Shop-Scheduling Problems with Transportation

Dissertation

im Fachbereich Mathematik/Informatik der

Universität Osnabrück

Sigrid Knust

Osnabrück, Juli 1999

Contents

1 Introduction

Over the last years problems arising in production planning have drastically changed due to modern technologies and computers. Fully automated manufacturing systems have enormously increased efficiency and flexibility in the production process. Typically, flexible manufacturing systems (FMS) consist of a relatively small number of versatile machines which can be used for various tasks. These machines are often grouped into cells in such a way that the whole production of a part can be performed within one of the cells. In these cells usually material handling systems (robots, automated guided vehicles) are available which transport parts from one machine to another. In order to utilize these automated facilities efficiently, the interaction between the machines and the transportation robots has to be carefully planned. However, in classical scheduling models only the processing of jobs on the machines is considered while transportation aspects are neglected. Thus, integrating transportation issues into classical scheduling models leads to more realistic models which may be used to solve modern practical problems.

Classical scheduling models for problems with multiple operations are the so-called shop-scheduling models. In these models jobs consisting of different operations have to be planned on certain machines in such a way that a given objective function is minimized (e.g. the completion time of the last operation or the number of late jobs when due dates are given). Each machine may process at most one operation at a time and operations belonging to the same job cannot be processed simultaneously. It is assumed that unlimited buffer space exists between the machines where jobs awaiting processing on the next machine are stored.

In this work we generalize these classical shop-scheduling problems by assuming that the jobs additionally have to be transported between the machines. We assume that this transportation has to be done by robots which can handle at most one job at a time. Besides transportation times which occur for the jobs during their transport, also empty moving times are considered which arise when a robot moves empty from one machine to another.

Mainly two types of problems can be distinguished: on the one hand, problems where no transportation conflicts occur, and on the other hand, problems with transportation conflicts. The first situation arises if sufficient transport robots are available and each transport can be performed without delay after processing. The second (more practical) case arises if we have only a limited number of robots and during certain time periods more jobs require transportation than free robots are available. In this situation we have to decide which jobs have to wait for their transportation and solving these conflicts becomes an important scheduling decision. In this work we mainly study problems of the second type. Besides several new complexity results for special cases with different objective functions we present an algorithm for a machine environment with a single robot where the makespan (i.e. the completion time of the last operation) is minimized.

Our work is organized as follows. In Section 2 we give a formulation of the general-shop problem with transportation times, which is the most general model we consider. We introduce basic notations and review the previous research on scheduling problems with transportation aspects. Shop problems without transportation conflicts (i.e. with sufficient robots) are investigated in Section 3. We extend the well-known disjunctive graph model for classical shop problems to these problems. Furthermore, an overview of known complexity results is given and some new results are derived. Additionally, we show that shop problems without transportation conflicts can be formulated as special cases of the general-shop problem with sequence-dependent setup times, which is another extension of the general-shop problem. In this model setup (changeover) times occur between operations which are processed consecutively on the same machine.

Problems with transportation conflicts (i.e. with a limited number of robots) are considered in Section 4. In this environment we do not only have to sequence the operations, but also have to decide on which robots the transportations are performed. The disjunctive graph model can be extended to this case as in Section 3 if an assignment of the robots to the different transport operations is fixed. Again, the problem can be formulated as another special case of the general-shop problem with sequence-dependent setup times.

In Section 5 we study several problems with a single robot according to their computational complexity and identify for special cases whether they are easy or hard. We show that only some very special cases in a flow-shop environment (here all jobs visit the machines in the same order) are easy, i.e. solvable in polynomial time. On the other hand, we prove that various special cases of flow-shop problems are already NP-hard, which means that it is very unlikely that they are polynomially solvable.

Thus, in the second part of this work our objective is to develop algorithms which deal with these hard problems in practice. We concentrate on the job-shop problem (a special shop problem where all job orders are fixed) with transportation and empty moving times and a single robot with the objective to minimize the makespan. Since scheduling the operations on the machines can be done in a similar way as in a classical job-shop problem without transportation, in Sections 6, 7, and 8 we concentrate on the robot. We study the subproblem which arises for the robot when some scheduling decisions for the machines have already been made. The resulting single-machine problem is very complex and may be considered as an interesting scheduling problem on its own. It can be regarded as a generalization of the traveling salesman problem with time windows where additionally minimal time-lags between certain jobs have to be respected and the makespan has to be minimized.

In Section 6 we describe how immediate selection techniques used for other scheduling problems can be adapted to this single-machine problem. Immediate selection (constraint propagation) procedures may be considered as preprocessing techniques which are used to reduce the search space and to direct algorithms towards good solutions. Additionally, these techniques can be used to calculate lower bounds for the optimal objective value, which is described in Section 7. There we also present another lower bound procedure which is based on linear programming. Since our linear programming formulation contains a huge number of columns, we cannot solve the linear programs directly. Therefore, we develop a procedure which is able to deal with such large linear programs using the so-called technique of delayed column generation.

Section 8 is devoted to the calculation of heuristic solutions for the single-machine problem. Besides a fast heuristic based on priority rules a tabu search algorithm is developed which iteratively searches for a good solution exploiting problem-specific properties of the single-machine problem. This algorithm for scheduling a single robot is used in Section 9, where it is integrated into a tabu search algorithm for the job-shop problem with transportation and a single robot. This algorithm is organized in two stages: while in the outer stage scheduling decisions for the machines are changed, in the inner stage the robot is optimized for the current machine situation.

Implementational details and computational results for the proposed algorithms can be found in Section 10. On the one hand, lower and upper bounds are calculated for various single-machine test instances arising from a job-shop environment with a single transport robot. On the other hand, our two-stage tabu search procedure is tested on several job-shop instances with different transport characteristics. Since no lower bounds are available for these instances, we compare our upper bounds with the results obtained by an existing procedure for the more general general-shop problem with setup times. We conclude this work with some final remarks including topics for further research.

2 The general-shop problem with transportation times

In this section we describe the general-shop problem with transportation times. It is a generalization of the classical general-shop problem (cf. Brucker [1998]) where not only the jobs on the machines have to be scheduled, but also the transportation of the jobs between the machines has to be considered. After formulating the problem we classify problems with transportation times and give elementary reductions between special cases. Related models for scheduling problems with transportation times which have been proposed in the literature are discussed in Subsection 2.3.

2.1 Problem formulation

At first we consider the classical general-shop problem which may be described as follows. We are given m machines $M_1, \ldots, M_m$ and n jobs $J_1, \ldots, J_n$. Each job J_j consists of n_j operations O_{ij} $(i = 1, \ldots, n_j)$. Operation O_{ij} has to be processed without preemption on a dedicated machine $\mu_{ij} \in \{M_1, \ldots, M_m\}$ for $p_{ij} \geq 0$ time units. It is allowed that operations of the same job (also consecutive operations) are assigned to the same machine. Each machine can only process one operation at a time and a job cannot be processed by more than one machine simultaneously. Between the operations arbitrary precedence constraints may be given. In this context $O_{ij} \rightarrow O_{uv}$ means that operation O_{ij} has to be finished completely before O_{uv} starts processing. The objective is to find a feasible schedule which minimizes some objective function depending on the completion times C_j of the jobs J_j, where the completion time C_j of a job J_j is defined to be equal to the completion time of the operation belonging to J_j, which is finished last. Additionally, release dates r_j or due dates d_j may be given for the jobs; weights w_j may be used in defining the objective function.

In the following we will generalize the general-shop problem by additionally considering **transportation times**. They occur if a job changes from one machine to another, i.e. if job J_j is processed on machine M_k and afterwards on machine M_l, a transportation time t_{jkl} arises. These transportation times may be job-dependent or job-independent ($t_{jkl} = t_{kl}$). We assume that the transportation is done by **transport robots** (automated guided vehicles) which can handle at most one job at a time.

As in classical shop problems we assume that sufficient buffer space exists between the machines. This means that each machine M_k has an unlimited output buffer where jobs processed on M_k and waiting for a robot may be stored. The jobs are automatically transferred into this buffer and no further times for this transfer are considered. Additionally, each machine M_l has an unlimited input buffer where jobs which have been transported and await processing on M_l may be stored.

We suppose $t_{jkk} = 0$ for all jobs J_j and all machines M_k. This means that if two consecutive operations of a job J_j are processed on the same machine, no transport is necessary between them. Furthermore, the transportation times are supposed to satisfy the following triangle inequality for all jobs J_j and all machines M_k, M_l, M_h:

$$t_{jkh} + t_{jhl} \geq t_{jkl}. \tag{2.1}$$

If this inequality does not hold, i.e. $t_{jkh} + t_{jhl} < t_{jkl}$ for some indices j, k, h, l, we could save time for the transport from M_k to M_l by first transporting job J_j from M_k to M_h and then to M_l. In practice, this situation is unlikely to occur, hence this assumption is not a real restriction.

We assume that there are r identical transport robots $R_1, \ldots, R_r$ and each job can be transported by any of these robots. Then the following two cases can be distinguished:

- an **unlimited number of robots** $r \geq n$,
- a **limited number of robots** $r < n$.

If we have an unlimited number of robots, for each of the n jobs J_j a robot exists which can do all transportations concerning job J_j. This means that each job can be transported immediately after finishing on a machine and no conflicts arise between different transport operations. Thus, we do not have to decide which robot transports which job and the transportation times can be modeled by minimal time-lags between operations of the same job. In the second case not only the machines, but also the robots are a resource with limited capacity. Since a robot can transport only one job at a time, situations may occur in which more jobs require transportation than free robots available. Then it has to be decided which robot transports which job and which jobs have to wait for a free robot before transportation.

In a further generalization of our described model the robots $R_1, \ldots, R_r$ may be non-identical and may have different characteristics (so-called multi-purpose robots). Then it can be possible that some jobs cannot be transported by any of the robots. This more general situation is modeled by associating with each job J_j a set $\mu_j^R \subseteq \{R_1, \ldots, R_r\}$ containing all robots by which J_j may be transported.

In the case of a limited number of robots we consider **empty moving times** t'_{kl} in addition to the transportation times. While the transportation times arise when a job is transported from one machine to another, the empty moving times arise when an empty robot moves from machine M_k to M_l without carrying a job. For these times we assume

$$t'_{kk} = 0, \quad t'_{kh} + t'_{hl} \geq t'_{kl} \text{ and } t'_{kl} \leq \min_{j=1}^{n} \{t_{jkl}\} \tag{2.2}$$

for all machine indices $k, h, l \in \{1, \ldots, m\}$. The first assumption means that no empty moving times have to be considered if the robot waits at a machine for the next transport. The second condition is again a triangle inequality and the third states that moving empty does not take longer than moving a job. In practical situations these assumptions are satisfied in most cases. Due to the empty moving times the scheduling decisions strongly depend on the routing of the robots. In order to avoid waiting times for the jobs the robots should be scheduled in such a way that they do not waste much time by empty moves. All data p_{ij}, t_{jkl}, t'_{kl} are assumed to be non-negative integers.

2.2 Classification and elementary reductions

For classifying different shop problems with transportation times we extend the $\alpha|\beta|\gamma$-notation for scheduling problems introduced by Graham et al. [1979]. Furthermore, we give elementary reductions between special cases.

Machine environment

In the α-field, indicating the machine environment, the classical shop problems are classified by a string $\alpha = \alpha_1, \alpha_2$ with the following meaning:

- $\alpha_1 = G$ indicates a general-shop as described in Subsection 2.1.
- $\alpha_1 = O$ indicates an open-shop. Each job J_j consists of $n_j = m$ operations and operation O_{ij} has to be processed on machine $\mu_{ij} = M_i\,(i = 1, \ldots, m)$. No precedence constraints are given.
- $\alpha_1 = J$ indicates a job-shop. The numbers n_j may differ for the jobs but for each job the order in which its operations are processed is fixed due to the chain precedence constraints $O_{1j} \to O_{2j} \to \ldots \to O_{n_j,j}$.
- $\alpha_1 = F$ indicates a flow-shop. This is a special case of a job-shop where each job J_j consists of $n_j = m$ operations and operation O_{ij} has to be processed on machine $\mu_{ij} = M_i\,(i = 1, \ldots, m)$.

If α_2 is equal to a positive integer $1, 2, \ldots$, then α_2 denotes the number of machines. If $\alpha_2 = m$, then m is an arbitrary, but fixed number of machines which is not part of the input. Entry α_2 is empty if the number of machines is arbitrary.

We extend the string α by two additional components, α_3, α_4, which describe the robot environment. If we have an unlimited number of identical robots, entries α_3 and α_4 are empty. If we have a limited number of identical robots and all jobs can be transported by any of the robots, we set $\alpha_3 = R$. The more general case, in which for each job a set μ_j^R of robots is given (multi-purpose robots), is indicated by $\alpha_3 = MPR$. If all sets μ_j^R contain only one element, we have the situation of dedicated robots and write $\alpha_3 = DR$. A positive integer $1, 2, \ldots$ in entry α_4 denotes the number of robots. If $\alpha_4 = r$, the number of robots is limited to an arbitrary, but fixed number r which is not part of the input. If the number of robots is limited and part of the input, entry α_4 is empty.

Job characteristics

In the β-field, indicating the job characteristics, we use the following entries $\beta_1, \dots, \beta_6$ (cf., e.g. Brucker [1998]):

- $\beta_1 = pmtn$ indicates that preemption is allowed, i.e. processing of a job or an operation may be interrupted and continued at a later time. If preemption is not allowed, entry β_1 is set to the empty symbol $\circ$.
- $\beta_2 = prec$ describes that precedence constraints between the jobs are given. If the precedence relations are restricted to a special structured graph (like chains, intree, outtree, ...), β_2 is set to the corresponding entry. If there are no precedence constraints, β_2 does not appear in β.
- β_3 specifies restrictions on the processing times and on the number of jobs or operations. If all processing times are equal to a constant p, we write $p_{ij} = p$; $p_{ij} = 1$ indicates that all operations have unit processing times. If we set β_3 to $n = k$, we have a fixed number k of jobs. The entry $n_j \leq k$ means that all jobs have at most k operations.
- $\beta_4 = r_j$ indicates that release dates are given for the jobs, i.e. a job cannot start processing before time r_j. If $r_j = 0$ for all jobs, β_4 is empty.
- $\beta_5 = d_j$ indicates that deadlines are given for the jobs, i.e. job j must finish not later than time d_j.
- $\beta_6 = s_{fg}$ indicates that sequence- and group-dependent setup times are given between the operations. All operations are divided into groups $G_1, \dots, G_q$ and a setup time s_{fg} occurs if an operation from group G_g is processed immediately after an operation from group G_f on the same machine. If all groups contain only one element, we write $\beta_6 = s_{ij}$. If all setup times are zero, β_6 is empty.

We extend the string β by two additional components, β_7, β_8, which describe the transportation and empty moving times. For the transportation times we distinguish the following eight cases in β_7:

- t_{jkl} indicates that we have job- and machine-dependent transportation times,
- $t_{jkl} = t_{jlk}$ indicates that we have job- and machine-dependent, symmetrical transportation times,
- t_j indicates that we have job-dependent, but machine-independent transportation times,
- $t_j \in \{T_1, T_2\}$ indicates that we have job-dependent, but machine-independent transportation times, which may take only two values T_1 and T_2,

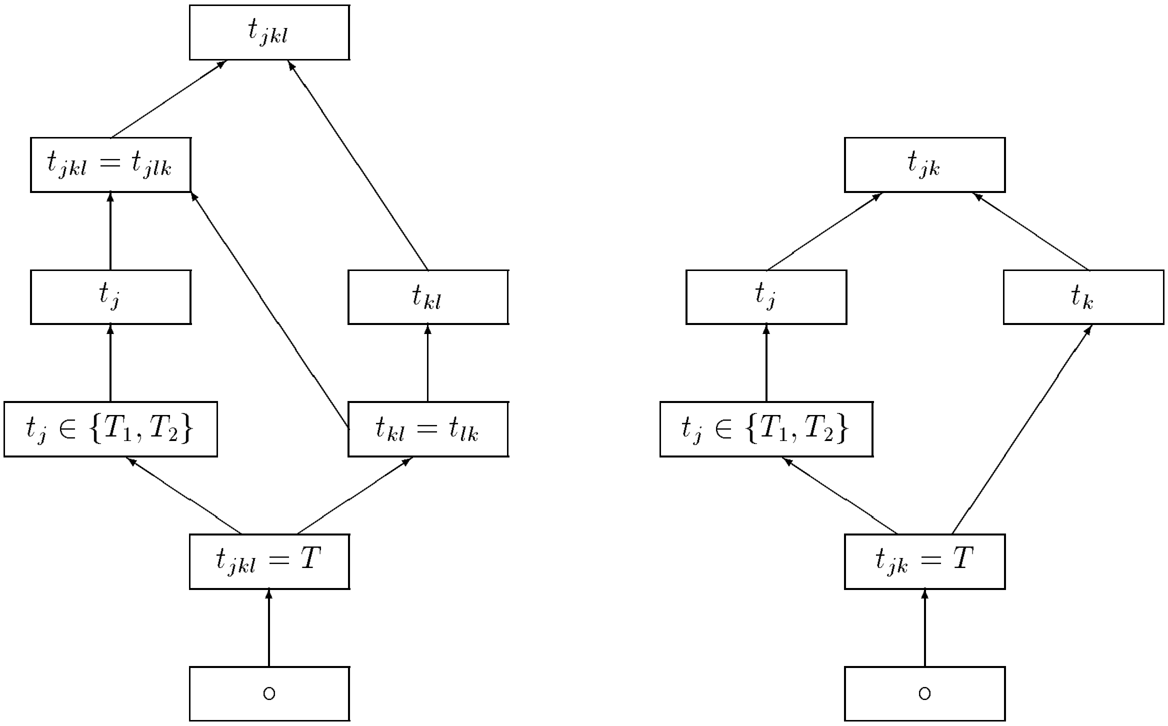


Figure 2.1: Reduction graphs for general-shops and flow-shops

- t_{kl} indicates that we have job-independent, but machine-dependent transportation times,
- $t_{kl} = t_{lk}$ indicates that we have job-independent, but machine-dependent symmetrical transportation times,
- $t_{jkl} = T$ indicates that all transportation times are equal to a constant T,
- the empty symbol $\circ$ indicates that all transportation times are 0.

In the case of flow-shops this general notation can be simplified since a job is always transported from machine M_k to machine M_{k+1} $(k = 1, \ldots, m-1)$. Therefore, we only have transportation times of the form t_{jk}, i.e. the second machine index l can be dropped. If we have only $m = 2$ machines, the robot always transports from M_1 to M_2. Therefore, the index k in the notation t_{jk} may also be dropped and the transportation times are simply denoted by t_j.

Elementary reductions for the transportation times in general-shops and flow-shops can be found in Figure 2.1.

For the empty moving times we distinguish three cases in β_8:

- t'_{kl} indicates that we have machine-dependent empty moving times,
- $t'_{kl} = T'$ indicates that all empty moving times are equal to a constant T',
- the empty symbol $\circ$ indicates that all empty moving times are 0.

Since empty moving times can be neglected in the case of an unlimited number of identical robots, β_8 only appears in β if we have $\alpha_3 \in \{MPR, R, DR\}$. Elementary reductions for the empty moving times can be found in Figure 2.2.

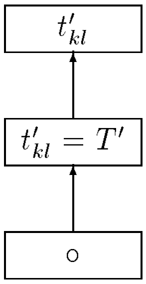


Figure 2.2: Reduction graph for empty moving times

Optimality criterion

In the third field γ an optimality criterion is indicated. Associated with each job J_j is a function f_j which depends on the completion time C_j of job J_j. Mainly two types of objective functions are considered:

- bottleneck objective functions $F = f_{\max} = \max\limits_{j=1}^{n} \{f_j(C_j)\}$,
- sum objective functions $F = \sum f_j = \sum\limits_{j=1}^{n} f_j(C_j)$.

The most common objective functions are

- the makespan $C_{\max} = \max\limits_{j=1}^{n} C_j$,
- the sum of completion times $\sum C_j$,
- the weighted sum of completion times $\sum w_j C_j$,
- the maximum lateness $L_{\max} = \max\limits_{j=1}^{n} \{C_j - d_j\}$, which is equivalent to the extended makespan $\max\limits_{j=1}^{n} \{C_j + q_j\}$ where delivery times q_j are considered for the jobs, i.e. job j stays for q_j time units in the system after processing,

- the weighted number of tardy jobs $\sum w_j U_j$ with $U_j = \begin{cases} 0, & C_j \le d_j \\ 1, & \text{otherwise} \end{cases}$,

- the total weighted tardiness $\sum w_j T_j = \sum_{j=1}^{n} w_j \max\{0, C_j - d_j\}$.

If the functions f_j are not specified, we set $\gamma = f_{\max}$ or $\gamma = \sum f_j$. An objective function $F = f_{\max}$ or $F = \sum f_j$ is called **regular** if all functions f_j are monotone non-decreasing with respect to the completion times C_j.
If deadlines d_j are specified, sometimes no objective function has to be minimized. Then only the problem is considered whether a feasible schedule exists or not, which is indicated by $\gamma = -$.

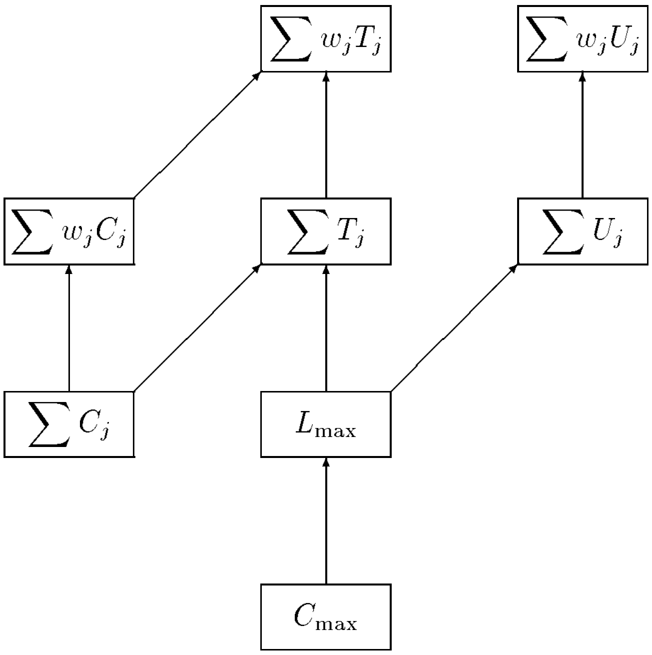


Figure 2.3: Reduction graphs for objective functions

Elementary reductions between the objective functions (cf. Brucker [1998]) are shown in Figure 2.3.

To illustrate the extended $\alpha|\beta|\gamma$-notation we present some examples:

- $G \mid t_{jkl} \mid C_{\max}$ denotes the general-shop problem with an unlimited number of identical robots and job- and machine-dependent transportation times, where the makespan has to be minimized,

- $J, MPR \mid t_{jkl}, t'_{kl} \mid C_{\max}$ denotes the job-shop problem with multi-purpose robots and arbitrary transportation and empty moving times,

- $F3, R1 \mid p_{ij} = p, r_j, t_j \mid \sum C_j$ denotes a flow-shop problem with $m = 3$ machines, transportation times and a single robot where the sum of completion times has to be minimized. All operations have the same processing time p and release dates are given for the jobs. The transportation times are machine-independent, but job-dependent. Empty moving times are not considered. An instance with $n = 3$ jobs and a corresponding feasible schedule, are presented in Figure 2.4. Note that the robot R is drawn three times, first for the transportations between machines M_1 and M_2, secondly for the transportations between M_2 and M_3, and finally, the whole schedule including both transportation stages together. The objective value of the schedule is $\sum_{j=1}^{n} C_j = C_1 + C_2 + C_3 = 14 + 21 + 24 = 59$.

j	1	2	3
r_j	0	2	9
t_j	1	4	2

$p_{ij} = p = 3$

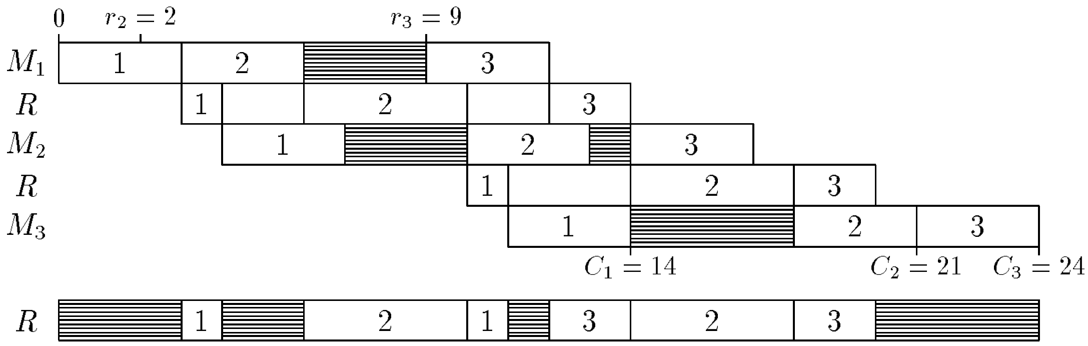


Figure 2.4: Instance and schedule for $F3, R1 \mid p_{ij} = p, r_j, t_j \mid \sum C_j$

2.3 Previous research

Scheduling problems with transportation aspects have received much attention in the literature, especially in recent years. Although there are many differences between the various models, they all deal with the interaction between the material handling and the classical job scheduling decisions. In this subsection we review the most important research concerning combined models for scheduling with transportation times.

An important issue in which the models differ is the existence or non-existence of buffers. In this work we assume that unlimited (input and output) buffer space exists between the machines where jobs may wait for a free robot or machine. If, on the other hand, the number of buffer units is limited (or even zero), jobs may have to wait on the last visited machine until a free robot becomes available. During this time period the machine is blocked, i.e. no other job can be processed on it. The same situation may occur before a machine with limited input buffer. If a transported job arrives at a machine which is still processing another part and the input buffer is full, the job has to wait on the robot until the machine has finished with the previous part. If no waiting times are allowed between transportation and processing, we get problems with an additional no-wait restriction. For an overview on classical shop problems with blocking or no-wait restrictions see Hall & Sriskandarajah [1996].

Shop problems with transportation times (delays) and an unlimited number of identical robots, or equivalently shop problems with minimal time-lags, have been studied by several authors. For flow-shop and open-shop problems complexity results have been derived by Mitten [1958], Yu [1996], Dell' Amico [1996], and Rayward-Smith & Rebaine [1992]. These results are summarized in more detail in Section 3.2. Approximation algorithms for job-shops and open-shops have been obtained by Dell' Amico [1996], Strusevich [1999] and Rebaine & Strusevich [1998].

Only few results are available for shop problems with transportation times, unlimited buffer space and a limited number of identical robots. Kise [1991] proved that minimizing the makespan in a two-machine flow-shop with constant transportation times and a single robot is already $\mathcal{NP}$-hard. King et al. [1993] proposed a branch-and-bound algorithm for a flow-shop environment with a single robot. Langston [1987] derived some approximation algorithms for a flexible flow-shop environment with two stages and interstage transportation times. Bilge & Ulusoy [1995] proposed a heuristic for simultaneously scheduling the machines and vehicles in a flexible manufacturing system with job-shop structure.

For flow-shop problems with transportation times, limited buffer space and a single robot some polynomial algorithms exist for special cases. Panwalkar [1991] and Levner et al. [1995] treated the two-machine case without output buffer behind the first machine and showed that this problem can be solved by considering n classical $F2 \mid\mid C_{\max}$ problems. Kise et al. [1991] and Stern & Vitner [1990] considered the two-machine case with an additional no-wait condition which can be formulated as a specially structured traveling salesman problem.

A class of problems which has received much attention in the literature during the last few years is the problem of cyclic scheduling in robotic cells without any buffers (for an overview cf. Crama et al. [1997]). A robotic flow-shop consists of m machines, an input and an output station and several robots. In the input station all parts to be processsed are stored which have to visit the machines in the same order until they are delivered to the output station. The robots perform all material handling operations in the cell, i.e. they load and unload the parts as well as transport them from one machine to the next. Contrary to classical shop problems no buffers exist between the machines, i.e. any part in the cell is either on a machine or carried by a robot. This implies that the parts have to be processed in the same order on all machines. The objective is to determine such an ordering of the parts, a sequence of robot moves and a corresponding schedule in such a way that a given objective function is minimized. In most cases repetitive manufacturing for a minimal part set is considered with the objective to minimize the long-run average cycle time or equivalently to maximize the throughput rate of the cell. Such problems with a single robot and job-independent transportation and loading times have been studied by many authors (e.g. Sethi et al. [1992], Hall et al. [1997], van de Klundert [1996], Crama & van de Klundert [1997], Sriskandarajah et al. [1998]).

Sethi et al. [1992] and Hall et al. [1997] concentrated on special cases with two or three machines. On the one hand, they considered the case of identical parts, where the part sequencing problem vanishes and it is sufficient to determine an optimal robot cycle. On the other hand, for multiple part types they considered pure part sequencing problems which arise for a fixed robot cycle. For some cycles the corresponding part sequencing problem is polynomially solvable, for others it becomes $\mathcal{NP}$-hard. The only problem where a polynomial algorithm has been found for simultaneously optimizing the part input sequence and the robot cycle is the two-machine case with identical parts.
Most investigations concentrate on so-called one-unit cycles, i.e. sequences of robot moves where exactly one part enters the system from the input station and one part leaves the system to the output station. These cycles can be infinitely repeated and are easy to implement. Much attention has been paid to the conjecture of Sethi et al. [1992] which claims that for identical parts the repetition of the best one-unit cycle reaches the optimum in the set of all robot moves. The conjecture is valid for $m = 2$ (Sethi et al. [1992]) and $m = 3$ machines (Crama & van de Klundert [1997], Brauner & Finke [1999]), but recently Brauner & Finke [1998] found a counterexample with $m = 4$ machines where a three-unit cycle performs better than the best one-unit cycle.

Robotic flow-shops with a single unit of buffer behind each machine have been studied by Finke et al. [1996]. They showed that in this environment the conjecture is true for an arbitrary number of machines. Robotic flow-shops with a no-wait condition have been considered by Kats & Levner [1997] and Agnetis [1997]. Kats & Levner [1997A] proposed an algorithm which optimizes the number of used robots and the cycle time simultaneously.

We conclude this overview by describing some other models in which robots are important for the scheduling decisions. Agnetis [1996] studied a problem in a flexible assembly system of assigning operations to machines with the two objectives of balancing the machine workloads and minimizing the total transportation time. For a special flexible manufacturing cell Błażewicz et al. [1994] proposed routing strategies for vehicles which are allowed to move on two track cycles with a common stretch. Their goal is to schedule the vehicles for a given machine schedule in such a way that no collisions between different vehicles arise on their common stretch and some objective function is minimized.

Another scheduling environment in which a robot is used are parallel machines with a so-called common server. In this environment jobs do not have to be transported from one machine to another, but have to be loaded onto the machines before processing. These setups are all done by a single server and cannot occur while a machine is processing another job. Many complexity results have been derived by Hall et al. [1994]. Glass et al. [1995] studied the same model in an environment with parallel dedicated machines.

3 Problems without transportation conflicts

In this section we study shop problems with transportation times where no conflicts between different transportations arise. This situation occurs if we have an unlimited number $r \geq n$ of identical robots $R_1, \ldots, R_r$, i.e. for each job a transport robot is always available. In Subsection 3.1 we extend the disjunctive graph model to the general-shop problem with transportation times. Then we introduce another generalization of the general-shop problem where sequence-dependent setup times are given. It is shown that the general-shop problem with transportation times can be formulated as a special case of this problem. Finally, in Subsection 3.2 complexity results are presented.

3.1 Extension of the disjunctive graph model

In this subsection the well-known disjunctive graph model for the classical job-shop problem developed by Roy & Sussmann [1964] is extended to the general-shop problem $G \mid t_{jkl} \mid C_{\max}$ with transportation times and an unlimited number of robots. Disjunctive graphs can be used to represent certain feasible schedules for general-shop problems. If the objective function is regular, the set of feasible schedules represented in this way always contains an optimal solution. Thus, the disjunctive graph may be used to construct optimal schedules. In order to simplify the presentation we number all operations O_{ij} ($j = 1, \ldots, n$; $i = 1, \ldots, n_j$) consecutively from 1 to $N := \sum_{j=1}^{n} n_j$. For each operation $u \in \{1, \ldots, N\}$ let $J(u)$ be the index j of the associated job J_j and let $M(u)$ be the machine on which u has to be processed. For a given instance of the general-shop problem with transportation times the **disjunctive graph** $G = (V, C \cup D_J \cup D_M)$ is defined as follows.

