Flow-Shop-Probleme mit Ressourcen und Rüstkosten (in German)*. Master Thesis, University of Osnabrück, 2014.

P. Brucker. *Scheduling algorithms*. Springer, 2007.

M. Bultmann. *Synchrone Flow-Shop-Probleme mit Job-Splitting (in German)*. Bachelor Thesis, University of Osnabrück, 2015.

H.G. Campbell, R.A. Dudek, and M.L. Smith. A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16(10):B630–B637, 1970.

P. Čap, O. Čepek, and M. Vlach. Weak and strong machine dominance in a nonpreemptive flowshop. *Scientiae Mathematicae Japonicae*, 61(2):319–334, 2005.

O. Čepek, M. Okada, and M. Vlach. Nonpreemptive flowshop scheduling with machine dominance. *European Journal of Operational Research*, 139(2):245–261, 2002.

J.R. Correa, M. Skutella, and J. Verschae. The power of preemption on unrelated machines and applications to scheduling orders. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 84–97. Springer, 2009.

D.G. Dannenbring. An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11):1174–1182, 1977.

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

J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965.

J.M. Framinan, J.N.D. Gupta, and R. Leisten. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(12):1243–1255, 2004.

J.M. Framinan, R. Leisten, and R. Ruiz-Usano. Comparison of heuristics for flowtime minimisation in permutation flowshops. *Computers & Operations Research*, 32(5):1237–1254, 2005.

M.R. Garey and D.S. Johnson. *Computers and intractability*. Freeman New York, 1979.

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

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

R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5(2):287–326, 1979.

M.P. Groover. *Automation, production systems, and computer-integrated manufacturing*. Prentice Hall Press, 2007.

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

J.C. Ho and J.N.D. Gupta. Flowshop scheduling with dominant machines. *Computers & Operations Research*, 22(2):237–246, 1995.

K.-L. Huang. *Flow shop scheduling with synchronous and asynchronous transportation times*. Ph.D. Thesis, The Pennsylvania State University, 2008.

K.-L. Huang and B.-W. Hung. Hybrid genetic algorithms for flowshop scheduling with synchronous material movement. pages 1–6, 2010.

S.M. Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

M. Kampmeyer. *Vehicle-Routing-Ansätze zur Lösung synchroner Flow-Shop-Probleme (in German)*. Master Thesis, University of Osnabrück, 2015.

S. Karabati and S. Sayin. Assembly line balancing in a mixed-model sequencing environment with synchronous transfers. *European Journal of Operational Research*, 149(2): 417–429, 2003.

J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

E.L. Lawler and J.M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969.

J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

J.Y.-T. Leung, H. Li, M. Pinedo, and J. Zhang. Minimizing total weighted completion time when scheduling orders in a flexible environment with uniform machines. *Information Processing Letters*, 103(3):119–129, 2007.

J.Y.T. Leung, H. Li, and M. Pinedo. Order scheduling models: an overview. In *Multidisciplinary scheduling: theory and applications*, pages 37–53. Springer, 2005.

D. Meignan, A. Koukam, and J.C. Créput. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6): 859–879, 2010.

C.E. Miller, A.W. Tucker, and R.A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

C.L. Monma and A.H.G. Rinnooy Kan. A concise survey of efficiently solvable special cases of the permutation flow-shop problem. *RAIRO Recherche Opérationelle*, 17:105–119, 1983.

J.M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109, 1968.

S. Murthy, R. Akkiraju, J. Rachlin, and F. Wu. Agent-based cooperative scheduling. In *Proceedings of AAAI Workshop on Constraints and Agents*, pages 112–117, 1997.

M. Nawaz, E.E. Enscore Jr, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

I.H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996.

Q.-K. Pan and R. Ruiz. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research*, 40(1):117–128, 2013.

B.J. Pine. *Mass customization: The new frontier in business competition*. Harvard Business School Press, Boston, MA, 1999.

M. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2012.

C.N. Potts and M.Y. Kovalyov. Scheduling with batching: a review. *European Journal of Operational Research*, 120:228–249, 2000.

J. Rachlin, R. Goodwin, S. Murthy, R. Akkiraju, F. Wu, S. Kumaran, and R. Das. A-teams: An agent architecture for optimization and decision-support. *Intelligent Agents V: Agents Theories, Architectures, and Languages*, pages 261–276, 1999.

H. Röck. Some new results in flow shop scheduling. *Mathematical Methods of Operations Research*, 28(1):1–16, 1984a.

H. Röck. The three-machine no-wait flow shop is np-complete. *Journal of the ACM*, 31 (2):336–345, 1984b.

R. Ruiz and C. Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494, 2005.

H. Sixiang, H. Hoogeveen, and P. Schuurman. A combinatorial property of pallet-constrained two machine flow shop problem in minimizing makespan. *Journal of Systems Science and Complexity*, 15(4), 2002.

M.L. Smith, S.S. Panwalkar, and R.A. Dudek. Flowshop sequencing problem with ordered processing time matrices. *Management Science*, 21(5):544–549, 1975.

W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

B. Soylu, Ö. Kirca, and M. Azizoğlu. Flow shop-sequencing problem with synchronous transfers and makespan minimization. *International Journal of Production Research*, 45 (15):3311–3331, 2007.

U. Stautner. *Kundenorientierte Lagerfertigung im Automobilvertrieb*. Deutscher Universitätsverlag, Wiesbaden, 2001.

S.M.A. Suliman. A two-phase heuristic approach to the permutation flow-shop scheduling problem. *International Journal of Production Economics*, 64(1):143–152, 2000.

E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

E. Taillard. Summary of best known lower and upper bounds of taillard's instances, April 2005. URL `http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/flowshop.dir/best_lb_up.txt`.

S. Talukdar, L. Baerentzen, A. Gove, and P. De Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4(4):295–321, 1998.

A. van den Nouweland, M. Krabbenborg, and J. Potters. Flow-shops with a dominant machine. *European Journal of Operational Research*, 62(1):38–46, 1992.

S. Waldherr and S. Knust. Two-stage scheduling in shelf-board production: A case study. *International Journal of Production Research*, 52:4078–4092, 2014.

S. Waldherr and S. Knust. Complexity results for flow shop problems with synchronous movement. *European Journal of Operational Research*, 242(1):34–44, 2015.

J.-B. Wang and Z.-Q. Xia. No-wait or no-idle permutation flowshop scheduling with dominating machines. *Journal of Applied Mathematics and Computing*, 17(1):419–432, 2005.

S. Xiang, G. Tang, and T.C.E. Cheng. Solvable cases of permutation flowshop scheduling with dominating machines. *International Journal of Production Economics*, 66(1):53–57, 2000.

A. Zimmermann. *Stochastic discrete event systems*. Springer, 2008.