- The set of vertices $V := \{1, \ldots, N\} \cup \{0, *\}$ represents all operations of the jobs. Additionally, there are two dummy nodes, a source 0 and a sink $*$. These two special nodes are used to define the start and the end of the operations. The operations u are weighted with their processing times p_u; the weights of the dummy nodes are 0.

- The set of directed arcs C represents the given precedence constraints $\rightarrow$ and is called the set of **conjunctions**. We have an arc $u \rightarrow v \in C$ if operation u has to be finished before v starts processing. These arcs are weighted with

$$w_{uv} = \begin{cases} t_{jkl}, & \text{if } J(u) = J(v) =: j \\ 0, & \text{otherwise} \end{cases}, \text{ where } M_k := M(u),\ M_l := M(v).$$

  Furthermore, we have arcs $0 \rightarrow u$ between the source 0 and all operations u without any predecessor and arcs $u \rightarrow *$ between all operations u without successor and the sink $*$. All these arcs are weighted with 0.

- The set of undirected edges D_J consists of all pairs of operations which belong to the same job and which are not linked by a conjunction. These pairs are called **job disjunctions**. An edge $u - v \in D_J$ represents the two possible orders in which the operations u and v may be processsed and is weighted with the pair (t_{jkl}, t_{jlk}), where $j := J(u) = J(v)$ and $M_k := M(u)$, $M_l := M(v)$.

- The set of undirected edges D_M consists of all pairs of operations which have to be processed on the same machine and which are not linked by a conjunction or a job disjunction. These pairs are called **machine disjunctions**. An edge $u - v \in D_M$ with $M(u) = M(v) = M_k$ represents the two possible orders in which the operations u and v may be processsed on M_k and is weighted with 0.

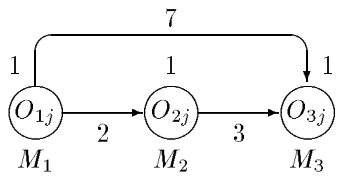
To solve the scheduling problem we have to find an optimal ordering for all operations belonging to the same job and an optimal ordering for all operations processed on the same machine respecting the given precedence constraints. This can be done by turning all undirected edges in $D_J \cup D_M$ into directed arcs. Concerning an edge in D_J weighted with a pair of transportation times, at this point the directed arc gets the corresponding unique weight. If we orient an edge $u - v$ with $j := J(u) = J(v)$ and $M_k := M(u)$, $M_l := M(v)$ in the direction $u \to v$, it gets the weight t_{jkl}, and if we orient it in the other direction, it gets the weight t_{jlk}. Disjunctions which have been directed are called **fixed**. A set $S = S_J \cup S_M$ of fixed job disjunctions S_J and fixed machine disjunctions S_M is called a **selection**. A selection S is called **complete** if

- all disjunctions in $D_J \cup D_M$ have been fixed, and

- the resulting graph $G(S) = (V, C \cup S_J \cup S_M)$ is acyclic.

Given a complete selection S we may construct a corresponding feasible schedule defined by the starting times of all operations as follows. For each operation u let $\ell(u)$ be the length of a longest path from 0 to u, where the length of a path is defined as the sum of the weights of all nodes and arcs on the path, node u excluded. We consider the schedule where each operation is started at time $\ell(u)$. Obviously, this schedule is feasible. We also claim that for this selection no operation can be started before time $\ell(u)$, i.e. no operation can be shifted to the left without changing the orders defined by S. Since the transportation times t_{jkl} only occur between operations of the same job which are processed consecutively, all transitive job arcs (these are arcs $u \xrightarrow{t_{jkl}} v$ where an operation w exists with $u \xrightarrow{t_{jkh}} w \xrightarrow{t_{jhl}} v$ and u, v, w belong to the same job) are superfluous and their weights must not be taken into account. However, due to the triangle inequality $t_{jkh} + t_{jhl} \geq t_{jkl}$ in (2.1) these transitive arcs do not change the longest path lengths $\ell(u)$. If the triangle inequality is not satisfied, the transitive arcs have to be removed from the disjunctive graph before the longest paths are calculated, since otherwise they may increase the longest path lengths. In this case feasible schedules may exist in which an operation u can be started before time $\ell(u)$, which implies that the longest path procedure cannot be used to construct optimal schedules.

Consider, for example, one job J_j consisting of 3 unit time operations O_{ij} with $\mu_{ij} = M_i\ (i = 1, 2, 3)$. Let $t_{j12} = 2, t_{j23} = 3, t_{j13} = 7$, i.e. the triangle inequality is violated.



With the orientation $O_{1j} \to O_{2j} \to O_{3j}$ operation O_{3j} can be started at time $p_{1j} + t_{j12} + p_{2j} + t_{j23} = 7$, but the path $0 \to O_{1j} \to O_{3j}$ has the length $p_{1j} + t_{j13} = 8$. Thus, a feasible schedule exists where O_{3j} can be started before time $\ell(O_{3j}) = 8$.

But, if inequality (2.1) is fulfilled, this situation cannot occur and each complete selection defines a feasible schedule, in which no operation can be shifted to the left. On the other hand, each feasible schedule defines an order of all operations belonging to the same job or processed on the same machine. These orders induce a complete selection and the corresponding schedule is not worse if the objective function is regular. Hence, in the set of schedules represented by complete selections an optimal solution always exists. The completion time $C_{\max}(S)$ of the schedule corresponding to a selection S is given by the length of a longest path from 0 to $*$ in the acyclic graph $G(S)$. Such a path is also called a **critical path**.

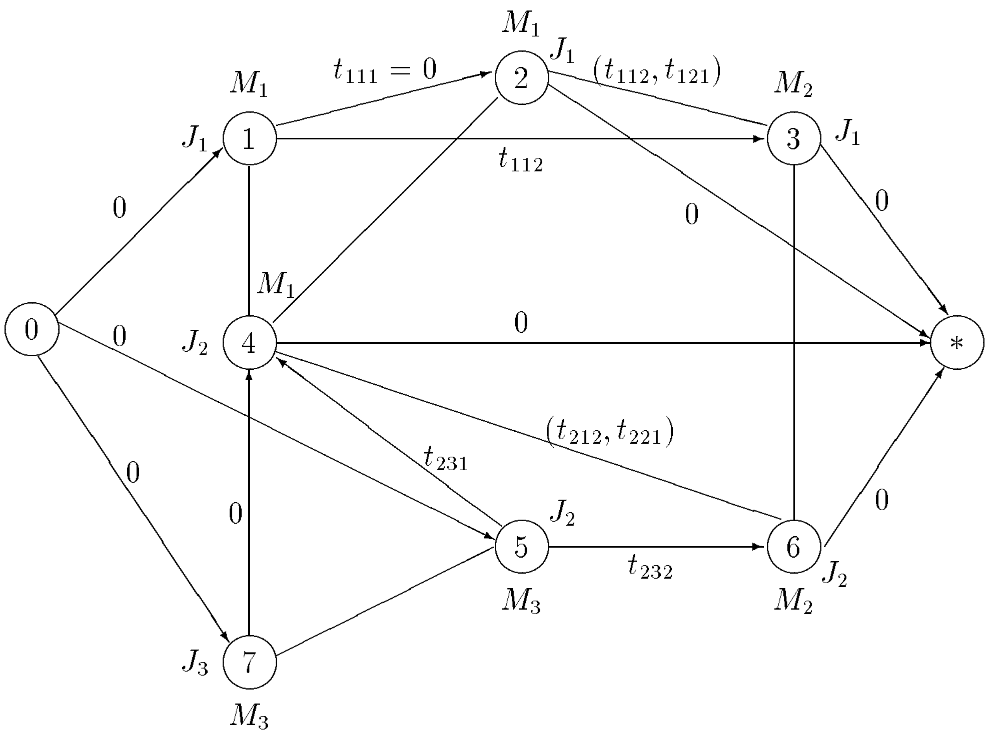
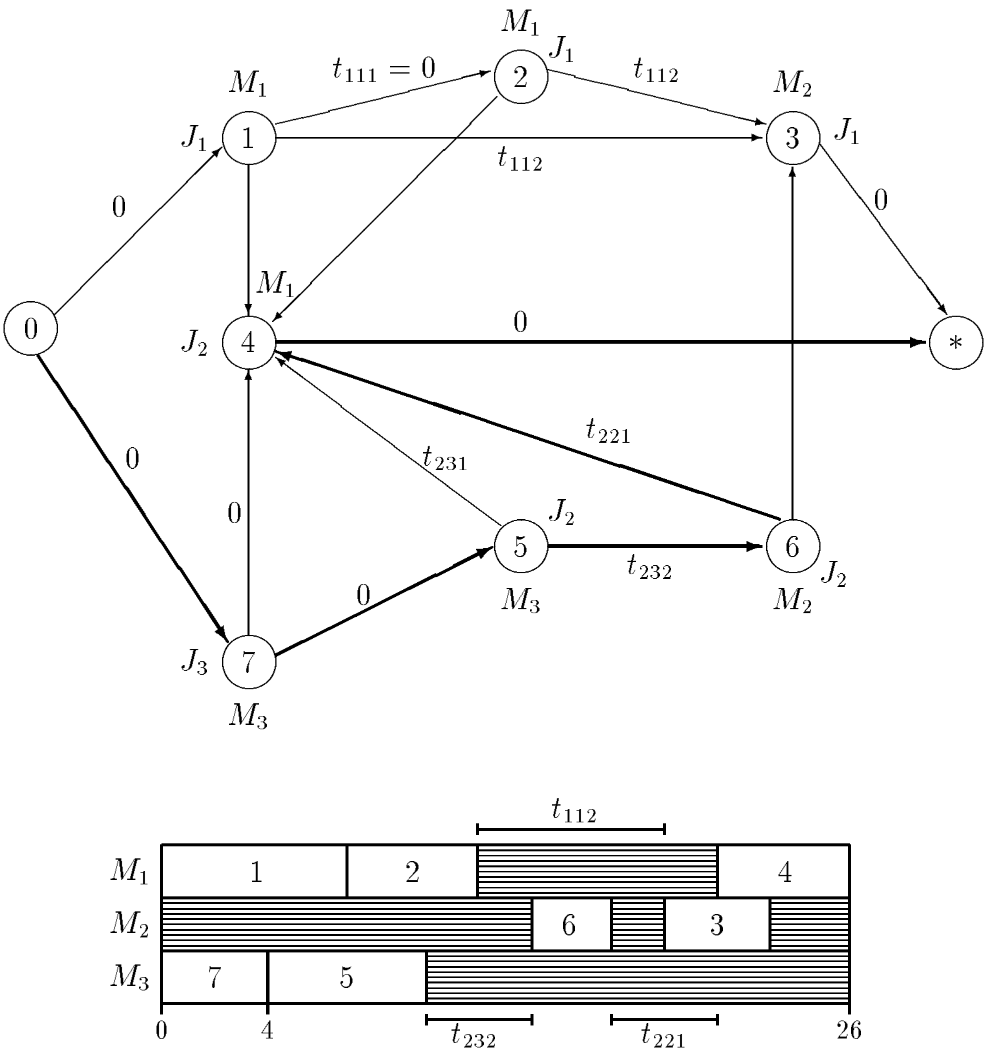


Figure 3.1: Disjunctive graph for a general-shop with $n = 3$, $m = 3$ and t_{jkl}

Example 3.1 : We consider a problem with $n = 3$ jobs, $m = 3$ machines and job- and machine-dependent transportation times t_{jkl}. The corresponding disjunctive graph can be found in Figure 3.1. We have conjunctions $1 \to 2, 1 \to 3, 5 \to 4, 5 \to 6$ and $7 \to 4$. Since operations 1 and 3 belong to the same job J_1, this arc is weighted with the corresponding transportation time t_{112}. The job disjunctions $2-3$ and $4-6$ are weighted with pairs of transportation times because these orders are not fixed. Furthermore, we have machine disjunctions $1-4, 2-4$ on M_1, $3-6$ on M_2 and $5-7$ on M_3. In Figure



u	1	2	3	4	5	6	7
p_u	7	5	4	5	6	3	4

$$t_{112} = 7, \quad t_{221} = 4, \quad t_{231} = 3, \quad t_{232} = 4.$$

Figure 3.2: Complete selection with a corresponding schedule

3.2 a complete selection and a corresponding schedule are shown for the given instance. All undirected arcs have been oriented in such a way that the resulting graph is acyclic. For example, the job disjunction $2-3$ is turned into the directed arc $2 \rightarrow 3$, which gets the weight t_{112}. For the given data a critical path is $P = (0 \rightarrow 7 \rightarrow 5 \xrightarrow{t_{232}} 6 \xrightarrow{t_{221}} 4 \rightarrow *)$ with length $p_7 + p_5 + t_{232} + p_6 + t_{221} + p_4 = 26$. □

Finally, we introduce the general-shop problem with sequence-dependent setup times and show that problem $G \mid t_{jkl} \mid C_{\max}$ with transportation times and an unlimited number of robots can be formulated as a special case of this problem.

The general-shop problem $G \mid s_{fg} \mid C_{\max}$ with sequence-dependent setup times has been considered by Brucker & Thiele [1996] and Küsters [1997]. In this extension of the general-shop problem the set of operations is partitioned into q disjoint groups $G_1, \ldots, G_q$. If on a machine an operation from group G_g is processed immediately after an operation from group G_f, a setup time s_{fg} occurs between these two operations. It is assumed that the weak triangle inequality

$$s_{fh} + \min_{u \in G_h} p_u + s_{hg} \geq s_{fg} \tag{3.1}$$

holds for all $f, g, h \in \{1, \ldots, q\}$, which again ensures that transitive arcs in the disjunctive graph do not change the longest path lengths. In the algorithm of Brucker & Thiele [1996] it is assumed that $s_{ff} = 0$ and even the strong triangle inequality $s_{fh} + s_{hg} \geq s_{fg}$ holds, which simplifies some calculations.

While in the model with setup times changeover times occur between operations which are processed on the same machine and belong to different groups, in the model with transportation times and an unlimited number of robots changeover times occur between operations which belong to the same job and are processed on different machines. Thus, an instance of the general-shop problem $G \mid t_{jkl} \mid C_{\max}$ with transportation times and an unlimited number of robots can be transformed into an instance of the general-shop problem $G \mid s_{fg} \mid C_{\max}$ with sequence-dependent setup times in the following way:

- interchange the role of jobs and machines,
- partition the operations $u = 1, \ldots, N$ into $n \cdot m$ groups $G_{(j,k)}$ ($j = 1, \ldots, n$; $k = 1, \ldots, m$) with

$$u \in G_{(j,k)} \text{ if and only if } J(u) = j \text{ and } M(u) = M_k,$$

- define the setup times between the groups by the corresponding transportation times:

$$s_{(j,k)(i,l)} = \begin{cases} t_{jkl}, & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases}$$

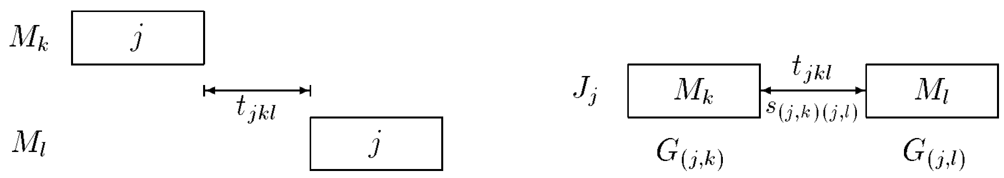


Figure 3.3: Situation of transportation and corresponding setup times

It is easy to see that the groups and setup times defined in this way model the same situation as when transportation times are given (see Figure 3.3). Due to inequality (2.1) the defined setup times satisfy for all groups the strong triangle inequality

$$s_{(j,k)(i,l)} + s_{(i,l)(h,v)} \geq s_{(j,k)(h,v)}.$$

In the case of job-independent transportation times t_{kl} it is sufficient to partition the operations into only m groups G_k ($k = 1, \ldots, m$), where $u \in G_k$ if $M(u) = M_k$. Then the setup times are defined by $s_{kl} = t_{kl}$.

Example 3.2: We consider an open-shop problem with $n = 4$ jobs, $m = 3$ machines, and job-independent, but machine-dependent transportation times t_{kl}. The corresponding disjunctive graph can be found in Figure 3.4. We have 3 groups G_k ($k = 1, 2, 3$), each consisting of the $n = 4$ operations processed on machine M_k. The matrix of setup times for the three groups is

$$s = \begin{pmatrix} 0 & t_{12} & t_{13} \\ t_{21} & 0 & t_{23} \\ t_{31} & t_{32} & 0 \end{pmatrix}.$$

If we consider job- and machine-dependent transportation times t_{jkl}, we get 12 groups $G_{(j,k)}$ ($j = 1, \ldots, 4$, $k = 1, 2, 3$), each consisting of a single operation. $\square$

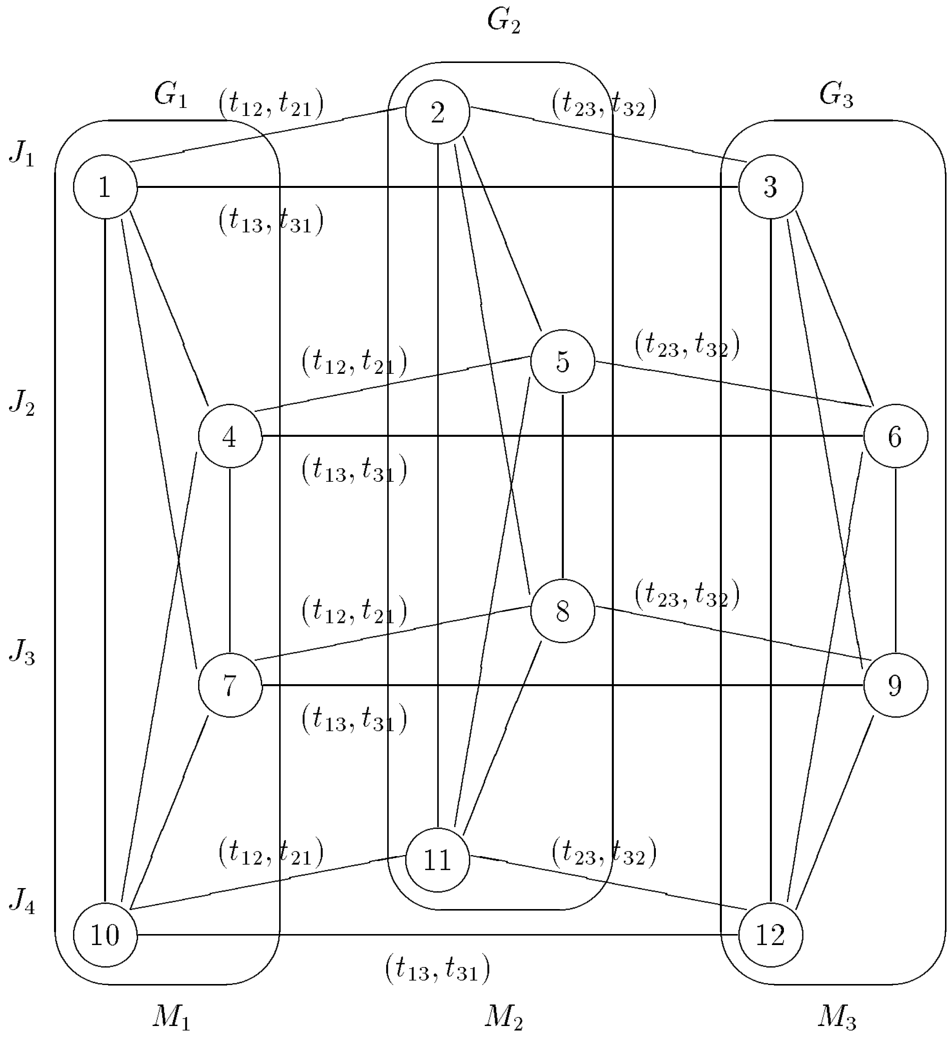


Figure 3.4: Disjunctive graph for an open-shop with $n = 4$, $m = 3$ and t_{kl}

3.2 Complexity results for transportation delays

In this subsection we give an overview of complexity results for shop problems with transportation times in which no transportation conflicts arise. These problems are equivalent to shop problems with minimal time-lags between operations belonging to the same job. The first problem of this type considered in the literature was the permutation flow-shop problem $F2\pi \mid t_j \mid C_{\max}$. Mitten [1958] showed that this problem can be solved in polynomial time by applying Johnson's algorithm (Johnson [1954]) for $F2 \mid\mid C_{\max}$ to an instance with processing times $p'_{1j} = p_{1j} + t_j$ and $p'_{2j} = p_{2j} + t_j$. If we do not restrict the search for an optimal schedule to permutation plans, the general problem $F2 \mid t_j \mid C_{\max}$ is much harder: Yu [1996] showed that this problem is already strongly $\mathcal{NP}$-hard in the case of unit processing times. On the other hand, in some special cases it can be shown that an optimal permutation plan always exists and hence the problem is polynomially solvable due to the result of Mitten [1958]. A simple example is problem $F2 \mid t_j = T \mid C_{\max}$. Since all transportation times are equal to a constant T, schedules for problems $F2 \mid\mid C_{\max}$ and $F2 \mid t_j = T \mid C_{\max}$ can be converted into each other by shifting the schedule on machine M_2 by T time units, and the objective values differ only by the constant T. Hence, an optimal schedule for the classical two-machine flow-shop is also optimal for the problem with constant transportation times.

Another sufficient condition for the existence of an optimal permutation plan has been given by Dell' Amico [1996]. If the processing and transportation times fulfill one of the conditions

$$\max_{j=1}^{n} \{t_j\} \leq \min_{j=1}^{n} \{p_{1j} + t_j\}$$

or

$$\max_{j=1}^{n} \{t_j\} \leq \min_{j=1}^{n} \{p_{2j} + t_j\},$$

then an optimal permutation plan always exists. Yu [1996] strengthened this result by showing that the condition

$$\max_{j=1}^{n} \{t_j\} \leq \min_{j=1}^{n} \{t_j + \max(p_{1j}, p_{2j})\}$$

is already sufficient.

Yu [1996] also considered problems in which the tranportation times may take only two values $T_1 < T_2$. He showed that in a flow-shop environment w.l.o.g. $T_1 = 0$ can be assumed (otherwise we may replace T_1, T_2 by $T'_1 = 0, T'_2 = T_2 - T_1$ and feasible schedules for the two corresponding instances can be converted into each other by shifting the schedule on machine M_2 by T_1 time units). Furthermore, he proved that problem $F2 \mid p_{1j} = p_{2j}, t_j \in \{0, T\} \mid C_{\max}$ is strongly $\mathcal{NP}$-hard and proposed an algorithm which constructs an optimal non-permutation solution for problem $F2 \mid p_{ij} = 1, t_j \in \{0, T\} \mid C_{\max}$ in polynomial time.

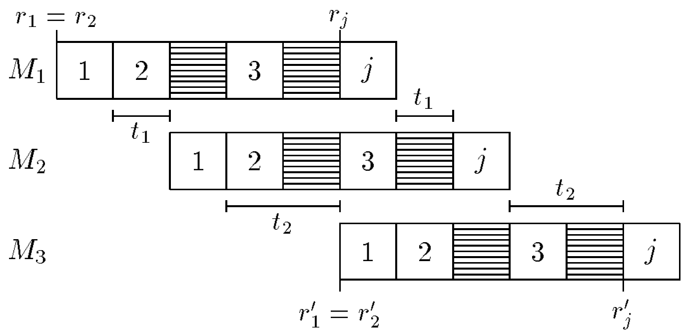


Figure 3.5: Schedule for problem $F \mid p_{ij} = 1, r_j, t_k \mid f$

In the following we will show that flow-shop problems with unit processing times and job-independent, but machine-dependent transportation times t_k are equivalent to single-machine problems. It is easy to see that for problem $F \mid p_{ij} = 1, r_j, t_k \mid f$ with a regular objective function f the search for an optimal schedule may be limited to permutation and no-wait plans. If in such a schedule job j starts at time S_{1j} on M_1, then it starts on M_m at time $S_{mj} = S_{1j} + m - 1 + \sum_{k=1}^{m-1} t_k$ (cf. Figure 3.5). If the release dates r_j are satisfied on M_1, then obviously the release dates $r'_j := r_j + m - 1 + \sum_{k=1}^{m-1} t_k$ must be satisfied on M_m. On the other hand, if in a single-machine schedule for M_m the release dates r'_j are respected, we can construct a feasible no-wait schedule for the whole flow-shop with the same completion times. Thus, for a flow-shop schedule and its objective value only the last machine is important and we get the single-machine problem $1 \mid p_j = 1, r_j \mid f$. From this equivalence we may conclude that problems $F \mid p_{ij} = 1, r_j, t_k \mid \sum w_j U_j, \sum w_j T_j$ are polynomially solvable, since the corresponding single-machine problems are polynomially solvable (assignment problems, cf. Brucker [1998]).

Problems with transportation times and more than two machines or with release dates or other objective functions are $\mathcal{NP}$-hard since the classical problems $F3 \mid\mid C_{\max}$, $F2 \mid r_j \mid C_{\max}$, $F2 \mid\mid L_{\max}$ and $F2 \mid\mid \sum C_j$ without transportation times are already strongly $\mathcal{NP}$-hard. The main results for flow-shop problems are summarized in Table 3.1. A $*$ indicates that the problem is strongly $\mathcal{NP}$-hard.

Open-shop problems with transportation times and no transportation conflicts were first considered by Rayward-Smith & Rebaine [1992]. They showed that problems $O2 \mid t_{jkl} = T \mid C_{\max}$ and $O \mid p_{ij} = 1, t_{kl} = t_{lk} \mid C_{\max}$ are $\mathcal{NP}$-hard. Yu [1996] strengthened the first result for the case in which the processing times on both machines are equal, and additionally showed that problems $O2 \mid p_{ij} = 1, t_j \mid C_{\max}$ and $O2 \mid p_{1j} = p_{2j}, t_j \in \{T_1, T_2\} \mid C_{\max}$ are strongly $\mathcal{NP}$-hard. Rayward-Smith & Rebaine [1992] proved that the length of an optimal schedule for $O \mid p_{ij} = 1, t_{jkl} = T \mid C_{\max}$ can be polynomially calculated by a simple formula and that a corresponding schedule can be constructed in $O(m^2 T)$ time. The construction of the schedule is not polynomial since the problem

Polynomially solvable:

$F2 \mid t_j = T \mid C_{\max}$	Johnson [1954]
$F2\pi \mid t_j \mid C_{\max}$	Mitten [1958]
$F2 \mid p_{ij} = 1, t_j \in \{0, T\} \mid C_{\max}$	Yu [1996]
$F2 \mid \max\{t_j\} \leq \min\{p_{1j} + t_j\} \mid C_{\max}$	Dell' Amico [1996]
$F2 \mid \max\{t_j\} \leq \min\{p_{2j} + t_j\} \mid C_{\max}$	Dell' Amico [1996]
$F2 \mid \max\{t_j\} \leq \min\{t_j + \max(p_{1j}, p_{2j})\} \mid C_{\max}$	Yu [1996]
$F \mid p_{ij} = 1, r_j, t_k \mid \sum w_j U_j$	Single-machine problem
$F \mid p_{ij} = 1, r_j, t_k \mid \sum w_j T_j$	Single-machine problem

$\mathcal{NP}$-hard:

$*$	$F2 \mid p_{ij} = 1, t_j \mid C_{\max}$	Yu [1996]
$*$	$F2 \mid p_{1j} = p_{2j}, t_j \in \{0, T\} \mid C_{\max}$	Yu [1996]
$*$	$F2 \mid r_j \mid C_{\max}$	Lenstra et al. [1977]
$*$	$F3 \mid\mid C_{\max}$	Garey et al. [1976]
$*$	$F2 \mid\mid L_{\max}$	Lenstra et al. [1977]
$*$	$F2 \mid\mid \sum C_j$	Garey et al. [1976]

Table 3.1: Complexity results for flow-shop problems with transportation delays

can be encoded in a very sparse way. For the two-machine open-shop problem with symmetrical transportation times t_j, where the transportation times are dominated by the processing times (i.e. $\max_{j=1}^{n}\{t_j\} \leq \min_{j=1}^{n}\min_{i=1}^{m}\{p_{ij}\}$), a polynomial algorithm has been found by Rebaine & Strusevich [1998].

In the following we will show that the two-machine open-shop problem with constant processing times $p_{ij} = p$ and job-independent transportation times t_{kl} can be solved in polynomial time.

Theorem 3.1: Problem $O2 \mid p_{ij} = p, t_{kl} \mid C_{\max}$ is polynomially solvable.

Proof: For the sake of simplicity we normalize the data in such a way that we have unit processing times on M_1 and M_2, but rational transportation times t_{12} and t_{21}. W.l.o.g. we assume $t_{12} \geq t_{21}$. A solution for this open-shop problem may be described by a partition (A_{12}, A_{21}) of the jobs, where A_{12} denotes the set of jobs which are first processed on M_1 and then on M_2, and where the jobs in A_{21} have the opposite machine order $M_2 \to M_1$. It can easily be proved by an exchange argument that for each partition (A_{12}, A_{21}) it is sufficient to consider the staged schedule S of the form as in Figure 3.6.

In the following we consider the set of all possible partitions and calculate the objective

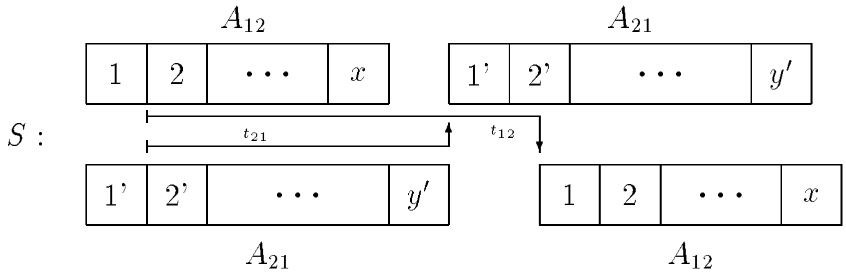


Figure 3.6: A staged schedule for problem $O2 \mid p_{ij} = p, t_{kl} \mid C_{\max}$

value for the schedule S belonging to an arbitrary partition (A_{12}, A_{21}), where $x := |A_{12}|$ and $|A_{21}| = n - x$. Then we show how an optimal partition can be determined.

For the schedule S^0 belonging to $A_{12} = \emptyset$ (i.e. $x = 0$) all jobs are transported from M_2 to M_1 and we have $C_{\max}(S^0) = n + t_{21} + 1$. Analogously, for $A_{21} = \emptyset$ (i.e. all jobs are transported from M_1 to M_2) we get $C_{\max}(S) = n + t_{12} + 1 \geq n + t_{21} + 1 = C_{\max}(S^0)$, which means that this schedule is dominated by S^0.
For $A_{12} \neq \emptyset, A_{21} \neq \emptyset$ (i.e. $x > 0$) the makespan of S is either determined by the schedule length on a machine, by the jobs in A_{12} or by the jobs in A_{21} (see Figure 3.6). Thus, the makespan of S is given by

$$C_{\max}(S) = \max\{n, x + t_{12} + 1, n - x + t_{21} + 1\}.$$

Let $f(x) := \max\{x + t_{12} + 1, -x + n + t_{21} + 1\}$ for $x > 0$ be the maximum of the last two terms and let $f(0) := n + t_{21} + 1 = C_{\max}(S^0)$. We consider the two cases (a) $t_{12} + 1 < n + t_{21} + 1$ and (b) $t_{12} + 1 \geq n + t_{21} + 1$ (cf. Figure 3.7).

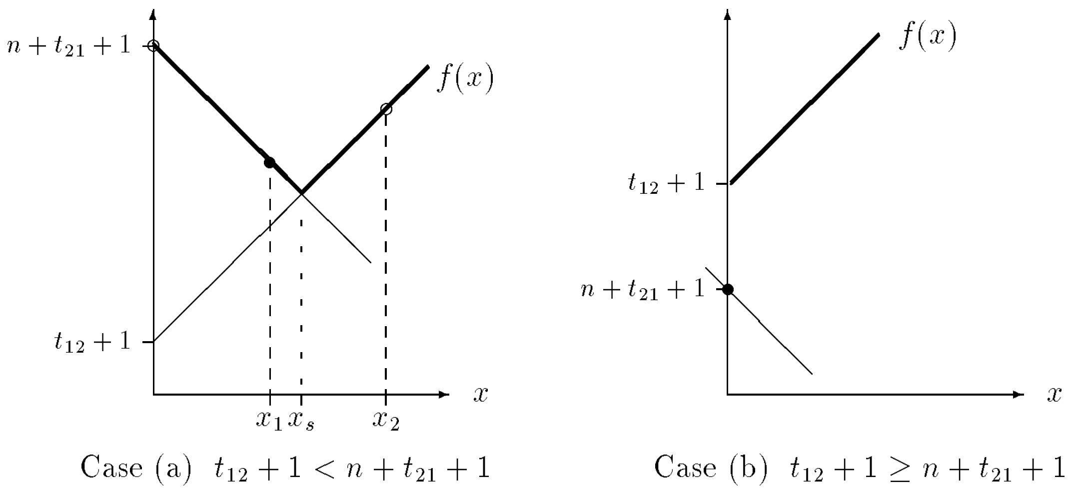


Figure 3.7: The function f

In the case (a) the minimum of f is achieved for $x_s = \frac{n+t_{21}-t_{12}}{2}$. Due to the integrality of x we have the choice between $x_1 := \lfloor x_s \rfloor$ and $x_2 := \lceil x_s \rceil$. Since the lines $x + t_{12} + 1$ and

$-x+n+t_{21}+1$ have the slopes 1 and -1, the minimum of $f(x_1)$ and $f(x_2)$ is obtained for the value $x^* \in \{x_1, x_2\}$ which is closer to their intersecting point x_s (cf. Figure 3.7 (a)), i.e.

$$x^* = \begin{cases} x_1, & \text{if } x_s - x_1 \leq \frac{1}{2} \\ x_2, & \text{otherwise.} \end{cases}$$

For the corresponding schedule S^* we have $C_{\max}(S^*) = \max\{n, f(x^*)\} \leq n + t_{21} + 1 = C_{\max}(S^0)$, i.e. S^* is optimal.

On the other hand, in the case (b) we have $f(x) \geq n + t_{21} + 1 = C_{\max}(S^0)$ for all $x \geq 0$. Thus, in this case schedule S^0 is optimal. □

Only a few results are available for job-shop problems with transportation times. The classical job-shop problems $J2 \mid\mid C_{\max}$ and $J3 \mid p_{ij} = 1 \mid C_{\max}$ are both strongly $\mathcal{NP}$-hard (Lenstra & Rinnooy Kan [1979]), which implies that only very restrictive special cases with transportation times may be polynomially solvable. Dell' Amico [1996] showed that the $\mathcal{NP}$-hard problem $J2 \mid n_j \leq 2, t_j \mid C_{\max}$ can be solved by constructing solutions of two $F2 \mid t_j \mid C_{\max}$ problems. Thus, the special case $J2 \mid n_j \leq 2, t_{jkl} = T \mid C_{\max}$ is polynomially solvable, since $F2 \mid t_j = T \mid C_{\max}$ is polynomially solvable.

The main results for open- and job-shop problems with transportation delays are summarized in Table 3.2.

Polynomially solvable:

$O2 \mid\mid C_{\max}$	Gonzalez & Sahni [1976]
$O \mid p_{ij} = 1, t_{jkl} = T \mid C_{\max}$	Rayward-Smith & Rebaine [1992]
$O2 \mid p_{ij} = p, t_{kl} \mid C_{\max}$	Theorem 3.1
$O2 \mid \max\{t_j\} \leq \min\{p_{ij}\} \mid C_{\max}$	Rebaine & Strusevich [1998]
$J2 \mid n_j \leq 2, t_{jkl} = T \mid C_{\max}$	Dell' Amico [1996]

$\mathcal{NP}$-hard:

*	$O2 \mid p_{ij} = 1, t_j \mid C_{\max}$	Yu [1996]
*	$O2 \mid p_{1j} = p_{2j}, t_j \in \{T_1, T_2\} \mid C_{\max}$	Yu [1996]
	$O2 \mid p_{1j} = p_{2j}, t_{jkl} = T \mid C_{\max}$	Yu [1996]
	$O3 \mid\mid C_{\max}$	Gonzalez & Sahni [1976]
*	$O \mid p_{ij} = 1, t_{kl} = t_{lk} \mid C_{\max}$	Rayward-Smith & Rebaine [1992]
*	$O2 \mid r_j \mid C_{\max}$	Lawler et al. [1981,1982]
*	$O2 \mid\mid \sum C_j$	Achugbue & Chin [1982]

Table 3.2: Complexity results for open- and job-shop problems with transportation delays

4 Problems with transportation conflicts

In this section we study shop problems with transportation times where conflicts between different transportations may arise. This situation occurs if we only have a limited number $r < n$ of robots $R_1, \dots, R_r$ or if we have an environment with multi-purpose robots, i.e. some jobs may not be transported by any robot. In these environments it may happen that more jobs require transportation than free robots are available. Then it has to be decided which job is transported by which robot and which job has to wait before its transportation.

In order to extend the disjunctive graph model to these situations we have to integrate the possible transport conflicts for the robots into the model. In the model of Section 3.1 the transportation times are modeled by weights on the corresponding *arcs* in the graph, which implies that these times are only taken into account as delays between different operations. Implicitly, it is assumed that all these transportations can be carried out simultaneously. In the situation with conflicts we have to ensure that the transportation stage becomes a bottleneck stage which will be done by introducing transport operations as additional *vertices* in the disjunctive graph and requiring that these operations have to be processed by the robots.
Since the transport operations depend on the machine routings of the jobs, it is not possible to model a general-shop problem in this way, because in a general-shop the machine orders for the jobs are not given in advance. However, to model a general-shop situation we may proceed in two steps. After fixing the order of each job's operations, the machine routings of the jobs are given and it is known between which machines a job has to be transported. Then the general-shop problem reduces to a job-shop problem with additional precedence relations between some operations belonging to different jobs. For this reason in the following we restrict our considerations to a job-shop environment.

In Subsection 4.1 we extend the disjunctive graph model to job-shop problems with dedicated robots and show that this problem can again be formulated as a special case of the general-shop problem with sequence-dependent setup times. In Subsection 4.2 we consider the subproblem of scheduling a single robot for a fixed machine selection, which will be studied in more detail in Sections 6-8.

4.1 Job-shop problems with multi-purpose robots

In this subsection we consider the job-shop problem $J, MPR \mid prec, t_{jkl}, t'_{kl} \mid C_{\max}$ with arbitrary precedence relations between the operations, transportation and empty moving times and multi-purpose transport robots $R_1, \dots, R_r$. Each job J_j consists of n_j operations $O_{ij}\ (i = 1, \dots, n_j)$ with chain precedence constraints $O_{1j} \rightarrow O_{2j} \rightarrow \dots \rightarrow O_{n_j,j}$. Furthermore, for each j a set $\mu_j^R \subseteq \{R_1, \dots, R_r\}$ is given containing all robots by which job J_j may be transported.

In order to extend the disjunctive graph model to these problems we proceed as follows. For each job J_j ($j = 1, \dots, n$) we introduce $n_j - 1$ so-called **transport operations** T_{ij} ($i = 1, \dots, n_j - 1$) with precedences $O_{ij} \to T_{ij} \to O_{i+1,j}$. The processing time of T_{ij} is equal to the transportation time of job J_j from machine μ_{ij} to $\mu_{i+1,j}$, i.e. $p(T_{ij}) = t_{jkl}$, when $\mu_{ij} = M_k$, $\mu_{i+1,j} = M_l$. The robots may be considered as additional "machines" which have to process the transport operations (where transport operation T_{ij} can only be processed by one of the robots in μ_j^R). To construct a feasible schedule for the job-shop problem with multi-purpose robots we have to

- assign each transport operation T_{ij} to a robot from the set μ_j^R,
- determine feasible starting times for all operations O_{ij} on the machines $M_1, \dots, M_m$, and
- determine feasible starting times for all transport operations T_{ij} on the assigned robots.

As in Section 3.1 we want to use the disjunctive graph model to construct optimal schedules. This means we determine orders for all operations processed on the same machine and orders for all transport operations assigned to the same robot. Since the possible conflicts between transport operations on the same robot depend on the robot assignment, such an assignment has to be given before a disjunctive graph can be defined. Thus, this problem may be solved with a two-stage approach and in the following we consider the situation of dedicated robots (problem $J, DR \mid prec, t_{jkl}, t'_{kl} \mid C_{\max}$). We assume that we are given a complete robot assignment ρ where each transport operation T_{ij} is assigned to a robot $\rho(T_{ij}) \in \mu_j^R$. Let G_{kl} ($k, l = 1, \dots, m$) be the set of all transport operations with the same transport routing, i.e. we have

$$T_{ij} \in G_{kl} \text{ if and only if } \mu_{ij} = M_k, \, \mu_{i+1,j} = M_l.$$

Then the **disjunctive graph** $G = (V, C \cup D_M \cup D_R)$ associated with ρ is defined as follows.

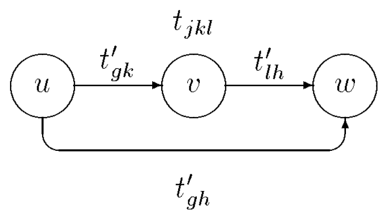
- The set of vertices $V := \{O_{ij} \mid j = 1, \dots, n;\ i = 1, \dots, n_j\} \cup \{T_{ij} \mid j = 1, \dots, n; i = 1, \dots, n_j - 1\} \cup \{0, *\}$ represents all operations and transport operations of the jobs and two dummy nodes 0 and $*$. The source 0 and the sink $*$ are used to define the start and the end of the operations. The operations O_{ij} are weighted with their processing times p_{ij}, the transport operations T_{ij} with their transportation times $p(T_{ij})$, and the weights of the dummy nodes are 0.
- The set of directed arcs C represents all precedence constraints $\to$ and is called the set of **conjunctions**. Besides the given precedence relations $O_{ij} \to O_{uv}$ between operations belonging to different jobs we have arcs $O_{ij} \to T_{ij} \to O_{i+1,j}$ for all jobs $j = 1, \dots, n$ and $i = 1, \dots, n_j - 1$, weighted with 0. Additionally, there are arcs between the source and all operations without any predecessor, and between all operations without any successor and the sink.

- The set of undirected edges D_M consists of all pairs of operations which have to be processed on the same machine and are not linked by a conjunction. These pairs are called **machine disjunctions**. An edge $O_{ij} - O_{uv} \in D_M$ with $\mu_{ij} = \mu_{uv} = M_k$ represents the two possible orders in which the operations O_{ij} and O_{uv} may be processsed on M_k and is weighted with 0.

- The set of undirected edges D_R consists of all pairs of transport operations which are not linked by a conjunction and are assigned to the same robot. These pairs are called **robot disjunctions**. An edge $T_{ij} - T_{uv} \in D_R$ represents the two possible orders in which the transport operations T_{ij} and T_{uv} may be processsed on the robot and is weighted with the pair (t'_{kl}, t'_{gh}) of empty moving times, when $T_{ij} \in G_{hk}, T_{uv} \in G_{lg}$.

To solve the scheduling problem we have to turn all undirected arcs in $D_M \cup D_R$ into directed ones. Concerning an edge in D_R weighted with a pair of empty moving times, at this point the directed arc gets the corresponding unique weight as in Section 3. In order to distinguish between machines and robots we call a set S_M of fixed machine disjunctions a **machine selection** and a set of fixed robot disjunctions S_R a **robot selection**. A set $\hat{S} = C \cup S_M \cup S_R$ is called a **selection**. If all disjunctive arcs have been fixed and the corresponding graph $G(\hat{S}) = (V, \hat{S})$ is acyclic, $\hat{S}$ is called a **complete selection**. Given a complete selection $\hat{S}$ a corresponding schedule can be constructed as in Section 3.1 by longest path calculations and starting each operation as early as possible. The makespan $C_{\max}(\hat{S})$ of the schedule corresponding to $\hat{S}$ is given by the length of a longest path from 0 to $*$ in $G(\hat{S})$.

Since empty moving times only occur between transportation times which are processed consecutively on the same robot, all transitive arcs corresponding to fixed robot disjunctions must not be taken into account in the longest path calculations. But, as on page 18, we show that paths consisting of transitive arcs are not longer than paths consisting of non-transitive arcs, i.e. the transitive arcs do not change the longest path lengths. Let u, v, w be three transport operations which are assigned to the same robot where the corresponding robot disjunctions are directed to $u \to v \to w$. Assuming that operation v belongs to job j, we have the following situation:



Due to the assumptions (2.2) for the empty moving times we have

$$t'_{gh} \leq t'_{gk} + t'_{kl} + t'_{lh} \leq t'_{gk} + \min_{i=1}^{n} \{t_{ikl}\} + t'_{lh} \leq t'_{gk} + t_{jkl} + t'_{lh}, \tag{4.1}$$

i.e. the transitive arc $u \to w$ is not longer than the path $u \to v \to w$ consisting of non-transitive arcs. By transitivity the same property also holds for all paths consisting of more than three transport operations, which shows that transitive arcs do not have to be removed from the disjunctive graph.

Example 4.1: We consider an instance of problem $J4, MPR2 \mid t_{jkl}, t'_{kl} \mid C_{\max}$ with $m = 4$ machines, $r = 2$ multi-purpose robots, $n = 3$ jobs, and 8 operations. Let $\mu_1^R = \mu_2^R = \{R_1, R_2\}$ and $\mu_3^R = \{R_2\}$. The corresponding disjunctive graph for the robot assignment $\rho(T_{11}) = \rho(T_{12}) = R_1$, $\rho(T_{21}) = \rho(T_{13}) = \rho(T_{23}) = R_2$ can be found in Figure 4.1. All transport operations which do not belong to the same job and which are assigned to the same robot are linked by a robot disjunction weighted with the corresponding empty moving times. For example, the disjunction on R_1 between the transport operations $T_{11} \in G_{12}$ and $T_{12} \in G_{23}$ is weighted with the pair (t'_{22}, t'_{31}). $\square$

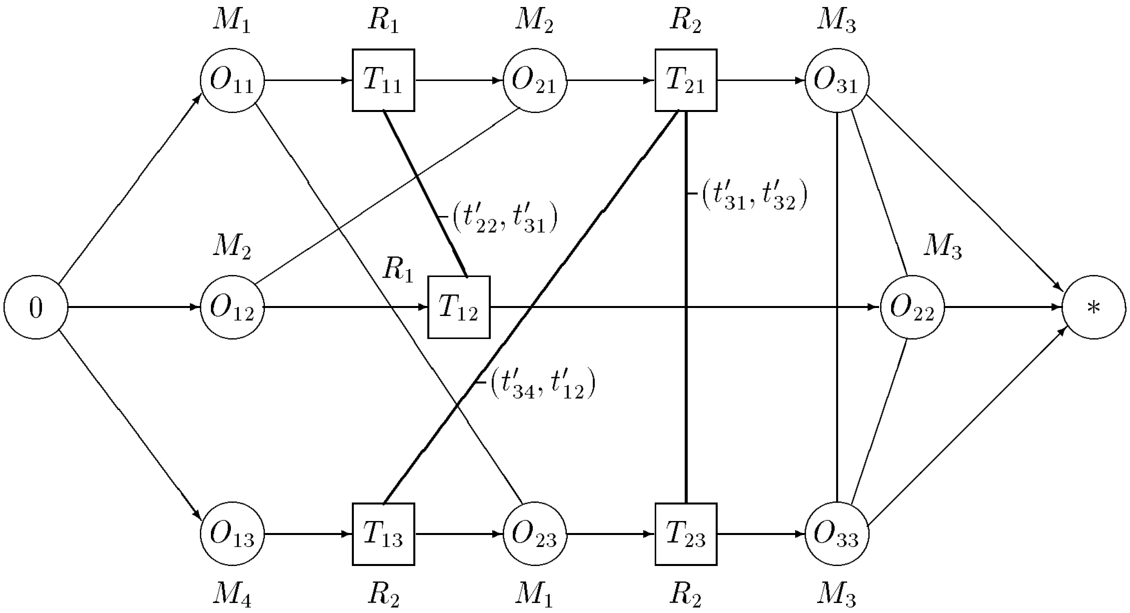


Figure 4.1: Disjunctive graph for a job-shop with $n = 3$, $m = 4$ and $r = 2$ robots

Again, as in Section 3.1, we have a relationship to the general-shop problem with sequence-dependent setup times, which will be described next. In the considered situation changeover times occur between transport operations T_{ij} which are assigned to the same robot (corresponding to the empty moving times). No setup times arise between the ordinary operations O_{ij}. Thus, an instance of the job-shop problem $J, DR \mid t_{jkl}, t'_{kl} \mid C_{\max}$ with m machines and r robots can be transformed into an instance of the job-shop problem $J \mid s_{fg} \mid C_{\max}$ with $m + r$ machines and sequence-dependent setup times in the following way:

- introduce r additional machines corresponding to the robots,
- partition the transport operations T_{ij} $(j = 1, \ldots, n;\ i = 1, \ldots, n_j - 1)$ into m^2 groups $G_{(k,l)}$ $(k, l = 1, \ldots, m)$ with

$$T_{ij} \in G_{(k,l)} \text{ if and only if } \mu_{ij} = M_k \text{ and } \mu_{i+1,j} = M_l,$$

- define the setup times between the groups by the corresponding empty moving times: $$s_{(k,l)(h,v)} = t'_{lh}.$$

Due to inequality (4.1) the defined setup times satisfy the weak triangle inequality

$$s_{(u,g)(k,l)} + \min_{T_{ij} \in G_{(k,l)}} p(T_{ij}) + s_{(k,l)(h,v)} \geq s_{(u,g)(h,v)}.$$

Finally, we mention some relationships of special cases of the job-shop problem with multi-purpose robots to other scheduling problems. If we have identical robots and no empty moving times (i.e. $t'_{kl} = 0$), problem $J, R \mid t_{jkl} \mid C_{\max}$ becomes a special case of the so-called hybrid job-shop problem, where the r robots correspond to one center consisting of r parallel machines. If in this environment only a single robot exists, problem $Jm, R1 \mid t_{jkl} \mid C_{\max}$ is equivalent to a classical job-shop problem with $m + 1$ machines, where the robot corresponds to an additional machine. Since this machine has to process many more operations than the other machines (each second operation of a job), it is also called a bottleneck machine.

4.2 Scheduling the robots for a fixed machine selection

In the previous subsection we considered the scheduling problem which arises in problem $J, MPR \mid prec, t_{jkl}, t'_{kl} \mid C_{\max}$ for a fixed robot assignment. This scheduling problem consists of two parts: on the one hand, the jobs have to be scheduled on the machines, and, on the other hand, the transport operations have to be scheduled on the assigned robots. In this subsection we go one step further and consider the situation of the robots when additionally the machine orders are given (i.e. a machine selection is fixed).

Such a hierarchical decomposition of the problem may be useful in different situations. Since for the machines we have a similar situation as in a classical job-shop, all techniques known for the job-shop problem may be applied to the machines. Thus, if we want to integrate a transportation stage into existing procedures, it remains to handle the robots appropriately. For example, the so-called shifting bottleneck procedure of Adams et al. [1988], which is an efficient heuristic algorithm for the job-shop problem, is based on a decomposition of the job-shop into different single-machine problems. Thus, to incorporate a transportation stage, the resulting generalized single-machine problems for the robots have to be studied.

Another decomposition approach for problem $J, R1 \mid t_{jkl}, t'_{kl} \mid C_{\max}$ will be described in Section 9 where in a local search algorithm machine and robot selections are changed in different ways.

In the following we assume that we are given a robot assignment and a machine selection S_M represented by the disjunctive graph $G(S)$ with $S = C \cup S_M$. We are interested in an optimal robot selection S_R respecting the precedence relations C and the orders on the machines defined by S_M. Let V' be the set of all transport operations and define the length of a path between two operations i, j in $G(S)$ as the sum of processing times of all operations on the path (p_i, p_j excluded).

With each transport operation $j \in V'$ we may associate a **head** and a **tail**. A head r_j is a lower bound for the earliest possible starting time of j, and a tail q_j is a lower bound for the time period between the completion time of j and the optimal makespan. For a given selection S heads r_j may be defined by the lengths l_{0j} of longest paths from the source 0 to node j in $G(S)$ since operation j cannot be started before all predecessors in $G(S)$ are scheduled. Symmetrically, tails q_j may be defined by the lengths l_{j*} of longest paths from j to $*$ in $G(S)$.

Since the given precedence constraints C and the fixed machine disjunctions S_M have to be respected in a feasible robot selection, some additional constraints for the transport operations can be derived from the graph $G(S)$. Obviously, if for transport operations $i, j \in V'$ a path from i to j in $G(S)$ exists, i has to be processed before j. Furthermore, the longest path lengths l_{ij} between i and j define minimal finish-start **time-lags** between the completion time C_i of i and the starting time S_j of j (e.g. between two consecutive transportations of the same job the job has to be processed on a machine for a certain time period). Let C' be the set of all precedence relations for the transport operations (including the transitive ones) induced in this way. For all $i \to j \in C'$ we have to fulfill $S_j - C_i \geq l_{ij}$.

All pairs of transport operations $i, j \in V'$ which are not linked by a conjunction from C', but are assigned to the same robot, cannot be processed simultaneously. This can again be modeled by a set D' of disjunctions. As previously described, these disjunctions $i - j \in D'$ are weighted with sequence-dependent **setup times** (s_{ij}, s_{ji}). The setup time s_{ij} arises if i is processed directly before j on the same robot (corresponding to an empty move between the two transportations).

Summarizing, to determine an optimal robot selection S_R for a fixed machine selection S_M we get the following scheduling problem for the transport operations with dedicated machines (corresponding to the assigned robots): schedule all transport operations $j \in V'$ on their assigned robots in such a way that the release dates r_j, the minimal time-lags l_{ij} and sequence-dependent setup times s_{ij} are respected and the extended makespan $\max_{j \in V'}\{C_j + q_j\}$ is minimized. Note that this objective value equals the makespan $C_{\max}(\hat{S})$ of the complete selection $\hat{S} = C \cup S_M \cup S_R$ since all restrictions of the job-shop problem are incorporated into the constraints for the problem of the robots.

In the following we decompose this problem with dedicated machines into single-machine problems for each robot. These problems are relaxations of the problem with dedicated machines since the interaction of the robots due to the time-lags is neglected if we consider the robots independently.

In the $\alpha|\beta|\gamma$-notation the single-machine problem for one robot can be denoted by $1 \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$, where $prec(l_{ij})$ indicates arbitrary non-negative finish-start time-lags $l_{ij} \geq 0$ (cf. Brucker & Knust [1998]). Since this problem contains several $\mathcal{NP}$-hard subproblems, it itself is strongly $\mathcal{NP}$-hard:

- if we have no minimal time-lags l_{ij} and all setup times s_{ij} are equal to 0, the problem is equivalent to $1 \mid r_j \mid \max\{C_j + q_j\}$ or $1 \mid r_j \mid L_{\max}$, which is strongly $\mathcal{NP}$-hard (Lenstra et al. [1977]),
- if all heads, tails and setup times are equal to 0, the resulting problem $1 \mid prec(l_{ij}) \mid C_{\max}$ is already strongly $\mathcal{NP}$-hard for chain-structured constant time-lags $l_{ij} = l$ (Balas et al. [1995]),
- if no heads, tails and time-lags are given, we get the traveling salesman problem $1 \mid s_{ij} \mid C_{\max}$, where the makespan has to be minimized. This problem is also $\mathcal{NP}$-hard in the strong sense (Garey & Johnson [1979]).

Relaxing the constraint that preemption is not allowed does not make the problem much easier: While problem $1 \mid pmtn, r_j \mid \max\{C_j + q_j\}$ is polynomially solvable (Jackson [1955]), problem $1 \mid pmtn, chains(l_{ij}) \mid C_{\max}$ (Balas et al. [1995]) is still $\mathcal{NP}$-hard. For problem $1 \mid pmtn, s_{ij} \mid C_{\max}$ it can be shown that preemption does not reduce the optimal objective value, hence the problem remains $\mathcal{NP}$-hard.

Algorithms have been presented in the literature for some of these hard subproblems. Carlier [1982] developed a very fast branch-and-bound algorithm for $1 \mid r_j \mid \max\{C_j + q_j\}$ which was extended by Balas et al. [1995] to problem $1 \mid prec(l_{ij}), r_j \mid \max\{C_j + q_j\}$ in which additionally minimal time-lags l_{ij} exist.

The decision version of problem $1 \mid r_j \mid \max\{C_j + q_j\}$ (i.e. given a threshold value T, does a feasible schedule exist with $\max\{C_j + q_j\} \leq T$?) may also be regarded as feasibility problem $1 \mid r_j, d_j \mid -$ where all operations have to be processed in their time windows $[r_j, d_j]$. Since q_j is a lower bound for the time we need after finishing operation j, in each schedule with an objective less than or equal to T operation j has to be finished not later than time $d_j := T - q_j$. Problem $1 \mid r_j, d_j, s_{ij} \mid -$ is equivalent to the asymmetric traveling salesman problem with time windows (ATSP-TW). Several algorithms have been developed for the ATSP-TW and the classical TSP without time windows (for an overview cf. Desrochers et al. [1988]).

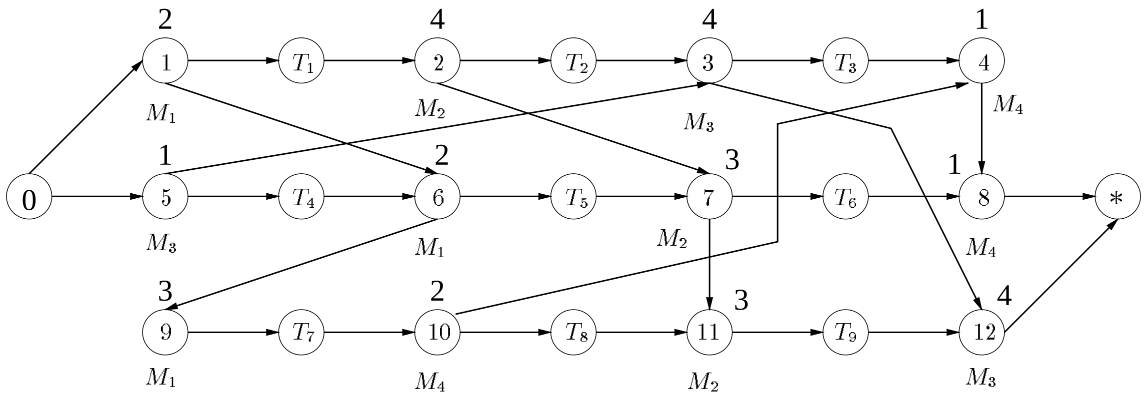


Figure 4.2: Machine selection S for a job-shop with one robot and $n = 3$ jobs

Example 4.2: We consider an instance of problem $J, R1 \mid t_{jkl}, t'_{kl} \mid C_{\max}$ with $n = 3$ jobs, 12 operations and $m = 4$ machines. The transportation times t_{jkl} are all assumed to be equal to 1, the empty moving times t'_{kl} are equal to 1 for $k \neq l$ and $t'_{kk} = 0$ otherwise. A complete machine selection S can be found in Figure 4.2. We have fixed the machine disjunctions on M_1 into the directions $1 \to 6 \to 9$, on M_2 into $2 \to 7 \to 11$, on M_3 into $5 \to 3 \to 12$, and on M_4 into $10 \to 4 \to 8$. The operations $u = 1, \ldots, 12$ are weighted with their processing times p_u, and all transport operations $T_1, \ldots, T_9$ have to be weighted with the transportation times $t_{jkl} = 1$.

In Figure 4.3 the disjunctive graph for the robot according to the machine selection S can be found. In this graph the arcs $T_i \to T_h$ between transport operations T_i, T_h are weighted with the longest path lengths l_{T_i,T_h} resulting from the fixed machine disjunctions. For example, from the order $2 \to 7$ on M_2 we get an arc $T_1 \to T_6$ with length $p_2 + p_7 = 7$. The arcs $0 \to T_i$ are weighted with the length of a longest path from 0 to T_i. For example, for T_7 we get $p_1 + p_6 + p_9 = p_5 + p(T_4) + p_6 + p_9 = 7$. Symmetrically, the arcs $T_i \to *$ are weighted with the length of a longest path from T_i to $*$. A disjunction, weighted with a pair of empty moving times, exists between all pairs of transport operations which are not linked by a directed arc. In Figure 4.3 not all disjunctions and empty moving times are drawn for clarity, but for each transport operation which transports from M_k to M_l the corresponding group G_{kl} is given.

In Figure 4.4 a complete robot selection S_R with a corresponding schedule can be found. The robot processes the transport operations in the order $T_1 \to T_2 \to T_4 \to T_5 \to T_7 \to T_8 \to T_6 \to T_9 \to T_3$ resulting in a critical path with length 26. Note that some redundant arcs (these are arcs (u, v) where a path between u and v exists which is not shorter than the weight of the arc) are omitted. □

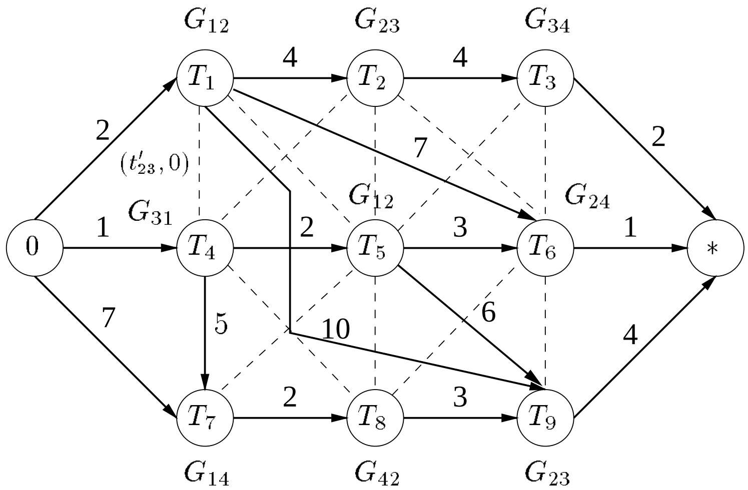


Figure 4.3: Disjunctive graph for the robot

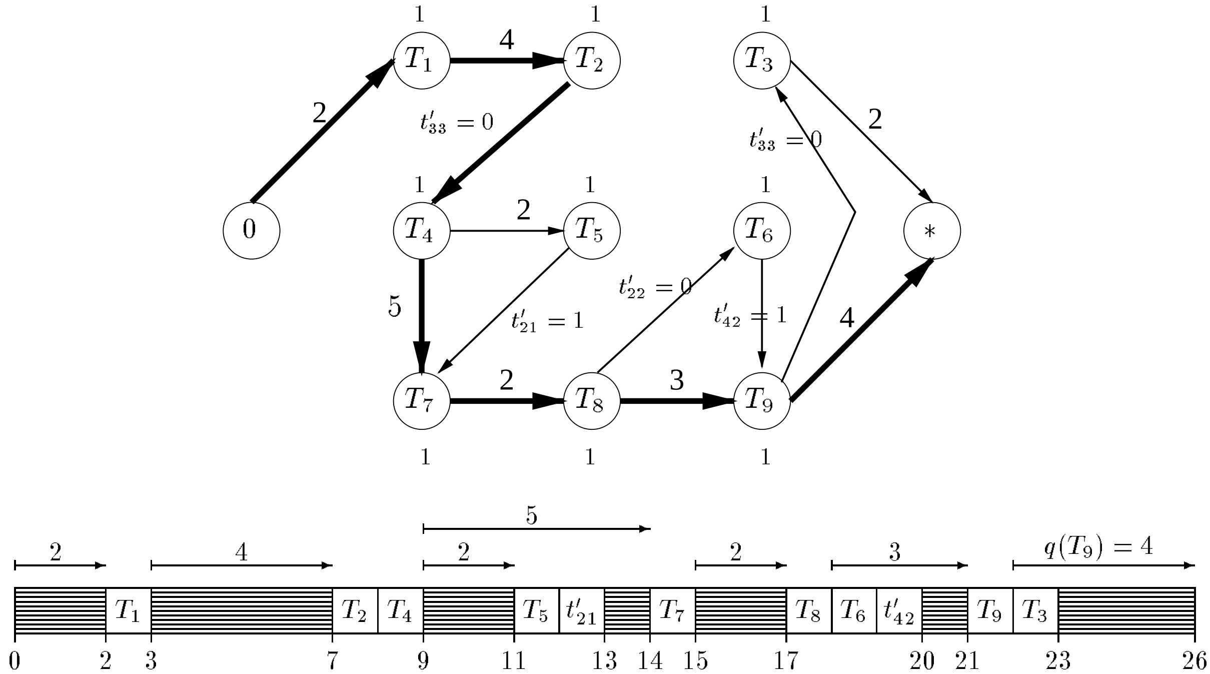
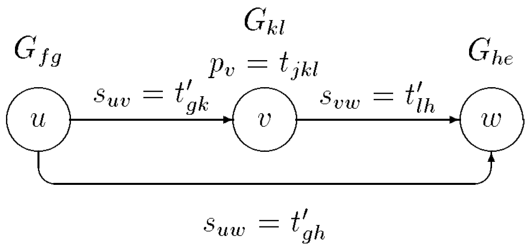


Figure 4.4: A robot selection and the corresponding schedule

Finally, we derive some special properties which are satisfied by the single-machine instances arising for a single robot. These properties will be used in connection with algorithms for the robot problem in Sections 6-8. Let u, v, w be three transport operations. Then we have a situation as follows where the setup times result from the corresponding empty moving times.



The setup times satisfy the weak triangle inequality $s_{uw} \leq s_{uv} + p_v + s_{vw}$ since, due to the assumptions (2.2) for the empty moving times, we have

$$s_{uw} = t'_{gh} \leq t'_{gk} + t'_{kl} + t'_{lh} \leq t'_{gk} + \min_{i=1}^{n} \{t_{ikl}\} + t'_{lh} \leq t'_{gk} + t_{jkl} + t'_{lh} = s_{uv} + p_v + s_{vw}.$$

Furthermore, we have the following triangle inequalities which link the setup times with the minimal time-lags:

$$s_{uw} \leq s_{uv} + p_v + l_{vw} \text{ for all } u - v, u - w \in D', v \to w \in C', \tag{4.2}$$

$$s_{uw} \leq l_{uv} + p_v + s_{vw} \text{ for all } v - w, u - w \in D', u \to v \in C'. \tag{4.3}$$

To prove the first inequality, we distinguish the following three cases:

- If v, w are transport operations belonging to the same job, this job has to be transported from machine M_l to machine M_h in between. Due to the triangle inequality in (2.2) and the fact that transportation times are not smaller than the corresponding empty moving times, we have $l_{vw} \geq t'_{lh}$, i.e. the inequality follows in the same way as above:

$$s_{uw} = t'_{gh} \leq t'_{gk} + t'_{kl} + t'_{lh} \leq s_{uv} + p_v + l_{vw}.$$

- If v, w belong to different jobs and the path in $G(S)$ between v and w does not contain a job arc, all arcs on the path must be directed machine disjunctions, i.e. we have $M_l = M_h$. Thus,

$$s_{uw} = t'_{gh} \leq t'_{gk} + t_{jkl} + t'_{lh} = t'_{gk} + t_{jkl} + t'_{ll} = s_{uv} + p_v \leq s_{uv} + p_v + l_{vw}$$

  due to $t'_{ll} = 0$ and $l_{vw} \geq 0$.

- The third case in which v, w belong to different jobs and the path in $G(S)$ between v and w contains a job arc can be proved analogously by combining the first two cases.

The second inequality can be proved in a similar way.

5 Complexity results for flow-shop problems with a single robot

In this section we consider flow-shop problems with transportation times, no empty moving times and a single robot (see also Hurink & Knust [1998]). In a flow-shop each job j consists of m operations $O_{1j} \rightarrow O_{2j} \rightarrow \ldots \rightarrow O_{mj}$, where operation O_{kj} has to be processed on machine $\mu_{kj} = M_k$. Therefore, a job is only transported from M_k to M_{k+1} $(k = 1, \ldots, m-1)$ and we have transportation times of the form t_{jk}.

New complexity results (polynomial algorithms as well as $\mathcal{NP}$-hardness proofs) are derived for special cases. At first we study problems with two machines, then in Subsection 5.2 we deal with problems with an arbitrary number m of machines. A summary of all known complexity results and some additional remarks can be found in Subsection 5.3.

5.1 Flow-shop problems with two machines

In this section we consider problems in which we have two machines M_1, M_2, one robot R and n jobs j with processing times p_{1j} on M_1 and p_{2j} on M_2. Furthermore, for each job j a transportation time t_j from M_1 to M_2 is given. This problem, $F2, R1 \mid t_j \mid C_{\max}$, is equivalent to the classical 3-machine flow-shop $F3||C_{\max}$ where the second machine corresponds to the robot R.

At the beginning of this section we have stated that we do not consider empty moving times. It is easy to see that for $m = 2$ machines the more general situation with empty moving times $t'_{21} \neq 0$ (arising for the robot when it moves back from machine M_2 to M_1 after having transported a job) is equivalent to our model without empty moving times. Given a problem with $t'_{21} \neq 0$, we construct an instance of the corresponding problem with $t'_{21} = 0$ by adding t'_{21} to all transportation times t_j. Each feasible schedule for the resulting problem can be transformed into a feasible schedule for the original problem and vice versa: we shift the whole schedule on machine M_2 by t'_{21} time units to the left (right), which changes the objective value by only a constant.

For problem $F2, R1 \mid t_j \mid C_{\max}$ we may restrict the search for an optimal solution to permutation plans, since for problem $F3||C_{\max}$ an optimal permutation plan always exists. The objective value of a permutation π is given by the value of a critical path. Such a path contains at first some operations on the first machine, then on the robot and finally on the second machine:

$$F(\pi) = \max_{k=1}^{n} \max_{l=k}^{n} \{\sum_{\lambda=1}^{k} p_{1\pi_\lambda} + \sum_{\lambda=k}^{l} t_{\pi_\lambda} + \sum_{\lambda=l}^{n} p_{2\pi_\lambda}\}. \tag{5.1}$$

In the following subsections we consider two special cases of this problem and show that they are both $\mathcal{NP}$-hard in the strong sense. In the first case we have arbitrary processing times of the jobs, but constant transportation times; in the second we have arbitrary transportation times, but constant processing times of all operations. In Subsections 5.1.3 and 5.1.4 we consider special cases of problem $F2, R1 \mid p_{ij} = p, t_j \mid C_{\max}$ which are polynomially solvable. In the first case we have only two transportation time values T_1 and T_2, and in the second case we assume unit processing times $p_{ij} = 1$.

5.1.1 Problem $F2, R1 \mid t_j = T \mid C_{\max}$

In problem $F2, R1 \mid t_j = T \mid C_{\max}$ we have arbitrary processing times on M_1 and M_2, but all transportation times t_j are equal to a constant T. The equivalent version of this problem with empty moving times t'_{21} has been shown to be $\mathcal{NP}$-hard in the ordinary sense by Kise [1991]. In the following we strengthen Kise's result by showing that the problem is strongly $\mathcal{NP}$-hard. Within the proof of the result we will use the following reformulation of the objective value (5.1) for a permutation π in the considered special case of equal transportation times:

$$F(\pi) = \max_{k=1}^{n} \max_{l=k}^{n} \{\sum_{\lambda=1}^{k} p_{1\pi_\lambda} + (l-k+1)T + \sum_{\lambda=l}^{n} p_{2\pi_\lambda}\}. \tag{5.2}$$

Theorem 5.1: Problem $F2, R1 \mid t_j = T \mid C_{\max}$ is $\mathcal{NP}$-hard in the strong sense.

Proof: We will show that the strongly $\mathcal{NP}$-complete problem 3-PARTITION is reducible to the decision problem of $F2, R1 \mid t_j = T \mid C_{\max}$.

3-PARTITION (3-PART): Given $3m$ positive numbers $a_1, \ldots, a_{3m}$ with $\sum_{i=1}^{3m} a_i = mb$ and $\frac{b}{4} < a_i < \frac{b}{2}$ for $i = 1, \ldots, 3m$, does there exist a partition $I_1, \ldots, I_m$ of the index set $\{1, \ldots, 3m\}$ such that $|I_j| = 3$ and $\sum_{i \in I_j} a_i = b$ for $j = 1, \ldots, m$?

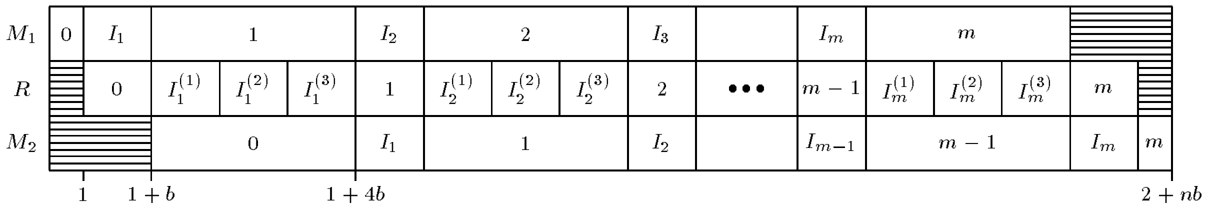
W.l.o.g. we can assume that in the instance of 3-PART $a_i > 1$ holds for all i (if this is not the case, we scale the data in an appropriate way). Given an arbitrary instance of 3-PART, we construct the following instance of the flow-shop problem $F2, R1 \mid t_j = T \mid C_{\max}$ with $n := 4m + 1$ jobs:

job 0 :	$p_{10} = 1,$	$p_{20} = 3b$	
job m :	$p_{1m} = 3b,$	$p_{2m} = 1$	
jobs $1, \ldots, m-1$:	$p_{1j} = 3b,$	$p_{2j} = 3b$	for $j = 1, \ldots, m-1$
jobs $m+1, \ldots, 4m$:	$p_{1,i+m} = a_i,$	$p_{2,i+m} = a_i$	for $i = 1, \ldots, 3m.$

The transportation time T is set to $T = b$ and we ask for a permutation π with $F(\pi) \leq y := 2 + (4m+1)b = 2 + nb$.

We show that a permutation π for the constructed instance with $F(\pi) \leq y$ exists if and only if 3-PART has a solution.

Assume that $I_1, \dots, I_m$ is a solution of 3-PART. Then we define a permutation π as follows.



Obviously, this permutation π fulfills $F(\pi) \leq y$.

Conversely, assume that the flow-shop problem has a solution π with $F(\pi) \leq y$. By setting $k = 1, l = n$ in (5.2) we get for all permutations π:

$$F(\pi) \geq p_{1\pi_1} + nT + p_{2\pi_n} \geq 1 + nb + 1 = y.$$

Thus, for a permutation π with $F(\pi) = y$ we may conclude that

(1) job 0 is processed at the first position, since $p_{1i} > 1$ for $i \neq 0$,

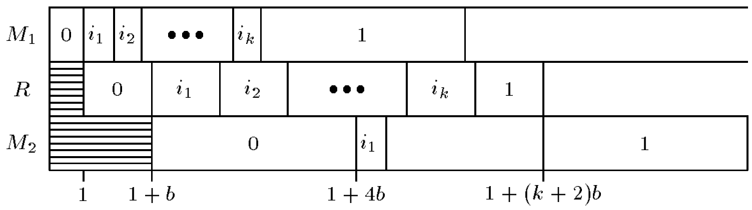
(2) job m is processed at the last position, since $p_{2i} > 1$ for $i \neq m$,

(3) machine M_1 processes jobs in the interval $[0, 1 + (n-1)b]$ without idle time,

(4) robot R transports jobs in the interval $[1, 1 + nb]$ without idle time,

(5) machine M_2 processes jobs in the interval $[1 + b, 2 + nb]$ without idle time.

W.l.o.g. we can assume that the jobs $1, \dots, m-1$ are processed w.r.t. increasing numbers. Let $I_1 := \{i_1, \dots, i_k\}$ be the set of jobs scheduled between jobs 0 and 1. We have $I_1 \neq \emptyset$, since otherwise there would be an idle period on the robot between jobs 0 and 1, which contradicts (4).



In the following we will show that $k = 3$ and $\sum_{i \in I_1} a_i = b$ hold. We use the variables C^π_{ij} denoting the completion time of job j on machine M_i in the permutation π. The values of these variables for the jobs in the set I_1 are given by

$$C^\pi_{1i_\mu} = 1 + \sum_{\lambda=1}^{\mu} p_{1i_\lambda} < 1 + \mu \cdot \frac{b}{2} < 1 + \mu b \quad (\mu = 1, \ldots, k),$$

i.e. the robot transports jobs $0, i_1, \ldots, i_k$ without idle time in the interval $[1, 1+(k+1)b]$.

• If $k \leq 2$ holds, we have $C^\pi_{11} = 1 + \sum_{\lambda=1}^{k} p_{1i_\lambda} + 3b > 1 + 3b$ and the robot finishes the transportation of job i_k at time $1 + (k+1)b$ and, thus, not after time $1 + 3b$. Hence, the robot has an idle period between jobs i_k and 1, which contradicts (4).

• If $k \geq 4$ holds, we have

$$\sum_{\lambda=1}^{k} p_{2i_\lambda} < k \cdot \frac{b}{2} \leq k \cdot \frac{b}{2} + (k-4) \cdot \frac{b}{2} = (k-2)b.$$

On the other hand, job 1 cannot start on M_2 earlier than time $1 + (k+2)b$, since jobs 0,1 and all jobs of I_1 have to be transported before. Thus, the time period between the completion time $C^\pi_{20} = 1 + 4b$ of job 0 on M_2 and the starting time of job 1 on M_2 is not completely filled with jobs from I_1, which contradicts (5).

Thus, we must have $k = 3$. This implies that job 1 is transported by the robot in the interval $[1 + 4b, 1 + 5b]$, i.e. job 1 starts on M_1 not after time $1 + b$.
Therefore, $\sum_{i \in I_1} p_{1i} \leq b$. On the other hand, job 1 starts on M_2 not before $1 + 5b$. Since we have no idle time on M_2 in $[1 + b, 2 + nb]$, we must have $4b + 1 + \sum_{\lambda=1}^{3} p_{2i_\lambda} \geq 1 + 5b$. Therefore, $\sum_{i \in I_1} p_{2i} \geq b$. Since $p_{1i} = p_{2i} = a_i$, we have $\sum_{i \in I_1} a_i = b$.

Analogously, we show that the remaining sets $I_2, \ldots, I_m$ separated by the jobs $1, \ldots, m$ contain 3 elements and fulfill $\sum_{i \in I_j} a_i = b$ for $j = 2, \ldots, m$. Thus, $I_1, \ldots, I_m$ define a solution of 3-PART. □

5.1.2 Problem $F2, R1 \mid p_{ij} = p, t_j \mid C_{\max}$

In problem $F2, R1 \mid p_{ij} = p, t_j \mid C_{\max}$ we have constant processing times of all operations on M_1 and M_2, but arbitrary transportation times between the machines. For the sake of simplicity we normalize the data in such a way that we have unit processing times on

M_1 and M_2, but rational, job-dependent transportation times $t_j \geq 0$ from M_1 to M_2. Now, the objective value (5.1) of a permutation π reduces to

$$\begin{aligned} F(\pi) &= \max_{k=1}^{n} \max_{l=k}^{n} \{k + \sum_{\lambda=k}^{l} t_{\pi_\lambda} + (n-l+1)\} \\ &= \max_{k=1}^{n} \max_{l=k}^{n} \{\sum_{\lambda=k}^{l} (t_{\pi_\lambda} - 1)\} + n + 2. \end{aligned}$$

If $\min_{j=1}^{n} \{t_j\} \geq 1$ holds, the transportation times dominate the processing times and we have $F(\pi) = \sum_{j=1}^{n} (t_j - 1) + n + 2$ for each permutation π. If $\max_{j=1}^{n} \{t_j\} \leq 1$ holds, the transportation times are dominated by the processing times and we have $F(\pi) = \max_{j=1}^{n} \{t_j - 1\} + n + 2$ for each permutation π. Since both values are a lower bound for the problem, each permuation is optimal in these cases.

Defining $s_j := t_j - 1$, the decision version of our problem corresponds to the following problem:

P1: Given are n rational numbers $s_1, \ldots, s_n \geq -1$ and a threshold value y. Does there exist a permutation π with $\tilde{F}(\pi) = \max_{k=1}^{n} \max_{l=k}^{n} \{\sum_{\lambda=k}^{l} s_{\pi_\lambda}\} \leq y$?

We will prove that our problem is $\mathcal{NP}$-hard by showing that P1 is $\mathcal{NP}$-complete.

Theorem 5.2: Problem $F2, R1 \mid p_{ij} = p, t_j \mid C_{\max}$ is $\mathcal{NP}$-hard in the strong sense.

Proof: We will show again that the strongly $\mathcal{NP}$-complete problem 3-PARTITION is reducible to problem P1.

W.l.o.g. we can assume that in the instance of 3-PART $b < 1$ holds (otherwise we scale the data in an appropriate way). Given an arbitrary instance of 3-PART, we construct the following instance of P1 with $n := 4m - 1$ integers:

$$\begin{aligned} s_i &:= \;\; a_i \quad \text{for } i = 1, \ldots, 3m \\ s_j &:= -b \quad \text{for } j = 3m+1, \ldots, 4m-1. \end{aligned}$$

We ask for a permutation π with $\tilde{F}(\pi) \leq y := b$.

Assume that $I_1, \ldots, I_m$ is a solution of 3-PART. Then we define

$$\pi := (\underset{b}{I_1}, \underset{-b}{3m+1}, \underset{b}{I_2}, \underset{-b}{3m+2}, \underset{b}{I_3}, \ldots, \underset{b}{I_{m-1}}, \underset{-b}{4m-1}, \underset{b}{I_m}).$$

Obviously, $\sum\limits_{\lambda=k}^{l} s_{\pi_\lambda} \leq b$ holds for all $1 \leq k \leq l \leq n$.

Conversely, assume that P1 has a solution π with $\tilde{F}(\pi) \leq b$. In π the $m-1$ items $3m+1, \ldots, 4m-1$ can divide the remaining items $\{1, \ldots, 3m\}$ into at most m subsets I_j. If less than m subsets are separated by these items, at least one subset I_j exists with $|I_j| \geq 4$ and $F(\pi) \geq \sum\limits_{i \in I_j} s_i > \sum\limits_{i \in I_j} \frac{b}{4} > 4 \cdot \frac{b}{4} = b$, which contradicts $\tilde{F}(\pi) \leq b$.

Thus, π must have the form $\pi = (I_1, 3m+1, I_2, 3m+2, I_3, \ldots, I_{m-1}, 4m-1, I_m)$ with m subsets $I_1, \ldots, I_m$ each containing three elements from $\{1, \ldots, 3m\}$. For all these sets I_j we have $\sum\limits_{i \in I_j} s_i \leq b$, since $\tilde{F}(\pi) \leq b$. From $\sum\limits_{j=1}^{m} \sum\limits_{i \in I_j} s_i = mb$ we get the equality $\sum\limits_{i \in I_j} s_i = b$. Thus, the sets $I_1, \ldots, I_m$ define a solution of 3-PART. $\square$

5.1.3 Problem $F2, R1 \mid p_{ij} = p, t_j \in \{T_1, T_2\} \mid C_{\max}$

In this subsection we consider a special case of the problem discussed in the previous subsection where we have unit processing times on the machines and the job-dependent transportation times t_j may take only two rational values T_1 and T_2. W.l.o.g. assume $T_1 < T_2$. If $T_1, T_2 \leq 1$ or $T_1, T_2 \geq 1$, each permutation is optimal, as observed in the previous subsection. Therefore, it remains to consider the case $T_1 < 1, T_2 > 1$.

For the considered problem the objective value of a permutation π can be calculated by

$$F(\pi) = \max_{k=1}^{n} \max_{l=k}^{n} \{\sum_{\lambda=k}^{l} s_{\pi_\lambda} + n + 2\} = \max_{k=1}^{n} \max_{l=k}^{n} \{H^\pi(k,l)\} \tag{5.3}$$

with $H^\pi(k,l) = \sum\limits_{\lambda=k}^{l} s_{\pi_\lambda} + n + 2$ and $s_j := t_j - 1$ for $j = 1, \ldots, n$.

Let $S_1 := T_1 - 1 < 0$ and $S_2 := T_2 - 1 > 0$. A job with $s_j = S_2$ will be called an "S_2-job"; a job with $s_j = S_1$ will be called an "S_1-job". Since we have only two types of jobs, we may decompose a permutation π into "S_2"-blocks and "S_1-blocks", where a block consists of a maximal number of consecutive jobs in π with the same s_j-value. For example, the permutation π with $s_\pi = (S_2, S_2, S_1, S_2, S_2, S_2, S_1, S_1, S_2)$ contains the blocks $(S_2, S_2), (S_1), (S_2, S_2, S_2), (S_1, S_1)$, and (S_2).

In the following lemmata we will show that the search for an optimal permutation π may be reduced to permutations with a special decomposition into blocks.

Lemma 5.1: It is sufficient to consider only permutations π with $s_{\pi_1} = s_{\pi_n} = S_2$.

Proof: Assume that $s_{\pi_1} = S_1$ and $s_\pi = (S_1, \ldots, S_1, S_2, \ldots)$. Let $\tilde{\pi}$ be the permutation we obtain by swapping the first S_1-block with the first S_2-job π_{k^*} leading to $s_{\tilde{\pi}} =$

$(S_2, S_1, \ldots, S_1, \ldots)$. Then we have $H^{\tilde{\pi}}(1,1) = S_2 + n + 2 = H^{\pi}(k^*, k^*)$ and $H^{\tilde{\pi}}(1,l) \leq H^{\tilde{\pi}}(1,1) = H^{\pi}(k^*, k^*)$ for all $l \leq k^*$. For all other values of k and l we get $H^{\tilde{\pi}}(k,l) \leq H^{\pi}(k,l)$ and therefore, $F(\tilde{\pi}) = \max H^{\tilde{\pi}}(k,l) \leq \max H^{\pi}(k,l) = F(\pi)$. The case $s_{\pi_n} = S_1$ can be treated in the same way. □

Since $S_1 < 0$ and $S_2 > 0$, we have

Lemma 5.2: The objective value (5.3) of a permutation π is achieved by a pair k, l, where π_k is the first job and π_l the last job of an S_2-block.

Let q be the number of jobs j with $t_j = T_2$. Then we have $n - q$ jobs with $t_j = T_1$. In the following we distinguish two cases.

- **Case 1:** $q > \frac{n}{2}$, i.e. we have more S_2-jobs than S_1-jobs.

Lemma 5.3: Let $q > \frac{n}{2}$ and $b := \lceil \frac{q}{r} \rceil$ with $r := n - q + 1$. Then an optimal permutation exists which contains r S_2-blocks of lengths b or $b - 1$, separated by single S_1-jobs.

Proof: Analogously to the proof of Lemma 5.1, we can show that we need not consider permutations in which two S_1-jobs are neighbored. Therefore we can conclude that we have $r = n - q + 1$ S_2-blocks, separated by single S_1-jobs. Thus, π has the form $s_\pi = (B_1^\pi, S_1, B_2^\pi, S_1, \ldots, S_1, B_r^\pi)$ with S_2-blocks B_i^π of length b_i^π $(i = 1, \ldots, r)$.

It remains to show that we only have blocks of lengths b and $b - 1$. Assume that there is no optimal permutation with only blocks of these lengths. Then in each optimal permutation we have at least one S_2-block B_i^π with length $b_i^\pi \geq b + 1$ or one block with $b_i^\pi < b - 1$. Let π be an optimal permutation with a block of length $b_i^\pi \geq b + 1$. Then there must be at least one block B_j^π of length $b_j^\pi \leq b - 1$ in π. Choose the blocks B_i^π, B_j^π such that $|i - j|$ is minimal, i.e. all blocks between B_i^π and B_j^π (if any) have length b. Let us assume $j > i$; the other case can be treated in the same way.

Due to Lemma 5.2 the objective value of π can be written as

$$F(\pi) = \max_{k=1}^{r} \max_{l=k}^{r} \{\sum_{\lambda=k}^{l} (b_\lambda^\pi \cdot S_2 + S_1) - S_1 + n + 2\} =: \max_{k=1}^{r} \max_{l=k}^{r} A^\pi(k,l). \tag{5.4}$$

Let $\tilde{\pi}$ be the permutation obtained from π by moving one S_2-job from B_i^π to B_j^π. Then

$$A^{\tilde{\pi}}(k,l) = \begin{cases} A^\pi(k,l), & k \leq i, l \geq j \text{ or } k > i, l < j \text{ or } k,l > j \text{ or } k,l < i \\ A^\pi(k,l) - S_2, & k \leq i \leq l < j \\ A^\pi(k,l) + S_2, & i < k \leq j \leq l \end{cases}.$$

Thus, we have $A^{\tilde{\pi}}(k,l) > A^\pi(k,l)$ only for $i < k \leq j \leq l$. However, for these cases we can find other terms which dominate the $A^{\tilde{\pi}}(k,l)$-values.

If $b \cdot S_2 + S_1 \geq 0$, then

$$A^{\tilde{\pi}}(k,l) = A^{\tilde{\pi}}(i,l) - \sum_{\lambda=i}^{k-1}(b_\lambda^{\tilde{\pi}} \cdot S_2 + S_1) \leq A^\pi(i,l) - (k-i)(b \cdot S_2 + S_1) \leq A^\pi(i,l).$$

If $b \cdot S_2 + S_1 < 0$, we distinguish the following three cases:

- $l > j$: $A^{\tilde{\pi}}(k,l) = A^\pi(j+1,l) + \sum_{\lambda=k}^{j}(b \cdot S_2 + S_1) \leq A^\pi(j+1,l)$
- $l = j > k$: $A^{\tilde{\pi}}(k,j) = A^\pi(k,j-1) + (b \cdot S_2 + S_1) \leq A^\pi(k,j-1)$
- $l = j = k$: $A^{\tilde{\pi}}(j,j) = b \cdot S_2 + n + 2 \leq (b+1) \cdot S_2 + n + 2 = A^\pi(i,i)$.

Therefore $F(\tilde{\pi}) = \max A^{\tilde{\pi}}(k,l) \leq \max A^\pi(k,l) = F(\pi)$.

The case in which a block B_i^π of length $b_i^\pi < b-1$ exists can be treated in a similar way. The result of the lemma follows by induction. □

Lemma 5.3 states that there is an optimal permutation in which the q S_2-jobs are partitioned into r blocks of lengths b and $b-1$. Thus, the numbers x_b and x_{b-1} of blocks with lengths b and $b-1$ are given by $x_{b-1} = rb - q$ and $x_b = n - rb + 1$. Therefore the problem reduces to the problem of ordering r blocks, x_b with length b and x_{b-1} with length $b-1$, such that (5.4) is minimized. If we define $\hat{s}_j := b \cdot S_2 + S_1$ for all blocks of length b and $\hat{s}_j := (b-1) \cdot S_2 + S_1$ for all blocks of length $b-1$, we again have the problem of sequencing r numbers $\hat{s}_j \in \{\hat{S}_1, \hat{S}_2\}$ with $\hat{S}_1 := (b-1) \cdot S_2 + S_1 < \hat{S}_2 := b \cdot S_2 + S_1$ such that

$$F(\pi) = \max_{k=1}^{r} \max_{l=k}^{r} \{\sum_{\lambda=k}^{l} \hat{s}_{\pi_\lambda} + n + 2\} - S_1 \tag{5.5}$$

is minimized. The objective functions (5.3) and (5.5) differ only in the constant S_1. Therefore the problem can be solved recursively.

- **Case 2:** $q \leq \frac{n}{2}$

Lemma 5.4: Let $q \leq \frac{n}{2}$ and $b' := \lceil \frac{n-q}{r'} \rceil$ with $r' := q-1$. Then an optimal permutation exists which contains r' S_1-blocks of lengths b' or $b'-1$, separated by single S_2-jobs.

Proof: Analogously to Case 1 we can show that we need not consider permutations in which two S_2-jobs are neighbored. Therefore, we can conclude that we have $r' = q-1$ S_1-blocks, separated by single S_2-jobs. Thus, π has the form $s_\pi = (S_2, A_1^\pi, S_2, A_2^\pi, S_2, \ldots, S_2, A_{r'}^\pi, S_2)$ with S_1-blocks A_i^π of length a_i^π.

The objective value of π can be written as

$$F(\pi) = \max_{k=1}^{r'} \max_{l=k}^{r'} \{\sum_{\lambda=k}^{l}(a_\lambda^\pi \cdot S_1 + S_2) + S_2 + n + 2\}. \tag{5.6}$$

The proof that it is sufficient to consider only permutations with blocks of lengths b' and $b'-1$ can be done analogously to the proof of Lemma 5.3. $\square$

Lemma 5.4 states that there is an optimal permutation in which the $n-q$ S_1-jobs are partitioned into r' blocks of lengths b' and $b'-1$. Thus, the numbers $y_{b'}$ and $y_{b'-1}$ of blocks with lengths b' and $b'-1$ are given by $y_{b'-1} = r'b' + q - n$ and $y_{b'} = n - r'b' - 1$. Therefore the problem reduces to the problem of ordering r' blocks, $y_{b'}$ with length b' and $y_{b'-1}$ with length $b'-1$, such that (5.6) is minimized. If we define $\hat{s}_j := b' \cdot S_1 + S_2$ for all blocks of length b' and $\hat{s}_j := (b'-1) \cdot S_1 + S_2$ for all blocks of length $b'-1$, we have to sequence r' numbers $\hat{s}_j \in \{\hat{S}_1, \hat{S}_2\}$ with $\hat{S}_1 := b' \cdot S_1 + S_2 < \hat{S}_2 := (b'-1) \cdot S_1 + S_2$ such that

$$F(\pi) = \max_{k=1}^{r'} \max_{l=k}^{r'} \{\sum_{\lambda=k}^{l} \hat{s}_{\pi_\lambda} + n + 2\} + S_2 \tag{5.7}$$

is minimized. The objective functions (5.3) and (5.7) differ only in the constant S_2. Therefore, also in the second case the problem can be solved recursively.

Using these results we can formulate the recursive procedure `Blocks` (n, q, S_1, S_2), which solves our problem.

```
PROCEDURE Blocks (n, q, S_1, S_2)
    IF n = 1 OR q = 1 OR S_1, S_2 ≥ 0 OR S_1, S_2 ≤ 0 THEN RETURN;
    IF q > n/2 THEN BEGIN
        r := n - q + 1;  b := ⌈q/r⌉;
        x_b := n - rb + 1;  x_{b-1} := rb - q;
        Ŝ_1 := (b - 1) · S_2 + S_1;  Ŝ_2 := b · S_2 + S_1;
        CALL Blocks (r, x_b, Ŝ_1, Ŝ_2);
    END
    ELSE BEGIN
        r' := q - 1;  b' := ⌈(n-q)/r'⌉;
        y_{b'} := n - r'b' - 1;  y_{b'-1} := r'b' + q - n;
        Ŝ_1 := b' · S_1 + S_2;  Ŝ_2 := (b' - 1) · S_1 + S_2;
        CALL Blocks (r', y_{b'-1}, Ŝ_1, Ŝ_2);
    END
```

Theorem 5.3: Procedure `Blocks` $(n, q, T_1 - 1, T_2 - 1)$ solves problem $F2, R1 \mid p_{ij} = 1, t_j \in \{T_1, T_2\} \mid C_{\max}$ in $O(\log n)$ time.

Proof: The optimality follows from the preceding considerations. It remains to prove the complexity. In Case 1 we have $r = n - q + 1 < n - \frac{n}{2} + 1 = O(\frac{n}{2})$. In Case 2 we have $r' = q - 1 < \frac{n}{2} - 1 = O(\frac{n}{2})$. Thus, in each case the number of jobs reduces by the factor 2. Therefore procedure `Blocks` (n, q, S_1, S_2) which can be executed in constant time is called at most $O(\log n)$ times. Since the input length of the problem is $O(\log n)$, the algorithm is polynomial. $\square$

Example 5.1: Let $n = 78$, $q = 33$, $T_1 = 0$ and $T_2 = 2.4$.

- 1. Iteration: `Blocks` $(78, 33, -1.0, 1.4)$: $q \leq \frac{n}{2}$, i.e. we get Case 2:

$$r' = 32, b' = 2, y_{b'} = 13, y_{b'-1} = 19, \hat{S}_1 = -0.6, \hat{S}_2 = 0.4.$$

  This yields 13 S_1-blocks of length 2 and 19 of length 1. These 32 blocks are separated by the 33 S_2-jobs. Let $\sigma_1^1 = (S_1, S_1, S_2)$ and $\sigma_2^1 = (S_1, S_2)$. Then the optimal permutation in this step has the form $\pi^1 = (S_2, \pi^2)$ where π^2 is the optimal permutation of sequences σ_1^1, σ_2^1 calculated in the next iteration.

- 2. Iteration: `Blocks` $(32, 19, -0.6, 0.4)$: $q > \frac{n}{2}$, i.e. we get Case 1:

$$r = 14, b = 2, x_b = 5, x_{b-1} = 9, \hat{S}_1 = -0.2, \hat{S}_2 = 0.2.$$

  This yields 5 σ_2^1-blocks of length 2 and 9 of length 1. These 14 blocks are separated by the 13 σ_1^1-blocks. Let $\sigma_1^2 = (\sigma_2^1, \sigma_1^1)$ and $\sigma_2^2 = (\sigma_2^1, \sigma_2^1, \sigma_1^1)$. Then the optimal permutation in this step has the form $\pi^2 = (\pi^3$ without the last sequence $\sigma_1^1)$ where π^3 is the optimal permutation of sequences σ_1^2, σ_2^2 calculated in the next iteration.

- 3. Iteration: `Blocks` $(14, 5, -0.2, 0.2)$: $q \leq \frac{n}{2}$, i.e. we get Case 2:

$$r' = 4, b' = 3, y_{b'} = 1, y_{b'-1} = 3, \hat{S}_1 = -0.4, \hat{S}_2 = -0.2.$$

  This yields 1 σ_1^2-block of length 3 and 3 of length 2. These 4 blocks are separated by the 5 σ_2^2-blocks. Let $\sigma_1^3 = (\sigma_1^2, \sigma_1^2, \sigma_1^2, \sigma_2^2)$ and $\sigma_2^3 = (\sigma_1^2, \sigma_1^2, \sigma_2^2)$. Then the optimal permutation in this step has the form $\pi^3 = (\sigma_2^2, \pi^4)$ where π^4 is the optimal permutation of sequences σ_1^3, σ_2^3 calculated in the next iteration.

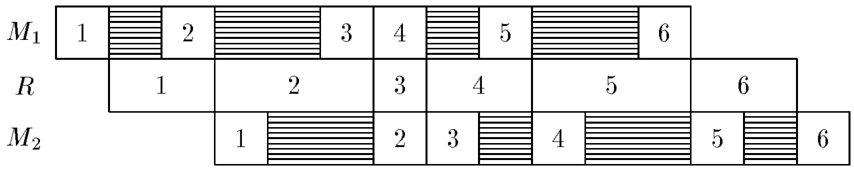
- 4. Iteration: `Blocks` $(4, 3, -0.4, -0.2)$: Since $S_1, S_2 \leq 0$, the recursion terminates and each permutation of the current sequences is optimal. We may set $\pi^4 = (\sigma_1^3, \sigma_2^3, \sigma_2^3, \sigma_2^3)$ and obtain the optimal permutation $\pi = \pi^1$. $\square$

5.1.4 Problem $F2, R1 \mid p_{ij} = 1, t_j \mid C_{\max}$

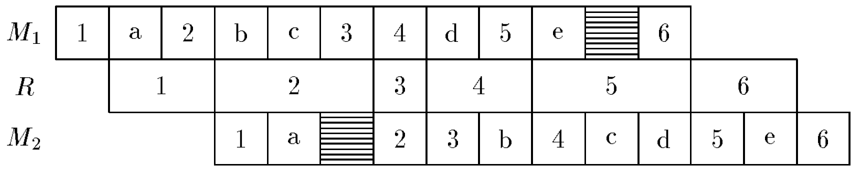
In this subsection we consider another special case of problem $F2, R1 \mid p_{ij} = p, t_j \mid C_{\max}$ where $p = 1$. All transportation times t_j are supposed to be non-negative integers. If $t_j > 0$ for all jobs j holds, we have $\min_{j=1}^{n} \{t_j\} \geq 1$ due to integrality. Thus, in this case the transportation times dominate the processing times as discussed before and each permutation is optimal. In the other case where some t_j-values are zero, we assume that $t_j = 0$ does not mean that job j does not have to be transported, but means that the job needs the robot for a very small time, i.e. the robot has to be available for transportation between processing on M_1 and M_2.

To solve the problem we divide the jobs j into two disjoint sets $J^+ := \{j \mid t_j > 0\}$ and $J^0 := \{j \mid t_j = 0\}$. In the following we describe an algorithm which constructs an optimal permutation plan. The algorithm is illustrated by an example where the jobs from J^+ are numbered from $1, \dots, 6$ and the jobs from J^0 by a, …, e.

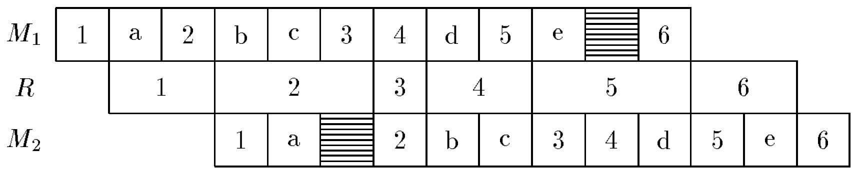
1. Choose an arbitrary permutation π^+ of the jobs from J^+. Plan all jobs in such a way that the robot transports without idle time and such that each job is processed on M_1 immediately before transportation and on M_2 immediately after transportation.



2. Insert all jobs from J^0 into the empty time periods in π^+ on M_1 and as early as possible into the empty time periods on M_2. If there are more jobs than empty time periods on M_1, put the remaining jobs to the end of π^+.



3. Reorder the jobs on M_2 according to the permutation π of the jobs on M_1, resulting in a permutation plan with the same makespan.



The makespan of the schedule resulting after Step 1 provides a lower bound for the optimal makespan. It is given by $F(\pi^+) = 2 + \sum\limits_{j \in J^+} t_j$.

Theorem 5.4: The described algorithm constructs an optimal permutation π for problem $F2, R1 \mid p_{ij} = 1, t_j \mid C_{\max}$ with makespan

$$F(\pi) = \max \{2 + \sum_{j \in J^+} t_j, \max_{j \in J^+} t_j + n + 1\}.$$

Proof: Consider the schedule after Step 2 (which has the same makespan as the final schedule). Let l be the last job from J^+ which has an idle time immediately before its processing on M_2.

- Case 1: No job of J^0 starts after l on M_2. In this case the makespan of the schedule is given by the makespan of the schedule after Step 1, i.e.

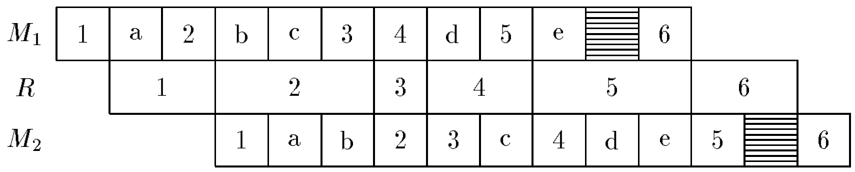
$$F(\pi) = 2 + \sum_{j \in J^+} t_j. \tag{5.8}$$

- Case 2: Jobs of J^0 are scheduled after l on M_2. By construction all jobs scheduled before l on M_1 are also scheduled before l on M_2 (due to the idle time before l). Thus, jobs of J^0 are also scheduled after l on M_1 and no idle times occur before l on M_1. Therefore, the makespan of the schedule after Step 2 is given by the complete processing and transportation of job l plus the number of jobs scheduled before l on M_1 plus the number of jobs scheduled after l on M_2. Since each job $j \neq l$ is either scheduled before l on M_1 or after l on M_2 (but not both), we get

$$F(\pi) = 1 + t_l + 1 + (n-1) = t_l + n + 1. \tag{5.9}$$

Furthermore, $F(\pi) \geq t_j + n + 1$ for all $j \in J^+$, i.e. $F(\pi) = \max_{j \in J^+} t_j + n + 1$. Since (5.8) and (5.9) are lower bounds for the optimal makespan, the constructed schedule is optimal. □

Remark: If $t_j = 0$ means that the robot is not needed for the transportation of job j, an optimal permutation plan does not necessarily exist. Obviously, Steps 1 and 2 of our algorithm solve the problem. Note that by planning the jobs with $t_j = 0$ as early as possible in the empty time periods on M_2 in Step 2, some jobs may overtake others during their transportation. For the example from above we get the following schedule where jobs b and d overtake jobs 2 and 5, respectively.



5.1.5 Problem $F2, R1 \mid p_{ij} = p, r_j, t_j = T \mid f$

In this subsection we consider problem $F2, R1 \mid p_{ij} = p, r_j, t_j = T \mid f$ with an arbitrary regular objective function f and show that this problem is equivalent to a single-machine problem.

If $T \leq p$ holds, then the robot is dominated by the machines M_1 and M_2 and a job never has to wait for transportation. Since the objective function is regular, it is sufficient to consider active, no-wait permutation plans where idle times on M_1 only occur due to release dates. Hence, if the starting times on M_1 in a feasible permutation are S_{1j} $(j = 1, \ldots, n)$, then the starting times on M_2 are given by $S_{2j} = S_{1j} + p + T$. Thus, we may set $r'_j = r_j + p + T$ and obtain the single-machine problem $1 \mid p_j = p, r_j \mid f$ for machine M_2.

If $T > p$ holds, then the robot dominates both machines M_1 and M_2 as follows. Each job can be processed on M_2 immediately after transportation, i.e. we can eliminate M_2 from consideration. Furthermore, we can eliminate M_1 by setting $r'_j = r_j + p$. Then scheduling the robot is equivalent to the single-machine problem $1 \mid p_j = p, r_j \mid f$ with processing times T.

From this equivalence we get the following results:

- Problems $F2, R1 \mid p_{ij} = p, r_j, t_j = T \mid L_{\max}$, $F2, R1 \mid p_{ij} = p, r_j, t_j = T \mid \sum C_j$ and $F2, R1 \mid p_{ij} = p, r_j, t_j = T \mid \sum w_j U_j$ are polynomially solvable by the polynomial algorithms for the single-machine problems $1 \mid p_j = p, r_j \mid L_{\max}$ (Simons [1978]), $1 \mid p_j = p, r_j \mid \sum C_j$ (Simons [1983]), and $1 \mid p_j = p, r_j \mid \sum w_j U_j$ (Baptiste [1998]),
- problem $F2, R1 \mid p_{ij} = p, t_j = T \mid \sum w_j T_j$ is polynomially solvable, since the corresponding single-machine problem $1 \mid p_j = p \mid \sum w_j T_j$ can be formulated as a polynomially solvable assignment problem (cf. Brucker [1998]).

5.2 Flow-shop problems with m machines

In this section we consider problems with an arbitrary number m of machines. Since problem $F, R1 \mid p_{ij} = p, t_{jk} \mid C_{\max}$ is already $\mathcal{NP}$-hard for $m = 2$, we study a problem with job-independent transportation times $t_{jk} = t_k$ and show that problem $F, R1 \mid n \geq m - 1, p_{ij} = p, t_k \mid C_{\max}$ is polynomially solvable. In Subsection 5.2.2 we give a general reduction from single-machine problems to flow-shop problems with a fixed number of machines, unit processing times and arbitrary transportation times. Using this reduction several $\mathcal{NP}$-hardness results are derived.

5.2.1 Problem $F, R1 \mid n \geq m - 1, p_{ij} = p, t_k \mid C_{\max}$

In this subsection we consider problem $F, R1 \mid n \geq m - 1, p_{ij} = p, t_k \mid C_{\max}$ with m machines, constant processing times and job-independent transportation times. In this problem we assume $n \geq m-1$, i.e. the number of jobs is greater than or equal to the number of transportation stages. From the practical point of view, this case is also the most relevant case, since in general the number of jobs will exceed the number of machines.

Again, we normalize the data in such a way that all jobs have unit processing times on each machine and the transportation times become rational numbers.

It is sufficient to consider only permutation plans, since the jobs are identical and overtaking does not reduce the completion times. Furthermore, the objective value is independent of the permutation and depends only on the sequence of the robot. We describe the robot sequence by a vector $Z = (z_1, \ldots, z_{n(m-1)})$ with $z_\lambda = k$ iff the λ-th move of the robot transports the next available job from M_k to M_{k+1}. The corresponding schedule is obtained by starting all jobs on the machines as early as possible.

For $k = 1, \ldots, m-1$ let $a^k := (1, \ldots, k)$ and $\hat{a}^k := (k, \ldots, m-1)$ be the special sequences in which the robot serves the machines in the order $1, 2, \ldots, k$ and $k, k+1, \ldots, m-1$, respectively. In the following we will show that there exists an optimal robot sequence which contains only these subsequences. Let

$$Z^* := (a^1, a^2, \ldots, a^{m-2}, \underbrace{a^{m-1}, a^{m-1}, \ldots, a^{m-1} = \hat{a}^1}_{n-m+2}, \hat{a}^2, \hat{a}^3, \ldots, \hat{a}^{m-1}).$$

Z^* consists of

- a “building-up” phase where in each step one more machine is served,
- an “identity” phase where all machines are served regularly, and
- a “building-down” phase where in each step one machine less is served.

In Figure 5.1 an example for this robot sequence with $n = 6$ and $m = 4$ is given, which in general is not a first-come-first-served sequence. The next theorem proves that this sequence is optimal.

Theorem 5.5: The sequence Z^* defines an optimal robot sequence for problem $F, R1 \mid n \geq m-1, p_{ij} = p, t_k \mid C_{\max}$.

Proof: Let $l \in \{0, \ldots, m-1\}$ be the index with $\sum\limits_{\lambda=1}^{l} t_\lambda \leq 1$ and $\sum\limits_{\lambda=1}^{l+1} t_\lambda > 1$, where $l = 0$ if $t_1 > 1$ and $l = m-1$ if $\sum\limits_{k=1}^{m-1} t_k \leq 1$. In the following we will calculate the objective value of the schedule associated with Z^*.

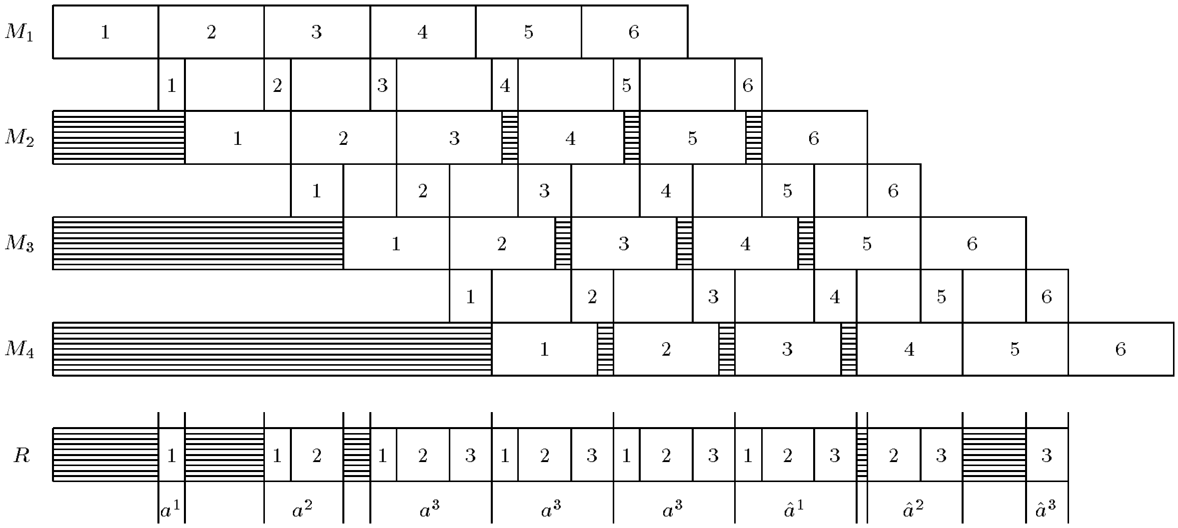


Figure 5.1: The robot sequence Z^*

At first we consider the sequence $(a^1, a^2, \ldots, a^l)$. The transportations of a^1 can be done in the interval $[1, 1+t_1]$, those of a^2 in $[2, 2+t_1+t_2]$, and those of a^k in $[k, k+\sum_{\lambda=1}^{k} t_\lambda]$ for $k = 1, \ldots, l$. When the robot has transported a job from M_j to M_{j+1} it is back at M_j after at most one time unit. Thus, the next job completed on M_j after one time unit does not have to wait for the robot, i.e. we have a no-wait plan. Only the robot has a waiting time between a^k and a^{k+1} of length $\tau_k := 1 - \sum_{\lambda=1}^{k} t_\lambda$ for $k = 1, \ldots, l$ (cf. Figure 5.2).

- Case 1: $l = m-1$
  In this case each job in the partial sequence $(\underbrace{a^{m-1}, \ldots, a^{m-1}}_{n-m+2})$ is again transported immediately after finishing on a machine, i.e. we have a no-wait plan.
- Case 2: $l < m-1$
  We consider the sequence $(a^{l+1}, \ldots, a^{m-1})$. The transportations of a^{l+1} can be done in the interval $[l+1, l+1+\sum_{\lambda=1}^{l+1} t_\lambda]$, those of a^{l+2} in $[l+1+\sum_{\lambda=1}^{l+1} t_\lambda, l+1+\sum_{\lambda=1}^{l+1} t_\lambda + \sum_{\lambda=1}^{l+2} t_\lambda]$, and those of a^{l+k} in $[l+1+\sum_{\mu=1}^{k-1}\sum_{\lambda=1}^{l+\mu} t_\lambda, l+1+\sum_{\mu=1}^{k}\sum_{\lambda=1}^{l+\mu} t_\lambda]$ for $k = 1, \ldots, m-1-l$, since $\sum_{\lambda=1}^{l+1} t_\lambda > 1$. This means that the robot transports jobs in the sequence $(a^{l+1}, \ldots, a^{m-2}, \underbrace{a^{m-1}, a^{m-1}, \ldots, a^{m-1} = \hat{a}^1}_{n-m+2})$ without idle time and all machines are served regularly.

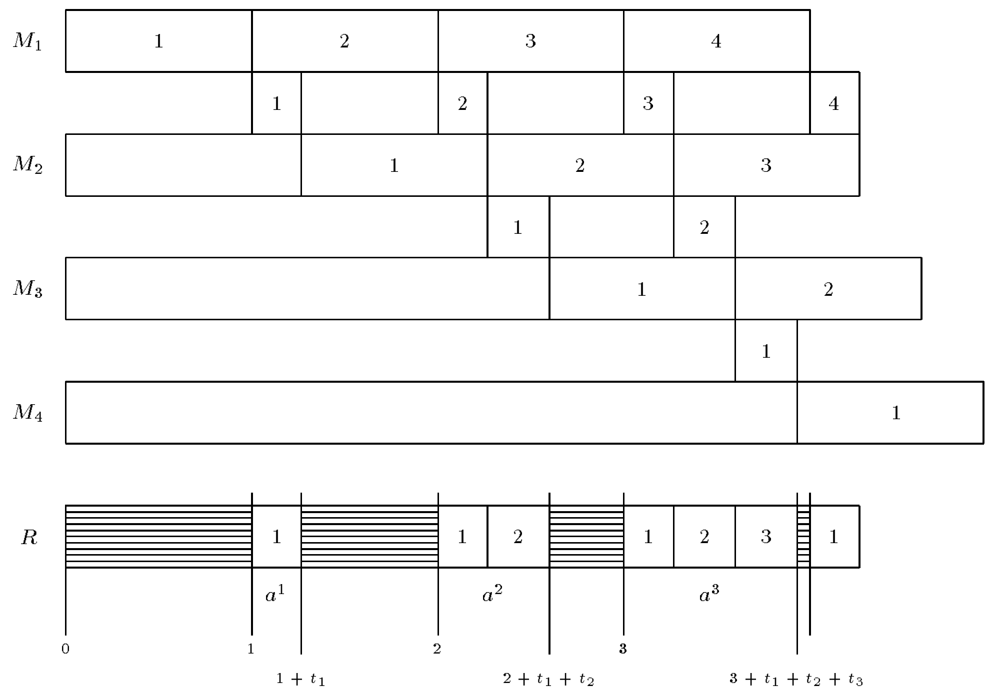


Figure 5.2: The no-wait part of robot sequence Z^*

Symmetrically, let $\hat{l} \in \{0, \ldots, m-1\}$ be the index with $\sum\limits_{\lambda=m-\hat{l}}^{m-1} t_\lambda \leq 1$ and $\sum\limits_{\lambda=m-\hat{l}-1}^{m-1} t_\lambda > 1$, where $\hat{l} = 0$ if $t_{m-1} > 1$.

Analogously we can show that the robot is always busy in the sequence $(\hat{a}^1, \ldots, \hat{a}^{m-\hat{l}-1})$. Furthermore, from index $m - \hat{l}$ up to index $m-1$ we have a no-wait plan and the robot only has waiting times $\hat{\tau}_k := 1 - \sum\limits_{\lambda=k}^{m-1} t_\lambda$ for $k = 1, \ldots, \hat{l}$ between $\hat{a}^{m-k-1}$ and $\hat{a}^{m-k}$.

In the first case $(l = m-1)$, the whole schedule is a no-wait schedule with

$$F(Z^*) = n + m - 1 + \sum_{k=1}^{m-1} t_k,$$

where each job is transported immediately after processing. Since a no-wait schedule provides a lower bound for the optimal makespan, Z^* must be optimal.

In the second case $(l < m-1)$, Z^* results in the following working and waiting time sequence for the robot:

$$(a^1, \tau_1, a^2, \ldots, a^l, \tau_l, a^{l+1}, a^{l+2}, \ldots, a^{m-2}, \underbrace{a^{m-1}, \ldots, a^{m-1} = \hat{a}^1}_{n-m+2},$$
$$\hat{a}^2, \ldots, \hat{a}^{m-\hat{l}-1}, \hat{\tau}_{\hat{l}}, \hat{a}^{m-\hat{l}}, \ldots, \hat{a}^{m-2}, \hat{\tau}_1, \hat{a}^{m-1}).$$

Its objective value consists of processing the first job on M_1, the last job on M_m, all transportation times and some waiting times for the robot, i.e.

$$F(Z^*) = 2 + n\sum_{k=1}^{m-1} t_k + \sum_{k=1}^{l} \tau_k + \sum_{k=1}^{\hat{l}} \hat{\tau}_k.$$

We claim that all waiting times are unavoidable. We have shown that the first sum $\sum_{k=1}^{l} \tau_k$ results from a no-wait schedule for all transported jobs in the sequence $(a^1, \ldots, a^l)$. If we had an unlimited number of robots, we could not improve this schedule since each job is already transported as early as possible. Thus, the waiting times for the single robot are unavoidable. Symmetrically, the waiting times $\sum_{k=1}^{\hat{l}} \hat{\tau}_k$ are unavoidable, i.e. Z^* is optimal. □

Remark: For the case $n < m-1$ the proof of the result in Theorem 5.5 cannot be adapted. It is an open question whether an analogous result also holds for this case (we did not succeed in proving such a result, but neither did we find a counterexample).
Finke et al. [1996] considered a robotic flow-shop cell with a single unit of buffer behind each machine where all jobs are identical, i.e. $p_{ij} = p_i$ holds. They showed that repeating the sequence a^{m-1} from the middle part of Z^* minimizes the mean cycle time C_t in repetitive manufacturing where initially all machines are occupied by a job. Note that in our algorithm a single unit of buffer behind each machine is also sufficient. The main difference between the two results is the objective function. While minimizing the cycle time C_t in an environment with $p_{ij} = p$ becomes trivial ($C_t = \max\{p, \sum_{k=1}^{m-1} t_k\}$), it is not obvious how to sequence the robot in a schedule with minimal makespan when not all machines are initially occupied by a job. Note that the sequence Z^* is not makespan-optimal in the more general case $p_{ij} = p_i$.

5.2.2 Problem $Fm, R1 \mid p_{ij} = 1, r_j, t_{jk} \mid f$

In this subsection we consider problem $Fm, R1 \mid p_{ij} = 1, r_j, t_{jk} \mid f$ with an arbitrary, but fixed number $m \geq 2$ of machines (i.e. m is not part of the input), unit processing times, release dates, arbitrary transportation times t_{jk} and an objective $f \in \{C_{\max}, L_{\max}, \sum C_j, \sum w_j C_j, \sum U_j, \sum w_j U_j, \sum T_j, \sum w_j T_j\}$. We show that this problem is at least as difficult as the single-machine problem $1 \mid r_j \mid f$.

Theorem 5.6: For all objective functions $f \in \{C_{\max}, L_{\max}, \sum C_j, \sum w_j C_j, \sum U_j, \sum w_j U_j, \sum T_j, \sum w_j T_j\}$ and any fixed number $m \geq 2$ of machines problem $1 \mid r_j \mid f$ reduces polynomially to $Fm, R1 \mid p_{ij} = 1, r_j, t_{jk} \mid f$.

Proof: In problem $Fm, R1 \mid p_{ij} = 1, r_j, t_{jk} \mid f$ the robot has to transport all jobs in $m-1$ transportation stages between machines M_k and M_{k+1} for $k = 1, \ldots, m-1$. The idea of our reduction is to transform the single-machine problem into the transport problem of the robot in the last stage $m-1$. This is done by defining the transportation times $t_{j,m-1}$ to be equal to the processing times from the single-machine problem and scaling them with a large, but polynomially bounded number a afterwards. By this scaling technique the last transportation stage becomes the bottleneck and dominates all other processing and transportation times.

In the following we will describe the reduction in more detail. We assume that we are given an instance I of problem $1 \mid r_j \mid f$ with n jobs, processing times p_j, release dates r_j, due dates d_j and weights w_j. If the objective function f does not contain due dates or weights, we assume $d_j = 0$ or $w_j = 1$.
Let $W := \sum\limits_{j=1}^{n} w_j$, $a := 2n^2mW$ and $b := (2m-3)n + 1$.

From I we construct the following instance I' of problem $Fm, R1 \mid p_{ij} = 1, r_j, t_{jk} \mid f$ with a fixed number $m \geq 2$ of machines (i.e. we cannot choose the value m as input in our reduction), n jobs and

$$t_{j1} = t_{j2} = \ldots = t_{j,m-2} := 1, \quad t_{j,m-1} := ap_j,$$

$$r'_j := ar_j - (2m-3), \quad d'_j := ad_j + b, \quad w'_j := w_j \qquad \text{for } j = 1, \ldots, n.$$

Obviously, this transformation is polynomial and holds for any fixed number m of machines.

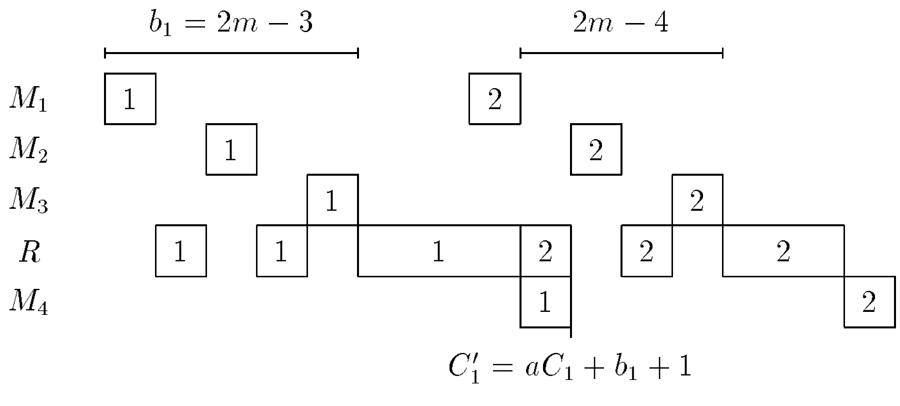


Figure 5.3: Transformed schedule S' for $m = 4$

A feasible single-machine schedule S for I with completion times C_j can be transformed into a feasible flow-shop schedule S' for I' as follows. We use the permutation of jobs induced by S on all machines $M_1, \ldots, M_m$. Furthermore, the robot first does all transportations for the first job, then all transportations for the second and finally all transportations for the last job. The resulting no-wait schedule S' (cf. Figure 5.3) is feasible w.r.t. the release dates r'_j and has completion times $C'_j = aC_j + b_j + 1$, where the value b_j corresponds to the sum of transportation and idle times of the robot before job j is transported in stage $m-1$. This value is equal to a sum of processing and transportation times which occur on the machines $M_1, \ldots, M_{m-1}$ and the transportation stages $1, \ldots, m-2$ before. If job j is scheduled at position $q \in \{1, \ldots, n\}$ in S', we have $b_j = (2m-3)+(q-1)(2m-4)$, which implies $b_j+1 \leq n(2m-3)+1 = b$ for all jobs j. Thus, if due dates d_j are given and $C_j \leq d_j$ in S holds, we have $C'_j = aC_j+b_j+1 \leq ad_j+b = d'_j$ in S'.

Conversely, a feasible flow-shop schedule S' for I' with completion times C'_j can be transformed into a feasible single-machine schedule S for I by scheduling the jobs according to the permutation of the jobs in the last transportation stage $m-1$. Obviously, for the resulting completion times C_j we have $aC_j \leq C'_j$. Thus, if due dates d_j are given and $C'_j \leq d'_j$ in S' holds, we have $aC_j \leq C'_j \leq d'_j = ad_j + b$ in S, i.e. $C_j \leq d_j + \frac{b}{a} < d_j + 1$, which implies $C_j \leq d_j$ due to integrality.

It remains to compare the objective values of S and S'. We can show:
For $f \in \{C_{\max}, L_{\max}, \sum C_j, \sum w_jC_j, \sum T_j, \sum w_jT_j\}$ we have

$$f(S) \leq y \text{ if and only if } f(S') \leq y' := ay + Wb. \tag{5.10}$$

For $f \in \{\sum U_j, \sum w_jU_j\}$ we have

$$f(S) \leq y \text{ if and only if } f(S') \leq y' := y. \tag{5.11}$$

We prove (5.10) for $f = \sum w_jC_j$. The other objective functions f are treated similarly.
If $f(S) = \sum w_jC_j \leq y$, then

$$f(S') = \sum w_jC'_j = \sum w_j(aC_j + b_j + 1) = a\sum w_jC_j + \sum w_j(b_j+1) \leq ay + Wb = y'.$$

If, on the other hand, $f(S') = \sum w_jC'_j \leq y'$, then

$$f(S) = \sum w_jC_j \leq \frac{1}{a}\sum w_jC'_j \leq \frac{y'}{a} = y + \frac{Wb}{a} \leq y + \frac{W(2mn)}{2n^2mW} = y + \frac{1}{n},$$

which implies $f(S) \leq y$ because $f(S)$ and y are assumed to be integers.

To prove (5.11) we can use the derived result that $C_j \leq d_j$ in S holds if and only if we have $C'_j \leq d'_j$ in S'. This equivalence implies that the sets of late jobs in S and S' are the same, i.e. the objective values $f(S)$ and $f(S')$ are equal for $f \in \{\sum U_j, \sum w_jU_j\}$. $\square$

By this general reduction some $\mathcal{NP}$-hardness results for flow-shop problems with any fixed number m of machines and a single robot can be derived. Problem $Fm, R1 \mid p_{ij} = 1, r_j, t_{jk} \mid f$ with $f \in \{L_{\max}, \sum C_j\}$ and problem $Fm, R1 \mid p_{ij} = 1, t_{jk} \mid \sum w_jT_j$ are strongly $\mathcal{NP}$-hard for each $m \geq 2$, since the corresponding single-machine problems are strongly $\mathcal{NP}$-hard. Problem $Fm, R1 \mid p_{ij} = 1, t_{jk} \mid f$ with $f \in \{\sum w_jU_j, \sum T_j\}$ is at least $\mathcal{NP}$-hard in the ordinary sense for each $m \geq 2$, since the corresponding single-machine problem is $\mathcal{NP}$-hard in the ordinary sense.

5.3 Summary and additional remarks

In the previous subsections we have derived new complexity results for flow-shop problems with a single robot. A summary of these results can be found in Table 5.1, where a $*$ again denotes that the problem is strongly $\mathcal{NP}$-hard. Note that the $\mathcal{NP}$-hardness of problem $F3, R1 \mid p_{ij} = p, t_{jk} \mid L_{\max}$ and, symmetrically, of $F3, R1 \mid p_{ij} = p, r_j, t_{jk} \mid C_{\max}$ can be shown by a reduction similar to the one presented in Subsection 5.1.2.

Nevertheless, there are still many open problems in this area; for an overview see the web-site

`http://www.mathematik.uni-osnabrueck.de/research/OR/class`.

Polynomially solvable:

$F2, R1 \mid p_{ij} = 1, t_j \mid C_{\max}$	Section 5.1.4
$F2, R1 \mid p_{ij} = p, t_j \in \{T_1, T_2\} \mid C_{\max}$	Section 5.1.3
$F, R1 \mid n \geq m-1, p_{ij} = p, t_k \mid C_{\max}$	Section 5.2.1
$F2, R1 \mid p_{ij} = p, r_j, t_j = T \mid \sum w_j C_j$	Section 5.1.5
$F2, R1 \mid p_{ij} = p, r_j, t_j = T \mid \sum w_j U_j$	Section 5.1.5
$F2, R1 \mid p_{ij} = p, r_j, t_j = T \mid \sum T_j$	Section 5.1.5
$F2, R1 \mid p_{ij} = p, t_j = T \mid \sum w_j T_j$	Section 5.1.5

$\mathcal{NP}$-hard:

$*$	$F2, R1 \mid p_{ij} = p, t_j \mid C_{\max}$		Section 5.1.2
$*$	$F2, R1 \mid t_j = T \mid C_{\max}$		Kise [1991], Section 5.1.1
$*$	$F3, R1 \mid p_{ij} = p, r_j, t_{jk} \mid C_{\max}$		Section 5.3
$*$	$Fm, R1 \mid p_{ij} = 1, r_j, t_{jk} \mid L_{\max}$	$\forall\, m \geq 2$	Section 5.2.2
$*$	$F3, R1 \mid p_{ij} = p, t_{jk} \mid L_{\max}$		Section 5.3
$*$	$Fm, R1 \mid p_{ij} = 1, r_j, t_{jk} \mid \sum C_j$	$\forall\, m \geq 2$	Section 5.2.2
	$Fm, R1 \mid p_{ij} = 1, t_{jk} \mid \sum w_j U_j$	$\forall\, m \geq 2$	Section 5.2.2
	$Fm, R1 \mid p_{ij} = 1, t_{jk} \mid \sum T_j$	$\forall\, m \geq 2$	Section 5.2.2
$*$	$Fm, R1 \mid p_{ij} = 1, t_{jk} \mid \sum w_j T_j$	$\forall\, m \geq 2$	Section 5.2.2
$*$	$F2 \mid r_j \mid C_{\max}$		Lenstra et al. [1977]
$*$	$F3 \mid\mid C_{\max}$		Garey et al. [1976]
$*$	$F2 \mid\mid L_{\max}$		Lenstra et al. [1977]
$*$	$F2 \mid\mid \sum C_j$		Garey et al. [1976]

Table 5.1: Complexity results for flow-shop problems with a single robot

In the preceding subsections we have neglected empty moving times for the robot when it moves back to a previous machine. For the flow-shop case with $m = 2$ machines (where only empty moving times t'_{21} from machine M_2 back to M_1 occur) we have shown that these problems are equivalent to the corresponding problems without empty moving times ($t'_{21} = 0$). Thus, all algorithms for problems with $m = 2$ also solve the more general case with empty moving times. Note that for $m \geq 3$ machines such a result in general is not valid, since the routing of the robot is not fixed in advance.

Finally, we want to summarize the consequences of the obtained results to special cases of the classical 3-machine flow-shop problem $F3||C_{\max}$ where the processing times are constant for certain machines. As observed in Subsection 5.1.1, problem $F2, R1 \mid t_j \mid C_{\max}$ is equivalent to $F3||C_{\max}$ where the robot corresponds to the second machine. Thus, the $\mathcal{NP}$-hardness of problems $F2, R1 \mid p_{ij} = p, t_j \mid C_{\max}$ and $F2, R1 \mid t_j = T \mid C_{\max}$ implies that problems $F3 \mid p_{1j} = p_{3j} = p \mid C_{\max}$ and $F3 \mid p_{2j} = p_2 \mid C_{\max}$ are strongly $\mathcal{NP}$-hard. On the other hand, problem $F3 \mid p_{1j} = p_{3j} = 1 \mid C_{\max}$ is polynomially solvable since we have shown that $F2, R1 \mid p_{ij} = 1, t_j \mid C_{\max}$ is an easy problem. Furthermore, we claim that $F3 \mid p_{1j} = p_1, p_{2j} = p_2 \mid C_{\max}$ is polynomially solvable. By an exchange argument it can easily be proved that sorting the jobs according to non-increasing processing times p_{3j} leads to an optimal schedule. Symmetrically, $F3 \mid p_{2j} = p_2, p_{3j} = p_3 \mid C_{\max}$ can be solved by sorting the jobs according to non-decreasing processing times p_{1j}.

Polynomially solvable:

$F3 \mid p_{1j} = p_{3j} = 1 \mid C_{\max}$	Section 5.1.4
$F3 \mid p_{1j} = p_1, p_{2j} = p_2 \mid C_{\max}$	Section 5.3
$F3 \mid p_{2j} = p_2, p_{3j} = p_3 \mid C_{\max}$	Section 5.3

$\mathcal{NP}$-hard:

*	$F3 \mid p_{2j} = p_2 \mid C_{\max}$	Kise [1991], Section 5.1.1
*	$F3 \mid p_{1j} = p_{3j} = p \mid C_{\max}$	Section 5.1.2

Table 5.2: Complexity results for flow-shop problems with constant processing times

6 Immediate selection for problem $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$

Efficient immediate selection (constraint propagation) techniques have been developed for several scheduling problems like the job-shop problem (Carlier & Pinson [1989], Brucker et al. [1994]), a single-machine problem with positive and negative time-lags (Brucker et al. [1999]) or the resource-constrained project scheduling problem (cf. Brucker et al. [1998], Baptiste & Le Pape [1996]).

In this section we describe how these techniques can be adapted to the single-machine problem $1 \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$. In this problem we are given a set $V := \{1, \dots, n\} \cup \{0, *\}$ of jobs (operations) where 0 and $*$ are dummy jobs with $p_0 = p_* = 0$. All jobs $j \in V$ have to be processed without preemptions for p_j time units on a single machine. Job j cannot be started before its release date (head) r_j and stays in the system for q_j time units (tail) after processing. Furthermore, minimal finish-start time-lags $i \xrightarrow{l_{ij}} j \in C$ with integers $l_{ij} \geq 0$ are given for some pairs of jobs. If $i \xrightarrow{l_{ij}} j \in C$ holds, job j cannot be started earlier than l_{ij} time units after the completion of job i. By these time-lags precedence relations (conjunctions) are induced which will be denoted by $i \to j \in C$. We assume that all transitive arcs $i \to j$ (these are arcs where a job h with $i \to h \to j$ exists) are contained in the set C and that the time-lags are transitively adjusted, i.e. $l_{ih} + p_h + l_{hj} \leq l_{ij}$ holds. Furthermore, we regard the heads and tails as special time-lags by setting $l_{0j} := r_j$ and $l_{j*} := q_j$ for all jobs $j = 1, \dots, n$.

Since on a single machine two jobs cannot be processed simultaneously, all pairs of jobs which are not linked by a conjunction $i \to j$ or $j \to i$ are contained in the set D of disjunctions. These disjunctions $i - j \in D$ are weighted with sequence-dependent setup times s_{ij} with the following meaning: If j is directly processed after job i, a setup s_{ij} between the completion time of i and the starting time of j has to be considered. If, on the other hand, i is processed directly after j, a setup s_{ji} occurs. We assume that the setup times satisfy the weak triangle inequality $s_{ih} + p_h + s_{hj} \geq s_{ij}$ for all disjunctions. Furthermore, minimal time-lags and setups are supposed to satisfy the triangle inequalities

$$l_{ih} + p_h + s_{hj} \geq s_{ij} \text{ and } s_{ih} + p_h + l_{hj} \geq s_{ij}. \tag{6.1}$$

Since the time-lags are transitively adjusted, the inequality $l_{ih} + p_h + l_{hj} \geq s_{ij}$ is not supposed (because then also $i \xrightarrow{l_{ij}} j \in C$ holds).

In order to have a unique notation for the time which has to be considered after the completion time of job i, if job j is planned directly after i, we define

$$t_{ij} := \begin{cases} l_{ij}, & \text{if } i \to j \in C \\ s_{ij}, & \text{if } i - j \in D \end{cases} \tag{6.2}$$

for all jobs $i, j \in V$ where i may be processed before j, i.e. for which the relation $j \to i$ is not contained in C.

The problem is to determine feasible completion times C_j for all jobs $j \in V$ which respect the release dates, the minimal time-lags and the setup times, and minimize the objective $\max_{j=1}^{n} \{C_j + q_j\} = C_*$. As already described in Subsection 4.2, this problem arises for a robot in the job-shop problem with transportation times and dedicated robots for a fixed machine selection.

Immediate selection methods may be considered as preprocessing techniques which are used to reduce the search space and direct an algorithm towards good solutions. They deduce (select) additional restrictions which have to be fulfilled in every solution improving the best objective value found so far and thus, exclude irrelevant solutions from the search space. The larger the number of additional restrictions, the smaller the set of solutions which remains to be inspected by an optimization algorithm.
Additionally, immediate selection techniques may be used to prove optimality of a solution with objective value UB. If the methods detect that no feasible solution with objective value smaller than or equal to $UB - 1$ exists, we know that the value UB is optimal (supposing that all data are integral). This idea can also be used for lower bound calculations, which will be considered in more detail in Subsection 7.1. If a given hypothetical upper bound T for the objective value implies a contradiction, $T + 1$ is a valid lower bound value.

In connection with the single-machine problem $1 \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$ we use immediate selection techniques to

- derive additional precedence constraints (conjunctions C), and
- improve heads r_j, tails q_j and minimal time-lags l_{ij}.

In Subsection 6.1 we present techniques which are based on immediate selection concepts used for the job-shop problem (cf. Brucker et al. [1994]) and for a single-machine problem with positive and negative time-lags (Brucker et al. [1999]). In Subsection 6.2 we describe methods based on an adapted dynamic programming algorithm for the traveling salesman problem with time windows (Dumas et al. [1995]). Combining all these techniques leads to several immediate selection procedures, which are presented in Subsection 6.3.

6.1 Deriving additional conjunctions

Given an instance of problem $1 \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$ and a threshold value T, we define a **distance matrix** $d = (d_{ij})_{i,j \in V}$ by

$$d_{ij} = \begin{cases} 0, & \text{if } i = j \\ p_i + l_{ij}, & \text{if } i \to j \in C \\ -T, & \text{otherwise.} \end{cases} \tag{6.3}$$

For each pair of jobs $i, j \in V$ the value d_{ij} is a lower bound for the difference $S_j - S_i$ of their starting times (minimal start-start time-lag). If $i \to j \in C$ holds, we have to satisfy $S_j - S_i \geq p_i + l_{ij}$, which is ensured by the second condition in (6.3). Furthermore, in each feasible schedule with $S_0 = 0$ and $S_* \leq T$ we must have $S_i - S_j \leq S_* - S_0 \leq T$, which implies $S_j - S_i \geq -T$ (cf. the third condition in (6.3)). Heads and tails are regarded as special time-lags $l_{0j} = r_j, l_{i*} = q_i$ and are incorporated into d by the second condition in (6.3).

Note that the relation $S_j - S_i \geq d_{ij}$ has the following transitivity property:

$$\text{if } S_j - S_i \geq d_{ij} \text{ and } S_k - S_j \geq d_{jk}, \text{ then } S_k - S_i \geq d_{ij} + d_{jk}. \tag{6.4}$$

Thus, the entry d_{ij} in the distance matrix may be replaced by $\max\{d_{ij}, d_{ik} + d_{kj}\}$ for all triples $i, j, k \in V$. All these transitive adjustments can be done in $O(n^3)$ by the Floyd-Warshall algorithm (cf. e.g. Papadimitriou & Steiglitz [1982]). If, after applying this algorithm, $d_{ii} > 0$ holds for some job $i \in V$, a positive cycle is indicated and no feasible schedule for the threshold T exists.

Additional conjunctions may be derived from the adjusted distance matrix by testing the conditions of the following three lemmata. The proofs of the lemmata are omitted since they are simple adaptions of the original proofs (cf. Brucker et al. [1999] and Brucker et al. [1994]).

Lemma 6.1: (distance update) If $d_{ij} > -p_j$ holds for a disjunction $i - j \in D$, then we may introduce the conjunction $i \to j \in C$ and set $d_{ij} := \max\{d_{ij}, p_i + s_{ij}\}$. □

For the job-shop problem Brucker et al. [1994] derived conditions which imply that a job has to precede another job by considering sets of jobs with cardinality r (called r-sets). Taking into account the setup times s_{ij}, the r-set conditions for $r = 2$ and $r = 3$ can be extended as follows:

Lemma 6.2: (direct arcs) If $d_{0j} + p_j + s_{ji} + d_{i*} > T$ holds for a disjunction $i - j \in D$, then the conjunction $i \to j \in C$ may be fixed. □

Lemma 6.3: (triple arcs) If the three conditions

$$\begin{aligned} d_{0j} + p_j + s_{ji} + p_i + s_{ik} + d_{k*} &> T \\ d_{0j} + p_j + s_{jk} + p_k + s_{ki} + d_{i*} &> T \\ d_{0k} + p_k + s_{kj} + p_j + s_{ji} + d_{i*} &> T \end{aligned}$$

hold for a disjunction $i - j \in D$ and another job $k \in V$, then the conjunction $i \to j \in C$ may be fixed. □

Fixing all direct arcs and all triple arcs can be done in $O(n^2)$ and $O(n^3)$, respectively. For the job-shop problem Carlier & Pinson [1989] developed an efficient algorithm to fix more conjunctions based on considerations of cliques. These ideas have been generalized by Brucker & Thiele [1996] to the general-shop problem with setup times which satisfy the strong triangle inequality $s_{ij} \leq s_{ih} + s_{hj}$. Unfortunately, we could not extend this concept efficiently to the case in which the setups satisfy only the weak triangle inequality. In this case immediate selection procedures which are based on capacity considerations for intervals (cf. e.g. Brucker et al. [1994], Dorndorf et al. [1998]) can also not be generalized in an efficient way. This is due to the fact that processing and setup times cannot be considered separately. While in the case of the strong inequality bounds for the processing times and for the setup times may be calculated independently and added afterwards, this additivity property is not valid in the case of the weak triangle inequality. Therefore, in the next subsection we use ideas developed for the traveling salesman problem with time windows where the impact of processing times, setup times and time-lags is considered simultaneously.

If not stated otherwise, in the following we always assume that the distance matrix d is transitively adjusted and all transitive conjunctions are contained in the set C. On the other hand, for some procedures it is sufficient (and less time-consuming) to consider only time-lags which are not implied by transitivity, i.e. all time-lags $i \xrightarrow{l_{ij}} j \in C$ where no job h with $l_{ij} = l_{ih} + p_h + l_{hj}$ exists.

Note that if a disjunction $i - j \in D$ is replaced by a conjunction $i \xrightarrow{l_{ij}} j \in C$, according to Lemma 6.1, the time-lag l_{ij} is set to $l_{ij} := s_{ij}$. Thus, the triangle inequalities (6.1) remain valid after applying immediate selection.

6.2 Infeasibility tests and improving heads and tails

In this subsection we describe some immediate selection methods which may be used to detect infeasibility or to improve heads and tails for an instance of problem $1 \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$. We will solve a strong relaxation of the single-machine problem by adapting a dynamic programming algorithm of Dumas et al. [1995] for the asymmetric traveling salesman problem with time windows (ATSP-TW).

Given a threshold value T, we consider a relaxed version of the feasibility problem $1 \mid prec(l_{ij}), r_j, d_j, s_{ij} \mid -$ with deadlines $d_j := T - q_j$. By showing that no feasible solution for the relaxation exists, we may conclude that no feasible schedule for the original problem with threshold T exists. In the considered relaxation we require that a schedule respects

- the time windows $[r_j, d_j]$ of all jobs $j \in V$,
- all precedence relations $i \to j \in C$,
- all setup times s_{ij}, and

- all minimal time-lags l_{ij} between jobs i, j which are planned consecutively in the schedule.

Thus, the only constraints which are relaxed are the time-lags between jobs which are not planned consecutively. A feasible schedule for the relaxation is only feasible for the original problem if the triangle inequality $t_{ij} \leq t_{ih} + p_h + t_{hj}$ holds for all jobs i, j which are planned consecutively. Since the inequalities $l_{ij} > l_{ih} + p_h + l_{hj}$, $l_{ij} > s_{ih} + p_h + s_{hj}$, $l_{ij} > l_{ih} + p_h + s_{hj}$ or $l_{ij} > s_{ih} + p_h + l_{hj}$ for some $i, j, h \in V$ may hold, in general this triangle inequality is not valid.

To solve the relaxation we adapt a dynamic programming algorithm of Dumas et al. [1995]. We consider a subset $V^0 \subseteq V \setminus \{0\}$. A subset $S \subseteq V^0$ is called feasible if for each job $j \in S$ all predecessors $i \in V^0$ with $i \to j \in C$ are also contained in S. For a feasible set $S \subseteq V^0$ and a job $j \in S$ let $F(S, j)$ be the minimal starting time for job j when all jobs in the set $S \setminus \{j\}$ are processed before. Two possibilities exist for a feasible schedule realizing $F(S, j)$:

- either job j starts at its release date r_j, or
- job j starts at time $S_i + p_i + t_{ij}$ directly after another job i.

In the first case we have $F(S, j) = r_j$, in the second we have to select the minimal value $F(S \setminus \{j\}, i) + p_i + t_{ij}$ among all jobs $i \in S \setminus \{j\}$ which may be processed last in the set $S \setminus \{j\}$. If this minimal value is larger than the deadline $d_j - p_j$ of job j, the corresponding schedule with job j as a last job is infeasible w.r.t. the time windows, i.e. in this case we may set $F(S, j) := \infty$.

Thus, the values $F(S, j)$ can be calculated recursively as follows.

Initialization:

$$F(\{j\}, j) := \begin{cases} r_j, & \text{if } j \text{ has no predecessor } i \text{ with } i \to j \in C \\ \infty, & \text{otherwise} \end{cases}$$

for all $j \in V^0$.

Recursion:

$$F(S \cup \{j\}, j) = \begin{cases} \max[r_j, \min\limits_{i \in S}\{F(S, i) + p_i + t_{ij}\}] & \text{, if } \min\limits_{i \in S}\{F(S, i) + p_i + t_{ij}\} \leq d_j - p_j \\ \infty & \text{, otherwise} \end{cases}$$

for all $S \subseteq V^0$ and all $j \notin S$ whose predecessors are all contained in S.

For all other j the set $S \cup \{j\}$ is not feasible and we may also assume $F(S \cup \{j\}, j) := \infty$. The optimal solution value is given by $F(V^0, *)$. If $F(V^0, *) = \infty$, no feasible schedule for the set V^0 exists.

Contrary to the concepts of the previous subsection, the worst-case complexity of the dynamic programming algorithm is not polynomial. We have to calculate $F(S,j)$ for all feasible subsets $S \subseteq V^0$ and all jobs $j \in S$. Since each computation can be done in $O(n)$ time, the running time of the whole dynamic programming procedure is $O(n^2 \cdot 2^n)$. The storage requirement to store all $F(S,j)$-values is $O(n \cdot 2^n)$, which can be reduced to $O(n \cdot \binom{n}{\lceil n/2 \rceil})$ as follows. In the recursion the feasible sets S can be generated and evaluated according to non-increasing cardinalities. To calculate the objective value for a subset S with cardinality $k \in \{2,\ldots,n\}$, only subsets $S \setminus \{j\}$ of cardinality $k-1$ are needed. Thus, in iteration k we only have to store the feasible subsets of cardinalities $k-1$ and k. When we proceed with subsets of cardinality $k+1$, all subsets with cardinality $k-1$ can be removed. Since at most $\binom{n}{k}$ subsets of cardinality k exist, the maximal storage requirement is given by $\max\limits_{k=1}^{n} \{\binom{n}{k} + \binom{n}{k-1}\} \leq 2\binom{n}{\lceil n/2 \rceil}$ because the maximum is achieved for $k = \lceil \frac{n}{2} \rceil$. Thus, the storage requirement is bounded by $O(n \cdot \binom{n}{\lceil n/2 \rceil})$. Further details concerning the implementation of the dynamic programming procedure can be found in Subsection 10.2.

Although the worst-case complexity of the proposed algorithm is exponential, the practical complexity is much smaller. This is due to the fact that many subsets $S \subseteq V^0$ are infeasible w.r.t. the precedence relations, i.e. in practice much fewer than 2^n subsets have to be considered.

If we apply the dynamic programming algorithm to the whole set V, the optimal value $F(V,*)$ provides a lower bound for the optimal objective value of the single-machine problem. In particular, if no feasible solution for the relaxation exists, no feasible solution for the single-machine problem with threshold T exists. Since the dynamic programming procedure may be very time-consuming for larger n, in the following we describe some local considerations in which we apply the algorithm only to subsets V^0 of all jobs in V. One possibility is to consider an interval $[a,b]$ and all jobs which may be (at least partially) processed in this interval. This means we apply the algorithm to the set $V^{[a,b]} := \{j \in V \mid [r_j, d_j] \cap [a,b] \neq \emptyset\}$. If no feasible solution for this subset exists, we may conclude that no feasible solution exists for the whole set.

Another application of the dynamic programming procedure may improve heads and tails of the jobs as follows. For each job j we consider the set $S^j := \{i \mid i \rightarrow j \in C\} \cup \{j\}$ consisting of all predecessors of job j (including the transitive predecessors) and j itself. Since the value $F(S,j)$ defines the minimal starting time for job j when all jobs in the set $S \setminus \{j\}$ are processed before, the value $F(S^j,j)$ is a lower bound for the starting time of job j in each feasible schedule. Thus, if $F(S^j,j) > r_j$ holds, we may increase the head of j to the corresponding value. Note that during the calculation of the $F(S^j,j)$-values, the values $F(S^i,i)$ for all jobs $i \in S^j$ are also calculated. Hence, the algorithm does not have to be called individually for each job and heads of various jobs may be improved simultaneously. Symmetrically, by interchanging the role of heads and tails, tails q_j may be increased during a backward calculation for sets S^j consisting of all successors of job j and j itself.

6.3 Applications of the immediate selection procedures

In this subsection we describe how the concepts of the previous subsections may be integrated into several (more or less time-consuming) immediate selection procedures. The first procedure `Immediate_Selection1(T)` executes the following steps for a given threshold value T:

(1) Define the distance matrix d according to (6.3).

(2) Calculate the transitive closure of d taking into account all distance updates according to Lemma 6.1. If a positive cycle is detected, infeasibility is stated.

(3) Fix all direct arcs according to Lemma 6.2 and update d as in Step (2) until no further direct arcs can be found.

(4) Fix all triple arcs according to Lemma 6.3 and update d as in Step (2) until no further triple arcs can be found.

This procedure may be extended to a procedure `Immediate_Selection2(T)` which repeats Steps (3) and (4) until no further direct or triple arcs can be found. Since at most $O(n^2)$ arcs may be fixed and updating the distance matrix can be done in $O(n^3)$, both procedures are polynomial and their worst-case complexity is $O(n^5)$.

In order to improve heads and tails with the dynamic programming algorithm of Subsection 6.2 we derived the procedure `Improve_heads_tails1(T)`:

(1) For each job $j \in V$ determine the sets $S^j := \{i \mid i \rightarrow j \in C\} \cup \{j\}$ consisting of all (including the transitive) predecessors of job j and j itself.

(2) Consider the jobs according to a list in which the jobs are sorted w.r.t. non-increasing cardinalities of the sets S^j. For each job j in the list calculate $F(S^j, j)$ with the dynamic programming algorithm. If no feasible solution for subset S^j exists, infeasibility is detected. Otherwise, during the calculation of the value $F(S^j, j)$, all values $F(S^i, i)$ for predecessors i of j have also been calculated. Set $r_i := F(S^i, i)$ for all $i \in S^j$ and eliminate all considered jobs i from the list. If after considering all jobs some heads have been improved, transform these heads back into the distance matrix d and update d as in Step (2) from procedure `Immediate_Selection1(T)`.

(3) Symmetrically, improve tails q_j by considering the jobs according to non-increasing cardinalities of their successor sets $S^j := \{i \mid j \rightarrow i \in C\} \cup \{j\}$. If after considering all jobs some tails have been improved, transform these tails back into the distance matrix d and update d as in Step (2) from procedure `Immediate_Selection1(T)`.

Again, this procedure may be extended to a procedure `Improve_heads_tails2(T)` which repeats Steps (2) and (3) until no further heads or tails can be improved. In a further extension `Improve_heads_tails3(T)` after transforming back the improved heads and tails in Steps (2) and (3), the procedure `Immediate_Selection2(T)` is called. Again, Steps (2) and (3) are repeated until no further improvements can be derived. Note that these three procedures are not polynomial, since the dynamic programming algorithm has worst-case complexity $O(n^2 \cdot 2^n)$.

7 The calculation of lower bounds for problem $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$

In this section we describe the computation of lower bounds for the single-machine problem $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$.

An often used method for getting lower bounds is to solve relaxations of the problem: some restrictions which have to be fulfilled in every feasible solution are dropped. Obviously, the optimal solution value of the resulting problem is a lower bound for the optimal solution value of the original problem. Unfortunately, relaxing one of the constraints in problem $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$ does not make the problem much easier, since problems $1 \mid r_j \mid \max\{C_j + q_j\}$, $1 \mid chains\,(l_{ij}) \mid C_{\max}$, and $1 \mid s_{ij} \mid C_{\max}$ are strongly $\mathcal{NP}$-hard, as previously mentioned. Hence, the only way to obtain polynomially solvable relaxations is to drop more than one constraint. If we relax the setup times s_{ij} and the minimal time-lags l_{ij}, and allow preemption, the resulting problem $1 \mid pmtn, prec, r_j, \mid \max\{C_j + q_j\}$ is polynomially solvable (Jackson [1955]). Additionally, the minimal time-lags can be partially taken into account by modifying heads and tails of all operations according to the formulae

$$r_j := \max\{r_j, \max_{i \to j \in C}\{r_i + p_i + l_{ij}\}\}, \tag{7.1}$$

$$q_i := \max\{q_i, \max_{i \to j \in C}\{l_{ij} + p_j + q_j\}\}. \tag{7.2}$$

But since in this relaxation many constraints have been dropped, the resulting lower bounds are quite bad. Another possibility to obtain lower bounds is based on linear programming relaxations. In this approach the optimization problem is at first formulated as an integer linear program. Then the integer constraints are relaxed leading to a linear program which provides a lower bound for the optimal value of the integer program.

In this section we will propose two lower bound calculations for the considered single-machine problem. In Subsection 7.1 we describe how lower bounds can be calculated with immediate selection techniques. Approaches based on linear programming are presented in Subsections 7.2-7.4.

7.1 Lower bounds based on immediate selection

In this subsection we describe how the immediate selection techniques from the previous section can be used to calculate lower bounds in a so-called destructive approach. Contrary to constructive approaches which directly produce bounds by constructing solutions to relaxations, destructive approaches use another technique which is also called **destructive improvement** (cf. Klein & Scholl [1999]). To derive lower bounds with this technique we consider the decision version (feasibility problem) of the single-machine problem: Given a threshold value T, does a feasible schedule exist with an objective value smaller than or equal to T ? If we can prove that such a feasible schedule does not exist, then $T+1$ is a valid lower bound for the single-machine problem supposing that all data are integral. To contradict (destruct) a threshold value T again relaxations may be used. If we can state infeasibility for a relaxed problem, obviously the original problem is also infeasible. To find the best lower bound we search for the largest T, where infeasibility can be proved. This is done in an efficient way by binary search. In each step we consider an interval $[L, U]$ in which we search for a valid lower bound value. Initially, we start with a valid upper bound value U^0 and a valid lower bound value L^0. In each iteration we solve the feasibility problem for $T = \lfloor \frac{U+L}{2} \rfloor$. If we can prove that no feasible solution with an objective value smaller than or equal to T exists, we increase the lower bound L to the value $T+1$ and repeat the procedure with the interval $[T+1, U]$. Otherwise, if we do not succeed in proving infeasibility for the threshold T, we replace U by T and repeat the procedure in the interval $[L, T]$. The binary search procedure terminates as soon as $L \geq U$ holds after at most $O(\log(U^0 - L^0))$ steps. Then L equals the largest lower bound value which can be calculated in this way.

One possibility to detect infeasibility is to use immediate selection techniques as described in the previous section. Given a threshold value T for the single-machine problem $1 \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$ we define deadlines $d_j := T - q_j$ and consider the corresponding instance of the feasibility problem $1 \mid prec(l_{ij}), r_j, d_j, s_{ij} \mid -$. If one of the immediate selection methods proves that no feasible schedule for this instance exists, we know that $T+1$ is a valid lower bound value. Computational results for the procedures presented in Subsection 6.3 can be found in Section 10.2. Another method to detect infeasibility will be described in the next subsection.

7.2 A chain-packing bound

In this subsection we describe a destructive linear programming lower bound for the feasibility problem $1 \mid prec(l_{ij}), r_j, d_j, s_{ij} \mid -$ where the precedence constraints are decomposed into chains. This lower bound is based on a time-oriented approach and extends the job-packing bound used by Martin [1996] in connection with the classical job-shop problem.

We assume that a decomposition of the precedence constraints C into γ chains of the form $\Gamma_\varrho : \varrho_1 \rightarrow \varrho_2 \rightarrow \ldots \rightarrow \varrho_{n_\varrho}$ $(\varrho = 1, \ldots, \gamma)$ with chain lengths n_ϱ is given. Furthermore, we have some precedence constraints between jobs belonging to different chains. Obviously, each feasible single-machine schedule decomposes into corresponding "chain-schedules" for all jobs belonging to the same chain. These schedules will be the basic elements of our approach. A schedule for chain Γ_ϱ (described by the completion times of the jobs) will be called a **feasible chain-schedule** if it

- respects the time windows $[r_j, d_j]$ of all jobs j belonging to chain Γ_ϱ,
- respects all precedences $\varrho_1 \rightarrow \varrho_2 \rightarrow \ldots \rightarrow \varrho_{n_\varrho}$ in chain Γ_ϱ, and

- respects the minimal time-lags $l_{\varrho_k,\varrho_{k+1}}$ between consecutive jobs ϱ_k, ϱ_{k+1} belonging to chain Γ_ϱ.

Note that according to this definition time-lags l_{ij} between non-consecutive jobs i, j may be violated in a feasible chain-schedule. The reason for this weaker definition of feasibility will be explained later.

We denote by $\mathcal{S}_\varrho$ the set of all feasible chain-schedules σ for chain Γ_ϱ and define for each schedule $\sigma \in \mathcal{S}_\varrho$ binary coefficients

$$a_{j\sigma t} = \begin{cases} 1, & \text{if job } j \text{ finishes at time } t \text{ in } \sigma \\ 0, & \text{otherwise} \end{cases}$$

$$b_{\sigma t} = \begin{cases} 1, & \text{if a job is processed in } \sigma \text{ in the interval } [t-1, t] \\ 0, & \text{otherwise} \end{cases}$$

for all jobs j belonging to chain Γ_ϱ and $t = 1, \ldots, T$.

Furthermore, we introduce binary variables for every chain-schedule $\sigma \in \mathcal{S}_\varrho$ $(\varrho = 1, \ldots, \gamma)$

$$x_\sigma = \begin{cases} 1, & \text{if chain } \Gamma_\varrho \text{ is scheduled according to schedule } \sigma \in \mathcal{S}_\varrho \\ 0, & \text{otherwise.} \end{cases}$$

For each job $j \in V$ let $\Gamma(j)$ be the chain to which j belongs. Then the feasibility problem for the single machine with threshold T may be formulated as a 0-1-program as follows.

$$\sum_{\sigma \in \mathcal{S}_\varrho} x_\sigma = 1 \qquad \varrho = 1, \ldots, \gamma \tag{7.3}$$

$$\sum_{\varrho=1}^{\gamma} \sum_{\sigma \in \mathcal{S}_\varrho} b_{\sigma t} x_\sigma \leq 1 \qquad t = 1, \ldots, T \tag{7.4}$$

$$\sum_{\sigma \in \mathcal{S}_{\Gamma(j)}} \sum_{\tau=r_j+p_j}^{d_j} \tau\, a_{j\sigma\tau} x_\sigma - \sum_{\sigma \in \mathcal{S}_{\Gamma(i)}} \sum_{t=r_i+p_i}^{d_i} t\, a_{i\sigma t} x_\sigma \geq l_{ij} + p_j \qquad \forall\, i \to j \in \tilde{C} \tag{7.5}$$

$$\sum_{\sigma \in \mathcal{S}_{\Gamma(i)}} a_{i\sigma t} x_\sigma + \sum_{\sigma \in \mathcal{S}_{\Gamma(j)}} \sum_{\tau \in I_t^{i,j}} a_{j\sigma\tau} x_\sigma \leq 1 \qquad \begin{array}{l} \forall\, i - j \in D; \\ t = r_i + p_i, \ldots, d_i \end{array} \tag{7.6}$$

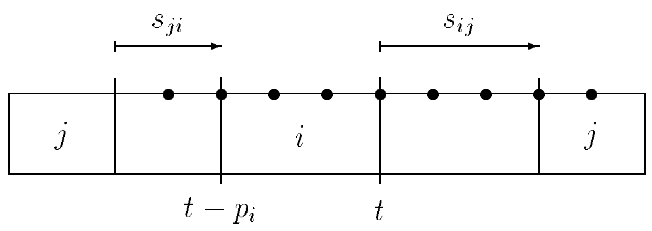
$$x_\sigma \in \{0, 1\} \qquad \forall\, \sigma \in \bigcup_{\varrho=1}^{\gamma} \mathcal{S}_\varrho \tag{7.7}$$

Due to the equalities (7.3) exactly one schedule $\sigma \in \mathcal{S}_\varrho$ is chosen for each chain Γ_ϱ $(\varrho = 1, \ldots, \gamma)$. The capacity conditions (7.4) ensure that in every time period $[t-1, t]$ at most one job is scheduled. Constraints (7.5) take care of the minimal time-lags by requiring $C_j - C_i \geq l_{ij} + p_j$ for all $i \xrightarrow{l_{ij}} j \in C$. Since the minimal time-lags between consecutive jobs of the same chain are already satisfied by definition in a feasible chain-schedule $\sigma \in \mathcal{S}_\varrho$, in (7.5) it is sufficient to consider only constraints corresponding to the reduced set

$$\tilde{C} := \{i \to j \in C \mid i, j \text{ are not consecutive jobs in a chain}\}.$$

The weighted disjunctions $i - j \in D$ are taken into account by conditions (7.6) as follows. Assuming that job i finishes at time $t \in [r_i + p_i, d_i]$, we have two possibilities for scheduling job j (cf. the figure below):

- If j is scheduled after i, the earliest completion time of j is $C_j = t + s_{ij} + p_j$,
- if j is scheduled before i, the latest completion time of j is $C_j = t - p_i - s_{ji}$.



Thus, job j cannot be finished in the interval

$$I_t^{i,j} := [t - p_i - s_{ji} + 1, t + s_{ij} + p_j - 1] \cap [r_j + p_j, d_j].$$

If we have $a_{i\sigma t} = 1$, which means that i finishes at time t in σ, all possible completion times of job j in the interval $I_t^{i,j}$ (as indicated by the black cycles in the figure) are blocked by constraints (7.6).

In order to reduce the number of constraints some superfluous ones may be omitted:

- The constraints in (7.4) only have to be considered for intervals $[t-1, t]$ in which at least two jobs may be processed, i.e. for all t with $|\{j \mid [r_j, d_j] \cap [t-1, t] \neq \emptyset\}| \geq 2$. Since in all other time periods at most one job may be processed, the corresponding capacity constraints are automatically fulfilled for each collection of feasible chain-schedules.
- Besides the constraints corresponding to consecutive jobs in a chain all constraints in (7.5) can be dropped which correspond to time-lags induced by transitivity (i.e. all $i \xrightarrow{l_{ij}} j \in \tilde{C}$ where a job h exists with $l_{ij} = l_{ih} + p_h + l_{hj}$).
- In the disjunctive constraints (7.6), all constraints for pairs $(i-j, t)$ may be omitted for which the interval $I_t^{i,j}$ is empty. This case occurs if no time periods which are blocked by job i finishing at time t are contained in the time window $[r_j + p_j, d_j]$ of job j.

For the lower bound calculation we consider the fractional relaxation of (7.3) to (7.7). Since each variable x_σ is already bounded by 1 due to (7.3), it is sufficient to replace the 0-1-constraints (7.7) by the condition $x_\sigma \geq 0 \quad \forall\, \sigma \in \bigcup_{\varrho=1}^{\gamma} \mathcal{S}_\varrho$.

If this relaxation has no feasible fractional solution, then $T+1$ is a valid lower bound for the single-machine problem. The largest bound obtained by this relaxation will be called the **chain-packing bound**. Later in subsection 7.4 we will show that solving the fractional relaxation is still an $\mathcal{NP}$-hard problem.

Note that if we decompose the precedence constraints into single nodes instead of chains and cover all conjunctions by conditions (7.5), the packing formulation (7.3) to (7.7) is equivalent to a formulation based on so-called **time-indexed variables** (cf. Queyranne & Schulz [1994]). These binary variables are defined for all jobs $j \in V$ and all time periods $t = 1, \ldots, T$ by

$$x_{jt} = \begin{cases} 1, & \text{if job } j \text{ finishes at time } t \\ 0, & \text{otherwise.} \end{cases}$$

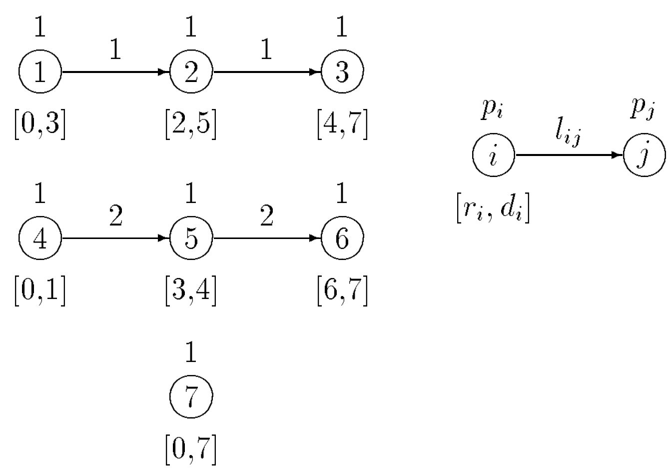
It can be shown that the chain-packing bound is at least as strong as the bound based on these time-indexed variables, since each feasible fractional chain-packing schedule induces a feasible fractional time-indexed solution

$$x_{jt} := \sum_{\sigma \in \mathcal{S}_{\Gamma(j)}} a_{j\sigma t} x_\sigma,$$

where x_{jt} is the sum of portions of job j which finish at time t in the chain-packing schedule.

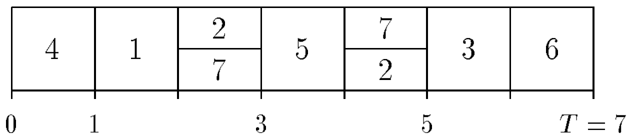
Thus, the lower bound based on the time-indexed variables cannot be larger than the chain-packing bound. In the following example it can be seen that the chain-packing bound may produce better bounds than the time-indexed relaxation.

Example 7.1: We are given 7 jobs with unit processing times and the following precedences, time-lags and time windows:



We consider the feasibility problem with threshold value $T = 7$. In each feasible chain-packing schedule (fractional or non-fractional) the chain consisting of jobs 4,5 and 6 has to be planned non-fractionally in the time windows [0,1], [3,4] and [6,7]. Obviously, no feasible chain-schedule exists for the chain $1 \to 2 \to 3$ which is compatible with the other chain, i.e. the feasibility problem does not have a solution. On the other hand, consider the fractional time-indexed solution

$$x_{41} = x_{54} = x_{67} = 1, x_{12} = x_{36} = 1, x_{23} = x_{25} = x_{73} = x_{75} = 0.5.$$



It can easily be checked that the two conditions for the conjunctions $1 \to 2$ and $2 \to 3$ in (7.5) are satisified by this solution. For example, for the conjunction $1 \to 2$ we have

$$\sum_{t=r_2+p_2}^{d_2} t\, x_{2t} - \sum_{t=r_1+p_1}^{d_1} t\, x_{1t} = 3x_{23} + 5x_{25} - 2x_{12} = 2 \geq l_{12} + p_2 = 2,$$

the condition for $2 \to 3$ can be verified analogously. Hence this solution is feasible for the fractional time-indexed relaxation. □

In the following we describe how the chain-packing bound can be calculated in an efficient way. By introducing two additional variables, λ_1, λ_2, the fractional relaxation of the feasibility problem (7.3) to (7.7) can be formulated as a linear optimization problem:

$$\text{MIN } \lambda_1 + \lambda_2 \tag{7.8}$$

s.t.

$$\sum_{\sigma \in \mathcal{S}_\varrho} x_\sigma = 1 \qquad \varrho = 1, \ldots, \gamma \tag{7.9}$$

$$\lambda_1 - \sum_{\varrho=1}^{\gamma} \sum_{\sigma \in \mathcal{S}_\varrho} b_{\sigma t} x_\sigma \geq -1 \qquad t = 1, \ldots, T \tag{7.10}$$

$$\sum_{\sigma \in \mathcal{S}_{\Gamma(j)}} \sum_{\tau=r_j+p_j}^{d_j} \tau\, a_{j\sigma\tau} x_\sigma - \sum_{\sigma \in \mathcal{S}_{\Gamma(i)}} \sum_{t=r_i+p_i}^{d_i} t\, a_{i\sigma t} x_\sigma \geq l_{ij} + p_j \qquad \forall\, i \to j \in \tilde{C} \tag{7.11}$$

$$\lambda_2 - \sum_{\sigma \in \mathcal{S}_{\Gamma(i)}} a_{i\sigma t} x_\sigma - \sum_{\sigma \in \mathcal{S}_{\Gamma(j)}} \sum_{\tau \in I_t^{i,j}} a_{j\sigma\tau} x_\sigma \geq -1 \qquad \begin{array}{c} \forall\, i - j \in D; \\ t = r_i + p_i, \ldots, d_i \end{array} \tag{7.12}$$

$$x_\sigma \geq 0 \qquad \forall\, \sigma \in \bigcup_{\varrho=1}^{\gamma} \mathcal{S}_\varrho \tag{7.13}$$

$$\lambda_1, \lambda_2 \geq 0. \tag{7.14}$$

In this formulation the capacity of the machine and the weighted disjunctions may be violated, which is compensated by the two "penalty variables", λ_1 and λ_2. Note that we do not need an additional variable λ_3 in constraints (7.11), since it is easy to determine solutions for the subproblem described by conditions (7.9) and (7.11). In this subproblem we only have to calculate feasible chain-schedules respecting all minimal time-lags l_{ij}, which can be done as follows. For each job j the length $\ell(j)$ of a longest path from 0 to j in the precedence graph is determined taking into account all processing times and minimal time-lags of its predecessors (cf. the construction of a feasible schedule defined by a complete selection in the disjunctive graph model in Section 3.1). By scheduling all jobs as early as possible at time $\ell(j)$ we get a schedule in which all time-lags are respected, but the machine capacity and some disjunctions may be violated. If we decompose this schedule into a collection of γ chain-schedules, we obtain a solution which is feasible according to conditions (7.11).

This means that the vector x in a feasible solution $(x, \lambda_1, \lambda_2)$ for (7.9) to (7.14) only has to fulfill all minimal time-lags l_{ij} according to (7.5); conditions (7.4) and (7.6) may be violated. Then the variable λ_1 in (7.10) denotes the maximal violation of the load of the machine, the variable λ_2 in (7.12) denotes the maximal violation of a weighted disjunction. A feasible solution for the fractional relaxation of (7.3) to (7.7) exists if and only if the linear program (7.8) to (7.14) has an optimal solution value with $\lambda_1^* = 0$ and $\lambda_2^* = 0$, i.e. $\lambda_1^* + \lambda_2^* = 0$.

Unfortunately, the number of variables in this LP-formulation grows exponentially with the number of jobs. However, it is not necessary to generate and store all these columns. We can work with just a few at a time and generate new chain-schedules when they are really needed. Such an approach, which was suggested by Gilmore & Gomory [1961] for the cutting-stock problem, is called **delayed column generation** and will be described next.

The basic idea of column generation is not to generate and store all columns of the problem formulation in advance, but to generate them only as needed. This strategy can be carried out with the revised simplex method for linear programming (cf. Chvátal [1983]). In the revised simplex method a basis B consisting of basic variables is changed in each iteration by calculating an entering and a leaving variable. Given the current basis B a set of multipliers y (corresponding to the dual variables) is computed by $yB = c_B$, where c_B is the vector of the coefficients in the objective function belonging to the current basic columns. By this way the **reduced costs** $c_B^y := c_B - yB$ of the basic variables become zero and the multipliers are used to **price out** the non-basic columns, i.e. to compute their reduced costs $c_N^y := c_N - yB$ (where c_N denotes the vector of the coefficients in the objective function belonging to the non-basic basic columns). If any reduced cost is negative, the corresponding non-basic column may be used as an entering column improving the current objective value. After determining a leaving column, the multipliers are calculated again and the process is repeated until no further non-basic columns with negative reduced costs exist. In this case the current basic solution is primal- and dual-feasible, i.e. the solution is optimal.

In the following we describe the details of our column generation procedure. For this purpose the corresponding dual problem to the primal problem (7.8) to (7.14) is required. If we denote the dual variables corresponding to constraints (7.9) by z_ϱ, those corresponding to (7.10) by y_t, and the variables belonging to conditions (7.11) and (7.12) by $y_{i\to j}$ and $y_{(i,j),t}$, respectively, the dual problem has the form

$$\text{MAX} \sum_{\varrho=1}^{\gamma} z_\varrho - \sum_{t=1}^{T} y_t + \sum_{i\to j\in\tilde{C}} (l_{ij}+p_j) y_{i\to j} - \sum_{i-j\in D} \sum_{t=r_i+p_i}^{d_i} y_{(i,j),t} \tag{7.15}$$

s.t.

$$\sum_{t=1}^{T} y_t \leq 1 \tag{7.16}$$

$$\sum_{i-j\in D} \sum_{t=r_i+p_i}^{d_i} y_{(i,j),t} \leq 1 \tag{7.17}$$

$$\begin{aligned} z_\varrho - \sum_{t=1}^{T} b_{\sigma t} y_t + \sum_{\substack{i\in\Gamma_\varrho \\ \{j\mid j\to i\in\tilde{C}\}}} \sum_{t=r_i+p_i}^{d_i} t\, a_{i\sigma t} y_{j\to i} - \sum_{\substack{i\in\Gamma_\varrho \\ \{j\mid i\to j\in\tilde{C}\}}} \sum_{t=r_i+p_i}^{d_i} t\, a_{i\sigma t} y_{i\to j} \\ - \sum_{\substack{i\in\Gamma_\varrho \\ \{j\mid i-j\in D\}}} \sum_{t=r_i+p_i}^{d_i} a_{i\sigma t} y_{(i,j),t} - \sum_{\substack{i\in\Gamma_\varrho \\ \{j\mid j-i\in D\}}} \sum_{\tau=r_j+p_j}^{d_j} \sum_{t\in I_\tau^{j,i}} a_{i\sigma t} y_{(j,i),\tau} \leq 0 \\ \varrho=1,\ldots,\gamma;\ \forall\ \sigma\in\mathcal{S}_\varrho \end{aligned} \tag{7.18}$$

$$\begin{aligned} y_t &\geq 0 && t=1,\ldots,T && (7.19)\\ y_{i\to j} &\geq 0 && \forall\ i\to j\in\tilde{C} && (7.20)\\ y_{(i,j),t} &\geq 0 && \forall\ i-j\in D;\ t=r_i+p_i,\ldots,d_i && (7.21)\\ z_\varrho &\in \mathbb{R} && \varrho=1,\ldots,\gamma. && (7.22) \end{aligned}$$

In each iteration of the revised simplex method we work with a restricted set of columns which contains a basis, the variables λ_1, λ_2 and all slack variables according to inequalities (7.10) to (7.12). Initially, for each chain $\Gamma_\varrho\,(\varrho=1,\ldots,\gamma)$ we calculate a chain-schedule $\sigma^{(\varrho)}\in\mathcal{S}_\varrho$ which is compatible with the other chain-schedules w.r.t. the precedences and the minimal time-lags l_{ij} according to (7.11). As previously described, this can be done by longest path calculations in the precedence graph and scheduling each job as early as possible. We set $x_{\sigma^{(\varrho)}}=1$ for all these γ chain-schedules $\sigma^{(\varrho)}$ and determine the maximal violations λ_1 and λ_2 in inequalities (7.10) and (7.12). It is easy to see that we obtain a feasible basic non-fractional starting solution for (7.9) to (7.14) if we choose the γ variables $x_{\sigma(\varrho)}$, the penalty variables λ_1, λ_2 and slack variables for the remaining inequalities in (7.10) to (7.12) as basic variables.

In the general step of the procedure the linear program is solved to optimality with the current set of columns. Then the multipliers $z_\varrho, y_t, y_{i\to j}, y_{(i,j),t}$ which correspond with the current basis are calculated. If these values satisfy all constraints (7.16) to (7.22), we have a solution which is primal- and dual-feasible. Then we have finished because the current value $\lambda_1 + \lambda_2$ is optimal. In the case $\lambda_1 + \lambda_2 > 0$, we state infeasibility, otherwise the fractional relaxation has a feasible solution. If the multipliers do not satisfy all dual constraints (7.16) to (7.22), only condition (7.18) may be violated for a column which is not in the current working set. The other conditions must be fulfilled since λ_1, λ_2 and all slack variables according to (7.10) to (7.12) are kept in the working set and the corresponding linear program has been solved to optimality. Hence, the obtained solution is primal- and dual-feasible for the restricted set of columns in the working set and the dual constraints (7.16) to (7.22) are satisfied. This implies that condition (7.18) may only be violated for a column σ which is not contained in the current set of columns. Such a violated constraint (7.18) corresponds to a column σ with negative reduced costs c_σ^y which may be used as an entering column. To use a column generation approach, calculating one such violated constraint should be possible without explicitly enumerating every non-basic column. A procedure for this so-called **pricing problem** will be described next.

For each chain-schedule $\sigma \in \mathcal{S}_\varrho$ ($\varrho = 1, \ldots, \gamma$) we define the costs

$$\begin{aligned} c(\sigma) \; := \; & \sum_{t=1}^{T} b_{\sigma t} y_t - \sum_{\substack{i \in \Gamma_\varrho \\ \{j \mid j \to i \in \tilde{C}\}}} \sum_{t=r_i+p_i}^{d_i} t\, a_{i\sigma t} y_{j\to i} + \sum_{\substack{i \in \Gamma_\varrho \\ \{j \mid i \to j \in \tilde{C}\}}} \sum_{t=r_i+p_i}^{d_i} t\, a_{i\sigma t} y_{i\to j} \\ & + \sum_{\substack{i \in \Gamma_\varrho \\ \{j \mid i-j \in D\}}} \sum_{t=r_i+p_i}^{d_i} a_{i\sigma t} y_{(i,j),t} + \sum_{\substack{i \in \Gamma_\varrho \\ \{j \mid j-i \in D\}}} \sum_{\tau=r_j+p_j}^{d_j} \sum_{t \in I_\tau^{j,i}} a_{i\sigma t} y_{(j,i),\tau}. \end{aligned} \tag{7.23}$$

If the reduced costs $c_\sigma^y = c(\sigma) - z_\varrho$ are negative, then for $\sigma \in \mathcal{S}_\varrho$ the dual restriction (7.18) is violated and the corresponding primal variable x_σ is eligible to enter the basis. To find such a column $\sigma \in \mathcal{S}_\varrho$ for chain Γ_ϱ or to prove that none exists, it is sufficient to calculate a chain-schedule $\sigma^* \in \mathcal{S}_\varrho$ with minimal reduced costs, i.e. we determine a schedule $\sigma^* \in \mathcal{S}_\varrho$ with $c(\sigma^*) = \min\{c(\sigma) \mid \sigma \in \mathcal{S}_\varrho\}$. If $c(\sigma^*) < z_\varrho$ holds, we have found a column with negative reduced costs, otherwise no improving column for chain Γ_ϱ exists.

We consider a chain-schedule $\sigma \in \mathcal{S}_\varrho$ and collect all contributions of a job i of chain Γ_ϱ to the costs $c(\sigma)$ in (7.23). If job i finishes at time $t' \in [r_i + p_i, d_i]$ in σ, we have $a_{i\sigma t} = 1$ only for $t = t'$ and $b_{\sigma t} = 1$ for all $t \in [t' - p_i + 1, t']$. The contribution to the last term of (7.23) is given by the sum of all dual variables $y_{(j,i),\tau}$ for which t' is contained in the interval $I_\tau^{j,i}$. Since $t' \in I_\tau^{j,i}$ if and only if $\tau \in I_{t'}^{i,j}$, this sum is $\sum\limits_{\{j|j-i\in D\}} \sum\limits_{\tau\in I_{t'}^{i,j}} y_{(j,i),\tau}$. Summarizing all terms we get the following costs for job i finishing at time t':

$$\begin{aligned} c(i,t') &= \sum_{t=t'-p_i+1}^{t'} y_t - t' \sum_{\{j|j\to i\in\tilde{C}\}} y_{j\to i} + t' \sum_{\{j|i\to j\in\tilde{C}\}} y_{i\to j} \\ &\quad + \sum_{\{j|i-j\in D\}} y_{(i,j),t'} \quad + \sum_{\{j|j-i\in D\}} \sum_{\tau\in I_{t'}^{i,j}} y_{(j,i),\tau}. \end{aligned}$$

Hence, the costs $c(\sigma)$ for a chain-schedule σ with completion times C_i^σ are given by

$$c(\sigma) = \sum_{i\in\Gamma_\varrho} c(i, C_i^\sigma).$$

Our pricing algorithm is based on dynamic programming and uses a forward recursion. For each chain Γ_ϱ we have to calculate a chain-schedule $\sigma^* \in \mathcal{S}_\varrho$ which minimizes $c(\sigma)$. In order to simplify the notation, in the following we assume that chain Γ_ϱ has the form $1 \to 2 \to \ldots \to n_\varrho$. Since the order of all jobs i belonging to chain Γ_ϱ is fixed, the problem is to find completion times C_i^σ in the given time windows $[r_i + p_i, d_i]$ which respect the time-lags between consecutive jobs and minimize the sum objective $c(\sigma) = \sum\limits_{i\in\Gamma_\varrho} c(i, C_i^\sigma)$.

Let $F_k(t)$ be the minimal costs for all feasible chain-schedules $\sigma \in \mathcal{S}_\varrho$ consisting of the first k jobs of chain Γ_ϱ, where job k finishes not later than time t. For a chain-schedule realizing $F_k(t)$ there are two possibilities:

- either job k finishes before time t, i.e. it finishes not later than time $t-1$, or
- job k finishes exactly at time t.

In the first case the costs $F_k(t)$ are equal to $F_k(t-1)$. In the second case we have to select the best chain-schedule for the first $k-1$ jobs where $k-1$ finishes not later than time $t - p_k - l_{k-1,k}$. Then the costs are equal to $F_{k-1}(t - p_k - l_{k-1,k}) + c(k,t)$. Since job k cannot be finished before time $t < r_k + p_k$, the costs $F_k(t)$ are set to infinity for $t < r_k + p_k$. Furthermore, since job k cannot be finished later than time d_k, we have $F_k(t) = F_k(d_k)$ for all $t > d_k$. These values do not have to be explicitly stored if in the second case the minimum of the time $t - p_k - l_{k-1,k}$ with the corresponding deadline d_{k-1} is always used.

Summarizing, the $F_k(t)$-values can be calculated recursively as follows.

Initialization:

$$F_1(t) := \begin{cases} \infty & , \quad t < r_1 + p_1 \\ \min\{F_1(t-1),\ c(1,t)\} & , \quad t \in [r_1 + p_1, d_1] \end{cases}$$

Recursion:

$$F_k(t) = \begin{cases} \infty & , t < r_k + p_k \\ \min\{F_k(t-1),\ F_{k-1}(\min\{t - p_k - l_{k-1,k}, d_{k-1}\}) + c(k,t)\}, & t \in [r_k + p_k, d_k] \end{cases}$$

$$\text{for } k = 2, \dots, n_\varrho.$$

The optimal solution value $c(\sigma^*)$ is given by $F_{n_\varrho}(d_{n_\varrho})$. We have to calculate $F_k(t)$ for each of the n_ϱ jobs k in chain Γ_ϱ and for all time periods $t \in [r_k + p_k, d_k]$. Since each computation can be done in constant time, the complexity of the dynamic programming procedure is $O(n_\varrho \Delta_\varrho^{\max}) = O(n_\varrho T)$ where $\Delta_\varrho^{\max} := \max\limits_{k \in \Gamma_\varrho} \{d_k - p_k - r_k\}$ denotes the maximal length of a time window of a job belonging to chain Γ_ϱ.

Finding a chain-schedule $\sigma^* \in \mathcal{S}_\varrho$ which realizes $F_{n_\varrho}(d_{n_\varrho})$ can be done by a backward construction. We start with time $t := d_{n_\varrho}$ and $k := n_\varrho$ as the last job of chain Γ_ϱ. If $F_k(t) = F_k(t-1)$, then an optimal schedule exists where k finishes not later than time $t-1$. Then we reduce t by one and continue. Otherwise, $F_k(t) < F_k(t-1)$ and job k finishes exactly at time t in an optimal chain-schedule. Then we replace t by $\min\{t - p_k - l_{k-1,k}, d_{k-1}\}$ and continue with the predecessor $k-1$. This backward construction finds an optimal schedule in $O(d_{n_\varrho} - r_1 + n_\varrho) = O(T + n_\varrho)$ steps since in each step we either reduce the time or move to the preceding job.

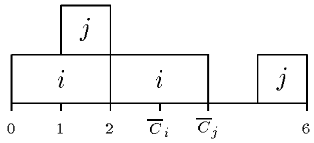
The complexity of the dynamic programming procedure is the reason why we do not require that time-lags between non-consecutive jobs also have to be satisfied in a feasible chain-schedule. If this condition had to be fulfilled, it would not be sufficient to store only the completion time t of the last scheduled job k. Then we have to store all completion times $t_1, \dots, t_{k-1}$ in order to calculate the minimal feasible starting time of job k which is given by the maximal value $t_i + l_{ik}$ of the jobs $i \in \{1, \dots, k-1\}$ with $i \to k \in C$. This extension increases the complexity of the dynamic programming procedure by the factor T which may lead to problems for larger T-values.

A nice property of the pricing problem is that it decomposes into γ problems which can be solved independently for each chain. The dynamic programming procedure can be applied successively to each chain Γ_ϱ and may produce more than one chain-schedule with negative reduced costs. Then all these columns are inserted into the working set of current columns and the linear program is resolved to optimality. If the size of the working set exceeds a certain value, some non-basic columns are deleted. Further details concerning the implementation of the column generation procedure can be found in Subsection 10.2.

7.3 Improvements of the chain-packing bound

In this subsection we consider some improvements of the chain-packing bound introduced in the previous subsection. The linear programming formulation described previously is based on a decomposition of the precedence constraints C into single chains. While the time-lags between consecutive jobs in these chains have to be respected in all considered schedules, the other time-lags and the setup times are only weakly taken into account in the fractional relaxation (by the inequalities (7.5) and (7.6)).

In order to get an impression how conjunctions are taken into account in the constraints of the fractional relaxation, we consider the following example with one conjunction $i \to j \in C$ and $p_i = 2, p_j = 1, l_{ij} = 0$. Let x be the fractional solution $x_{\sigma_k} = 0.5$ for $k = 1, \ldots, 4$ in which the jobs i and j are scheduled according to the schedules $\mathcal{S}^i := \{\sigma_1, \sigma_2\}$ and $\mathcal{S}^j := \{\sigma_3, \sigma_4\}$ where $C_i^{\sigma_1} = 2, C_i^{\sigma_2} = 4, C_j^{\sigma_3} = 2, C_j^{\sigma_4} = 6$:



Obviously, this solution x satisfies the constraints (7.3) and (7.4). Furthermore, it respects the constraint (7.5) for $i \to j \in C$ due to

$$\sum_{\sigma \in \mathcal{S}^j} C_j^\sigma x_\sigma - \sum_{\sigma \in \mathcal{S}^i} C_i^\sigma x_\sigma = 4 - 3 = 1 = l_{ij} + p_j.$$

At this example we see that the constraints (7.5) only ensure that the "average completion times" $\bar{C}_j := \sum_{\sigma \in \mathcal{S}_{\Gamma(j)}} C_j^\sigma x_\sigma$ satisfy the inequalities $\bar{C}_j - \bar{C}_i \geq l_{ij} + p_j$ for $i \to j \in C$. These inequalities can often easily be satisfied by solutions x in which several jobs are split into many fractions (i.e. where we have a lot of small x_σ-values). Hence, the set of feasible solutions for the fractional relaxation is much larger than the set of feasible solutions for the 0-1-program and the corresponding bounds are weak.

In the following we show how the fractional relaxation can be strengthened by considering more complicated subgraphs instead of chains. For this purpose we partition the set C into λ subsets $\Lambda_1, \ldots, \Lambda_\lambda$, where each Λ_ϕ ($\phi = 1, \ldots, \lambda$) consists of a fixed number q of chains $\Gamma_\varrho^{(\phi)} : \varrho_1^{(\phi)} \to \varrho_2^{(\phi)} \to \ldots \to \varrho_{n_\varrho^{(\phi)}}^{(\phi)}$ for $\varrho = 1, \ldots, q$.

Anologously to the definition of a feasible chain-schedule, a schedule σ for a subset Λ_ϕ will be called a **feasible q-chain-schedule** if it

- respects the time windows $[r_j, d_j]$ of all jobs j belonging to subset Λ_ϕ,

- respects all precedences $i \to j \in C$ between jobs i, j belonging to subset Λ_ϕ (not only between jobs belonging to the same chain, but also between jobs belonging to different chains),
- respects all setups s_{ij} between jobs $i, j \in \Lambda_\phi$, and
- respects all time-lags l_{ij} between jobs $i, j \in \Lambda_\phi$ consecutively planned in σ.

Note that, according to this definition, in a feasible q-chain-schedule time-lags l_{ij} between jobs $i, j \in \Lambda_\phi$ which are not planned consecutively may be violated. The reason for this weaker definition of feasibility is again the computational complexity of the pricing problem (cf. the previous subsection), which increases if more time-lags are regarded. On the other hand, our computational tests have shown that in the calculated q-chain schedules σ only few time-lags between non-consecutive jobs are violated. This may be explained by the fact that if two jobs i, j are not consecutively planned in σ, at least one other job h must be scheduled in between. This job ensures that i and j are separated by at least $t_{ih} + p_h + t_{hj}$ time units. If this value is not smaller than l_{ij}, the time-lag is satisfied.

We denote by $\mathcal{S}_\phi$ the set of all feasible q-chain-schedules σ for subset Λ_ϕ and use the same notations as in the previous subsection. The structure of the linear programming formulation (7.3) to (7.7) remains the same, only the constraints have to be modified. Especially, some inequalities in the linear program can be omitted since the feasibility constraints which are induced by these inequalities are already incorporated into the definition of feasible q-chain-schedules. In conditions (7.3) we have to choose a q-chain-schedule for each subset Λ_ϕ $(\phi = 1, \ldots, \lambda)$. The capacity conditions (7.4) remain the same and constraints (7.5) are defined for all time-lags $i \xrightarrow{l_{ij}} j \in C$ which are not induced by transitivity. Note that contrary to the situation in Subsection 7.2 where only single chains are considered, conditions (7.5) are not automatically fulfilled for consecutive jobs of the same chain in a feasible q-chain-schedule. Thus, these conditions are not superfluous. But, since in the definition of a feasible q-chain-schedule we require that all setup times between jobs in the same subset Λ_ϕ are satisfied, constraints (7.6) corresponding to these disjunctions can be dropped. Especially, if we consider all chains together in one subset (i.e. we have $q = \gamma$), all disjunctive constraints can be dropped.

Again we consider the fractional relaxation of this formulation and call the largest bound obtained by this relaxation the ***q*-chain-packing bound**. In order to obtain an initial basic solution, we again determine the earliest start schedule where each job is started as early as possible (taking into account all its predecessors and time-lags). By decomposing this schedule into schedules for the subsets Λ_ϕ we obtain λ schedules σ^ϕ. But, since in the earliest start schedule the machine capacity and the disjunctions may be violated, schedule σ^ϕ is not a feasible q-chain-schedule for subset Λ_ϕ. Nevertheless, we can use these infeasible schedules to obtain an initial basis. We treat the corresponding columns as infeasible columns and penalize them with a positive coefficient in the objective function. Then the column generation process can be started as usual. If it terminates, either no infeasible columns are contained in the optimal solution, or some infeasible variables remain in the basis with positive values. In the first case we have found a feasible solution, in the second case infeasibility can be stated.

It remains to describe the modifications of the pricing problem. The reduced costs of a q-chain-schedule σ with completion times C_i^σ are again given by a sum of the form

$$c(\sigma) = \sum_{i \in \Lambda_\phi} c(i, C_i^\sigma),$$

where $c(i,t)$ defines the contribution of job $i \in \Lambda_\phi$ finishing at time t.

The pricing algorithm is an extension of the dynamic programming algorithm for the case of a single chain presented above. In the following we consider an arbitrary, but fixed subset Λ_ϕ consisting of q chains $\Gamma_\varrho^{(\phi)} : \varrho_1^{(\phi)} \rightarrow \varrho_2^{(\phi)} \rightarrow \ldots \rightarrow \varrho_{n_\varrho^{(\phi)}}^{(\phi)}$ for $\varrho = 1, \ldots, q$. In order to simplify the notation in the following we will omit the upper index (ϕ) and assume that $\Gamma_\varrho^{(\phi)}$ has the form $\Gamma_\varrho : \varrho_1 \rightarrow \varrho_2 \rightarrow \ldots \rightarrow \varrho_{n_\varrho}$.

Our goal is to calculate a q-chain-schedule $\sigma^* \in \mathcal{S}_\phi$ which minimizes $c(\sigma)$. Since the orders for all jobs i in the chains Γ_ϱ $(\varrho = 1, \ldots, q)$ are fixed, the problem is to merge these q chains in such a way that the time windows, the precedences and setup times between all jobs, and the time-lags between any two consecutive jobs are respected and the sum objective $c(\sigma)$ is minimized.

In the following we consider partial q-chain-schedules for subsets of jobs. These partial schedules are represented by tuples $(k_1, \ldots, k_q)$ where $k_\varrho \in \{0, 1, \ldots, n_\varrho\}$ indicates how many jobs of chain Γ_ϱ are contained in the current partial schedule. Furthermore, for the last scheduled job j in this single-machine schedule we store the index $\alpha \in \{1, \ldots, q\}$ of the chain Γ_α to which j belongs.

For these tuples $(k_1, \ldots, k_q)$, $\alpha \in \{1, \ldots, q\}$ and $t = 1, \ldots, T$ let $F_{(k_1,\ldots,k_q),\alpha}(t)$ be the minimal costs for all feasible q-chain-schedules $\sigma \in \mathcal{S}_\phi$ consisting of the first k_ϱ jobs of chain Γ_ϱ for $\varrho = 1, \ldots, q$, where the last scheduled job belongs to chain Γ_α and this last job $j := \alpha_{k_\alpha}$ finishes not later than time t. Let $PRE(j) := \alpha_{k_\alpha - 1}$ denote the immediate predecessor of job j in the same chain Γ_α.

For a q-chain-schedule realizing $F_{(k_1,\ldots,k_q),\alpha}(t)$ there are three possibilities:

- either job j finishes before time t, i.e. it finishes not later than time $t-1$,
- job j finishes exactly at time t and directly before j its predecessor $i := PRE(j)$ in the same chain is processed, or
- job j finishes exactly at time t and directly before j a job $h := \varrho_{k_\varrho}$ of another chain Γ_ϱ with $\varrho \neq \alpha$ is processed.

In the first case the costs $F_{(k_1,\dots,k_q),\alpha}(t)$ are equal to $F_{(k_1,\dots,k_q),\alpha}(t-1)$. In the second case we have to select the best chain-schedule for the first $k_\alpha - 1$ jobs of chain Γ_α and the first k_ϱ jobs of the other chains Γ_ϱ with $\varrho \neq \alpha$, where i is the last job and i finishes not later than time $t - p_j - l_{ij}$. Then the costs are equal to $F_{(k_1,\dots,k_\alpha-1,\dots,k_q),\alpha}(t - p_j - l_{ij}) + c(j,t)$. In the third case we have to select the best chain-schedule for the first $k_\alpha - 1$ jobs of chain Γ_α and the first k_ϱ jobs of the other chains Γ_ϱ with $\varrho \neq \alpha$, where h is the last job and h finishes not later than time $t - p_j - t_{hj}$. Then the costs are equal to $F_{(k_1,\dots,k_\alpha-1,\dots,k_q),\varrho}(t - p_j - t_{hj}) + c(j,t)$.

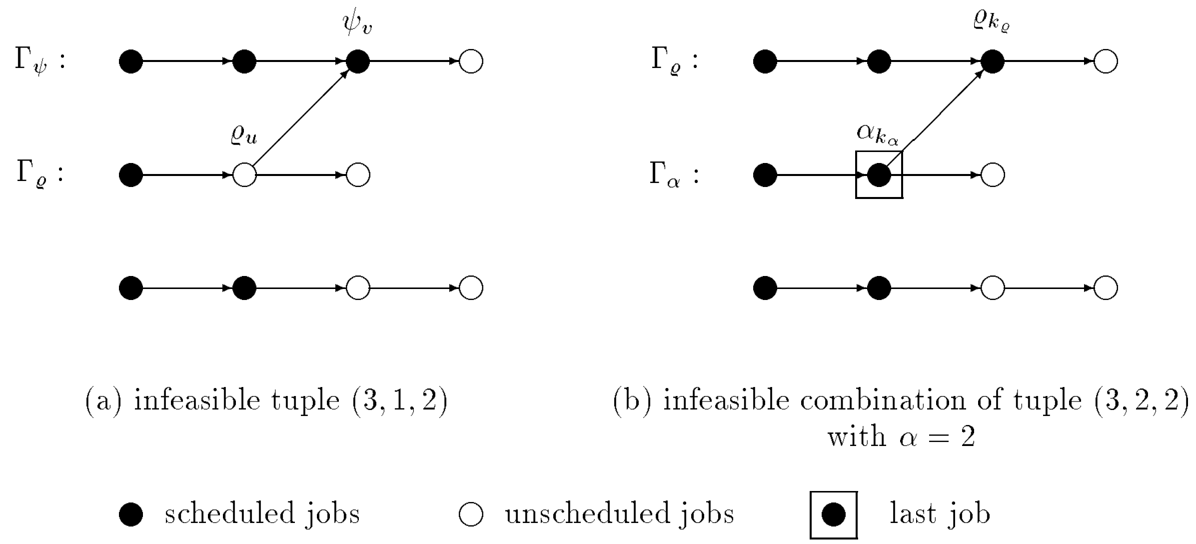


Figure 7.1: Infeasible situations

A tuple $(k_1,\dots,k_q)$ represents an infeasible subset w.r.t. the precedence relations if an unscheduled job ϱ_u in a chain Γ_ϱ (i.e. $u \in \{k_\varrho + 1, \dots, n_\varrho\}$) and a scheduled job ψ_v in a chain Γ_ψ (i.e. $v \in \{1,\dots,k_\psi\}$) exist which are linked by a conjunction $\varrho_u \to \psi_v \in C$ (cf. Figure 7.1 (a)). A tuple $(k_1,\dots,k_q)$ combined with an index α represents an infeasible partial schedule if $\alpha_{k_\alpha} \to \varrho_{k_\varrho} \in C$ holds for another chain index $\varrho \neq \alpha$, since in this case job α_{k_α} cannot be the last scheduled job in the subset represented by $(k_1,\dots,k_q)$ (cf. Figure 7.1 (b)). Additionally, each job j cannot be finished before time $t < r_j + p_j$. In all these cases the corresponding costs are set to infinity.

Summarizing, the $F_{(k_1,\dots,k_q),\alpha}(t)$-values can be calculated recursively as follows.

Initialization:

$$F_{(k_1,\dots,k_\alpha=0,\dots,k_q),\alpha}(t) := \infty \text{ for } \alpha = 1,\dots,q,\ k_\varrho \in \{0,\dots,n_\varrho\}\ \forall\ \varrho \neq \alpha;\ t = 1,\dots,T,$$

$$F_{(0,\dots,0,k_\alpha=1,0,\dots,0),\alpha}(t) := \infty \quad \text{for all } t < r_{\alpha_1} + p_{\alpha_1},$$

$$F_{(0,\dots,0,k_\alpha=1,0,\dots,0),\alpha}(t) := \infty \quad \text{for all } t \in [r_{\alpha_1} + p_{\alpha_1}, d_{\alpha_1}] \quad \text{if } \varrho_u \to \alpha_1 \in C \text{ for an } u \in \{1,\dots,n_\varrho\},\ \varrho \neq \alpha.$$

Otherwise, we set

$$F_{(0,\dots,0,k_\alpha=1,0,\dots,0),\alpha}(t) := \min\{F_{(0,\dots,0,k_\alpha=1,0,\dots,0),\alpha}(t-1),\ c(\alpha_1,t)\} \quad \text{for all } t \in [r_{\alpha_1}+p_{\alpha_1}, d_{\alpha_1}].$$

Recursion:

$$F_{(k_1,\dots,k_q),\alpha}(t) = \begin{cases} \infty, & t < r_j + p_j, \text{ or } j \to \varrho_{k_\varrho} \in C \text{ for one } \varrho \neq \alpha \\ & \text{or } \varrho_u \to \psi_v \in C \text{ for } u \in \{k_\varrho+1,\dots,n_\varrho\}, v \in \{1,\dots,k_\psi\} \\ & \text{with } \varrho \neq \psi \\ \min\{ & F_{(k_1,\dots,k_q),\alpha}(t-1), \\ & F_{(k_1,\dots,k_\alpha-1,\dots,k_q),\alpha}(\min\{t-p_j-l_{ij}, d_i\}) + c(j,t), \\ & \min_{\varrho\neq\alpha}\{F_{(k_1,\dots,k_\alpha-1,\dots,k_q),\varrho}(\min\{t-p_j-t_{\varrho_{k_\varrho}j}, d_{\varrho_{k_\varrho}}\}) + c(j,t)\} \\ \}, & t \in [r_j+p_j, d_j] \end{cases}$$

for all $(k_1,\dots,k_q)$ with $k_\varrho \in \{0,\dots,n_\varrho\}$; $\varrho,\alpha = 1,\dots,q$; and $j := \alpha_{k_\alpha}, i := PRE(j)$.

To obtain the optimal solution value $c(\sigma^*)$, we have to schedule all jobs (represented by the tuple $(n_1,\dots,n_q)$) and to decide which job is scheduled last (represented by an index $\alpha \in \{1,\dots,q\}$), i.e. $c(\sigma^*)$ is given by $\min\limits_{\alpha=1}^{q}\{F_{(n_1,\dots,n_q),\alpha}(d_{\alpha_{n_\alpha}})\}$.

We have to calculate $F_{(k_1,\dots,k_q),\alpha}(t)$ for all tuples $(k_1,\dots,k_q)$ with $k_\varrho \in \{0,1,\dots,n_\varrho\}$ for $\varrho = 1,\dots,q$, for all chains $\alpha = 1,\dots,q$ and for all time periods $t \in [r_j+p_j, d_j]$ with $j := \alpha_{k_\alpha}$. Since each computation can be done in $O(q)$ time, the complexity of the dynamic programming procedure for subset Λ_ϕ is $O(q^2\prod\limits_{\varrho=1}^{q} n_\varrho^{(\phi)}\Delta_\phi^{\max}) = O(q^2\prod\limits_{\varrho=1}^{q} n_\varrho^{(\phi)}T)$ where $\Delta_\phi^{\max} := \max\limits_{j\in\Lambda_\phi}\{d_j - p_j - r_j\}$ denotes the maximal length of a time window of a job j belonging to subset Λ_ϕ. Finding a q-chain-schedule $\sigma^* \in \mathcal{S}_\phi$ which realizes $c(\sigma^*)$ can again be done by a backward construction.

As mentioned previously, according to the definition of a feasible q-chain-schedule time-lags l_{ij} between jobs $i,j \in \Lambda_\phi$ which are not planned consecutively in σ may be violated. With stronger definitions of feasibility some of these violations can be forbidden, but then the complexity of the dynamic programming procedure increases enormously. For example, if we additionally require that minimal time-lags l_{ij} between consecutive jobs i,j in the same chain have to be respected, instead of storing a single time period t for a partial schedule, we have to store for each chain $\Gamma_\varrho^{(\phi)}$, $(\varrho = 1,\dots,q)$ a time period t_ϱ corresponding to the completion time of the last scheduled job in this chain. Then the complexity increases to $O(q^2\prod\limits_{\varrho=1}^{q} n_\varrho^{(\phi)}T^q)$. If we want to respect all time-lags, even for each job its completion time in a partial schedule has to be stored.

7.4 $\mathcal{NP}$-hardness of the fractional chain-packing relaxation

In this subsection we study the fractional relaxation of problem (7.3) to (7.7) from the complexity point of view and show that solving this relaxation is still an $\mathcal{NP}$-hard problem. In the following we consider the feasibility problem $1 \mid r_j, d_j \mid -$ which is a special case of the considered single-machine problem $1 \mid prec(l_{ij}), r_j, d_j, s_{ij} \mid -$ where no precedence constraints C, no minimal time-lags l_{ij} and no sequence-dependent setups s_{ij} exist. Since we have no precedences C, we consider a decomposition into single jobs $j = 1, \ldots, n$, where n is the number of jobs. Then the formulation (7.3) to (7.7) simplifies to

$$\sum_{\sigma \in \mathcal{S}_j} x_\sigma = 1 \qquad j = 1, \ldots, n \tag{7.24}$$

$$\sum_{j=1}^{n} \sum_{\sigma \in \mathcal{S}_j} b_{\sigma t} x_\sigma \leq 1 \qquad t = 1, \ldots, T \tag{7.25}$$

$$x_\sigma \in \{0, 1\} \qquad \forall\, \sigma \in \bigcup_{j=1}^{n} \mathcal{S}_j. \tag{7.26}$$

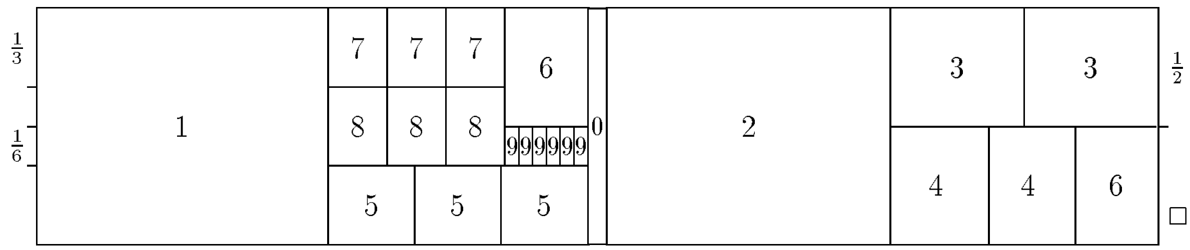
At first we show with an example that the fractional relaxation may have a feasible solution, although the binary program (7.3) to (7.7) has no feasible solution.

Example 7.2: We are given $n = 10$ jobs with the following data:

i	0	1	2	3	4	5	6	7	8	9
p_i	10	148	144	68	47	44	42	30	30	7
r_i	280	0	0	0	0	0	0	0	0	0
d_i	290	570	570	570	570	570	570	570	570	570

If this instance has a feasible solution respecting all time windows $[r_j, d_j]$, job 0 has to be planned exactly in its time window $[280, 290]$, since $r_0 + p_0 = d_0$ holds. Therefore the remaining jobs have to be planned in the intervals $[0, 280]$ and $[290, 570]$ both of length 280, i.e. we have to find a partition (I_1, I_2) of $\{1, \ldots, 9\}$ with $\sum\limits_{j \in I_1} p_j = \sum\limits_{j \in I_2} p_j = 280$. Checking all possibilities it can be shown that no such partition exists, i.e. (7.24) to (7.26) has no feasible solution.

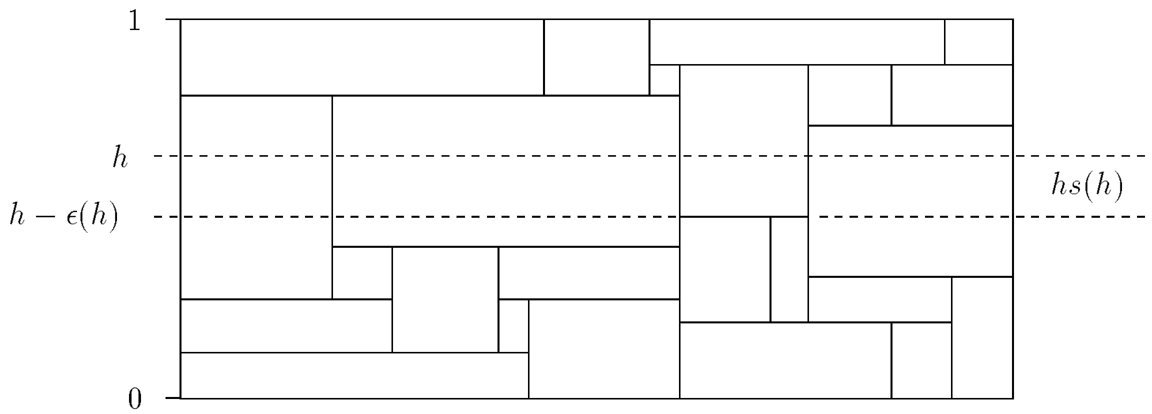
Relaxing condition (7.26) the following fractional solution exists, where we have values $x_\sigma \in \{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{6}\}$:



□

In the following we will show that the decision problem whether a feasible fractional solution for (7.24) to (7.25) exists or not, is a hard problem in view of complexity theory.

Given a feasible fractional solution x, let $\mathcal{S}_j^x := \{\sigma \in \mathcal{S}_j \mid x_\sigma > 0\}$ be the set of all job schedules used for job j in solution x. Due to conditions (7.25) all job schedules can be packed into the rectangle $[0,T] \times [0,1]$ without overlappings. This means that for each schedule $\sigma \in \mathcal{S}_j^x$ with starting time S_j^σ of job j we can define rectangles $[S_j^\sigma, S_j^\sigma + p_j] \times [a_j^\sigma, b_j^\sigma] \subseteq [0,T] \times [0,1]$ with $b_j^\sigma - a_j^\sigma = x_\sigma$ in such a way that none of these rectangles overlap in the big rectangle $[0,T] \times [0,1]$. For such a packing schedule and each value $h \in]0,1]$ let $hs(h)$ be the horizontal schedule in $[0,T] \times]h - \epsilon(h), h[$ where $\epsilon(h) > 0$ is chosen sufficiently small in such a way that the interval $]h-\epsilon(h), h[$ contains no a_j^σ- or b_j^σ-value (e.g. $\epsilon(h) := h - \max\{a_j^\sigma | a_j^\sigma \leq h\}$).



With this definition we can now show the $\mathcal{NP}$-hardness of the considered problem.

Theorem 7.1: The problem of deciding whether a feasible fractional solution for problem $1 \mid r_j, d_j \mid -$ exists or not is $\mathcal{NP}$-hard.

Proof: We reduce the following variant of the partitioning problem, which is $\mathcal{NP}$-complete in the ordinary sense (cf. Garey & Johnson [1979]), to problem $1 \mid r_j, d_j \mid -$:

EVEN-ODD-PARTITION (EO-PART): Given $2m$ positive numbers $a_1, \ldots, a_{2m}$ with $\sum_{i=1}^{2m} a_i = 2b$. Does there exist a partition (I_1, I_2) of the index set $\{1, \ldots, 2m\}$ such that $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i = b$ and I_1 contains exactly one element from $\{a_{2i-1}, a_{2i}\}$ for $i = 1, \ldots, m$?

Let $\Delta_i := |a_{2i} - a_{2i-1}|$ and $A_i := \max\{a_{2i-1}, a_{2i}\}$ for $i = 1, \ldots, m$.

Given an arbitrary instance of EO-PART, we construct the following instance of problem $1 \mid r_j, d_j \mid -$ with $n := 3m + 1$ jobs:

- $2m$ jobs $1, \ldots, 2m$ with

$$\left.\begin{array}{rcl} p_{2i-1} & := & a_{2i-1} + K_i, \\ p_{2i} & := & a_{2i} + K_i, \\ r_{2i-1} = r_{2i} & := & \sum_{\lambda=1}^{i-1} \min\{p_{2\lambda-1}, p_{2\lambda}\} + (i-1)B, \\ d_{2i-1} = d_{2i} & := & T, \end{array}\right. \qquad (i = 1, \ldots, m)$$

- m jobs $B_1, \ldots, B_m$ with

$$\left.\begin{array}{rcl} p_{B_i} & := & B, \\ r_{B_i} & := & \sum_{\lambda=1}^{i} \min\{p_{2\lambda-1}, p_{2\lambda}\} + (i-1)B, \\ d_{B_i} & := & \sum_{\lambda=1}^{i} \max\{p_{2\lambda-1}, p_{2\lambda}\} + iB, \end{array}\right. \qquad (i = 1, \ldots, m)$$

- job 0 with

$$p_0 := 1, \qquad r_0 := b + \sum_{i=1}^{m} K_i + mB, \qquad d_0 := r_0 + p_0$$

with $J := \{0, 1, \ldots, 2m, B_1, \ldots, B_m\}$ and constants

$$\begin{array}{rcl} T & := & d_0 + b + \sum_{i=1}^{m} K_i = 2b + 2\sum_{i=1}^{m} K_i + mB + 1 = \sum_{j \in J} p_j, \\ B & := & \sum_{i=1}^{m} \Delta_i + 1, \\ K_m & := & \sum_{\lambda=1}^{m} \Delta_\lambda + A_m + 1, \\ K_i & := & \sum_{\lambda=1}^{i} \Delta_\lambda + A_i + 1 + A_{i+1} + K_{i+1} \qquad (i = 1, \ldots, m-1). \end{array}$$

We show that EO-PART has a solution if and only if the constructed instance of $1 \mid r_j, d_j \mid -$ has a feasible fractional solution.

Assume that (I_1, I_2) is a solution of EO-PART. We renumber the elements in such a way that I_1 consists of all elements a_{2i-1} with odd indices. Then we construct the following non-fractional schedule for $1 \mid r_j, d_j \mid -$ with is given by the permutation

$$\pi := (1, B_1, 3, B_2, 5, B_3, \ldots, 2m-1, B_m, 0, 2, 4, 6, 8, \ldots, 2m).$$

The following properties of π can easily be shown for all $i = 1, \ldots, m$:

- $S_{2i-1} = \sum_{\lambda=1}^{i-1} p_{2\lambda-1} + (i-1)B \geq \sum_{\lambda=1}^{i-1} \min\{p_{2\lambda-1}, p_{2\lambda}\} + (i-1)B = r_{2i-1},$

- $S_{B_i} = \sum_{\lambda=1}^{i} p_{2\lambda-1} + (i-1)B \geq \sum_{\lambda=1}^{i} \min\{p_{2\lambda-1}, p_{2\lambda}\} + (i-1)B = r_{B_i}$,

- $C_{B_i} = S_{B_i} + B = \sum_{\lambda=1}^{i} p_{2\lambda-1} + iB \leq \sum_{\lambda=1}^{i} \max\{p_{2\lambda-1}, p_{2\lambda}\} + iB = d_{B_i}$.

Thus, we can conclude that the partial schedule $(1, B_1, 3, B_2, 5, \dots, 2m-1, B_m)$ contains no idle times and all these jobs are processed within their time windows. Furthermore, we have

- $S_0 = \sum_{j \in I_1} p_j + \sum_{i=1}^{m} p_{B_i} = \sum_{i=1}^{m}(a_{2i-1} + K_i) + mB = b + \sum_{i=1}^{m} K_i + mB = r_0$,

- $C_{2m} = d_0 + \sum_{j \notin I_1} p_j = d_0 + b + \sum_{i=1}^{m} K_i = T$.

Hence, also the jobs $0, 2, 4, \dots, 2m$ respect their time windows, i.e. π defines a feasible solution for $1 \mid r_j, d_j \mid -$.

Conversely, assume that the constructed instance of $1 \mid r_j, d_j \mid -$ has a feasible fractional solution x with a corresponding packing schedule σ^x in $[0, T] \times [0, 1]$. Then σ^x contains no idle times, since $T = \sum_{j \in J} p_j$ is the sum of all processing times. Since $d_0 = r_0 + p_0$, we must have $S_0 = r_0$ and $C_0 = d_0$ for all schedules in $\mathcal{S}_0^x$, i.e. job 0 is scheduled completely in its time window $[r_0, d_0]$. Due to $d_{B_i} - r_{B_i} = \sum_{\lambda=1}^{i} \Delta_\lambda + B$, and $\sum_{\lambda=1}^{i} \Delta_\lambda < B = p_{B_i}$, no horizontal schedule $hs(h)$ can contain a job B_i twice. Thus, in each $hs(h)$ for $h \in]0, 1]$ each B_i is contained exactly once.

In the following we will show that in a schedule $hs(h)$ also the jobs $1, \dots, 2m$ cannot be scheduled twice in the interval $[0, d_0]$. For this purpose we consider an arbitrary schedule $hs(h)$. Let X_i^h be the interval between the jobs B_{i-1} and B_i $(i = 1, \dots, m)$ in $hs(h)$. We will show that exactly one job from $\{2i-1, 2i\}$ must be scheduled in the interval X_i^h $(i = 1, \dots, m)$. If we then let I_1 be the set of the m jobs which are scheduled in $hs(h)$ in the intervals X_i^h $(i = 1, \dots, m)$, we have $\sum_{j \in I_1} p_j = r_0 - mB = b + \sum_{i=1}^{m} K_i$. This implies $\sum_{j \in I_1} a_j = b$, i.e. I_1 defines a solution of EO-PART.

Now let us consider an interval X_i^h. For the length $l(X_i^h)$ we have

$$\begin{aligned} l(X_i^h) &= S_{B_i} - C_{B_{i-1}} \leq d_{B_i} - p_{B_i} - (r_{B_{i-1}} + p_{B_{i-1}}) = d_{B_i} - r_{B_{i-1}} - 2B \\ &= \sum_{\lambda=1}^{i} \max\{p_{2\lambda-1}, p_{2\lambda}\} + iB - (\sum_{\lambda=1}^{i-1} \min\{p_{2\lambda-1}, p_{2\lambda}\} + (i-2)B) - 2B \end{aligned}$$

$$= \sum_{\lambda=1}^{i-1} [\max\{p_{2\lambda-1}, p_{2\lambda}\} - \min\{p_{2\lambda-1}, p_{2\lambda}\}] + \max\{p_{2i-1}, p_{2i}\}$$
$$= \sum_{\lambda=1}^{i-1} \Delta_\lambda + A_i + K_i. \tag{7.27}$$

We claim that only the jobs $\{2i-1, 2i\}$ can be planned in the interval X_i^h. First we show that only the jobs $1, 2, \ldots, 2i-1, 2i$ are available in X_i^h. Since the jobs $1, \ldots, 2m$ are ordered according to non-decreasing release dates, it is sufficient to show that $2i-1$ and $2i$ are available in X_i^h, and $2i+1$ and $2i+2$ are not. But, these two properties follow immediately from the fact that $r_{2i-1} = r_{2i} = r_{B_{i-1}} + B$ and $r_{2i+1} = r_{2i+2} = r_{B_i} + B = d_{B_i} - \sum_{\lambda=1}^{i} \Delta_\lambda > d_{B_i} - B$.

Now we show that a job from $\{2i-1, 2i\}$ cannot be planned in the later intervals $X_{i+1}^h, \ldots, X_m^h$, since

$$\min\{p_{2i-1}, p_{2i}\} = \min\{a_{2i-1}, a_{2i}\} + K_i \geq K_i > \sum_{\lambda=1}^{i} \Delta_\lambda + A_{i+1} + K_{i+1} \geq l(X_{i+1}^h)$$

due to (7.27), and $K_{i+1} > \sum_{\lambda=1}^{i+1} \Delta_\lambda + A_{i+2} + K_{i+2} \geq l(X_{i+2}^h)$, etc.
Thus, only the jobs $\{2i-1, 2i\}$ can be planned in X_i^h. It remains to prove that in X_i^h not both jobs from $\{2i-1, 2i\}$ can be scheduled and no job from $\{2i-1, 2i\}$ can appear twice. Due to (7.27) we get

$$2\min\{p_{2i-1}, p_{2i}\} = 2\min\{a_{2i-1}, a_{2i}\} + 2K_i \geq 2K_i > l(X_i^h),$$

i.e. at most one job from $\{2i-1, 2i\}$ can be contained in X_i^h. Since the rectangle $[0, T] \times [0, 1]$ is completely filled, exactly one job from $\{2i-1, 2i\}$ must be scheduled in the interval X_i^h, which completes the proof. □

8 The calculation of heuristic solutions for problem $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$

In this section we describe the calculation of heuristic solutions for the single-machine problem $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$. After introducing some preliminaries in Subsection 8.1, we present two heuristic approaches. In Subsection 8.2 we describe a simple heuristic based on priority rules, and in Subsection 8.3 we propose concepts for more time-consuming local search methods. Based on these ideas a tabu search algorithm is developed in Subsection 8.4.

8.1 Representation of feasible solutions

The set of feasible solutions for problem $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$ consists of all feasible schedules respecting the time-lags l_{ij}, release dates r_j and setup times s_{ij}. They may be represented by the starting times S_j for all jobs $j \in V = \{0, 1, \ldots, n, n+1 = *\}$. In the following we will show that the solution space may be reduced to the set of all permutations of the jobs $j \in V$ which are compatible with the precedence relations (i.e. if $i \rightarrow j \in C$ and $\pi_k = i$, $\pi_l = j$ in a permutation π, we have $k < l$). Obviously, each feasible schedule S for the single-machine problem uniquely defines a permutation $\pi^S = (0, \pi_1^S, \ldots, \pi_n^S, *)$ in which the jobs are ordered according to increasing starting times. On the other hand, with each permutation $\pi = (0, \pi_1, \ldots, \pi_n, *)$ which is compatible with the precedence constraints we associate the set of all feasible schedules in which the jobs are processed according to the order defined by π.

Among these schedules represented by π a schedule S^π with minimal makespan may be determined as follows. We set $S_0^\pi := 0$ and calculate for $k = 1, \ldots, n+1$ the starting time S_j^π of job $j := \pi_k$ by

$$S_j^\pi := \max\left[\max_{\{i \mid i \rightarrow j \in C\}} \{S_i^\pi + p_i + l_{ij}\}, S_{\pi_{k-1}}^\pi + p_{\pi_{k-1}} + s_{\pi_{k-1}j}\right]. \tag{8.1}$$

Obviously, job j cannot start before the time given by the first term

$$L_j^\pi := \max_{\{i \mid i \rightarrow j \in C\}} \{S_i^\pi + p_i + l_{ij}\}, \tag{8.2}$$

which ensures that all minimal time-lags l_{ij} are respected. The second term in (8.1) takes into account that j cannot start before its immediate predecessor π_{k-1} on the machine has finished and the setup $s_{\pi_{k-1}j}$ has been done. The complexity for calculating the schedule S^π is $O(n + |C|) = O(n^2)$.

The calculation of the starting times S_j^π implies that the resulting schedule S^π is an optimal schedule among all schedules represented by π since each job starts as early as possible. The makespan of S^π is given by $C_{\max}(S^\pi) = S_{\pi_{n+1}}^\pi$. This value corresponds to the length of a longest path in the acyclic graph $G^\pi = (V, A^\pi)$ with arcs $A^\pi := C \cup \{ (\pi_k, \pi_{k+1}) \mid k = 1, \ldots, n-1\}$ and arc lengths

$$d_{ij}^\pi := \begin{cases} p_i + l_{ij}, & \text{if } i \to j \in C \\ p_i + s_{ij}, & \text{if } i \to j \notin C \end{cases} \quad \text{for } (i,j) \in A^\pi.$$

For further considerations we are also interested in an optimal schedule $\bar{S}^\pi$ represented by π in which each job starts as late as possible. After calculating $C_{\max}(\pi) := C_{\max}(S^\pi)$, schedule $\bar{S}^\pi$ may be constructed by a backward calculation. We set $\bar{S}_{n+1}^\pi := C_{\max}(\pi)$ and calculate for $k = n, \ldots, 0$ the starting time $\bar{S}_i^\pi$ of job $i := \pi_k$ by

$$\bar{S}_i^\pi := \min\,[\min_{\{j \mid i \to j \in C\}} \{\bar{S}_j^\pi - p_i - l_{ij}\}, \bar{S}_{\pi_{k+1}}^\pi - p_{\pi_k} - s_{i\pi_{k+1}}]. \tag{8.3}$$

Note that S_i^π is the earliest starting time and $\bar{S}_i^\pi$ is the latest starting time for job i among all feasible schedules which are represented by π with makespan $C_{\max}(\pi)$.
Jobs $i \in V$ for which $S_i^\pi = \bar{S}_i^\pi$ holds, are called **critical**, since they cannot be moved in any optimal schedule represented by π. A sequence $(0 = i_0, i_1, \ldots, i_k, i_{k+1} = *)$ of critical jobs is called a **critical path** if for all pairs $(i,j) := (i_\lambda, i_{\lambda+1})$ with $\lambda \in \{0, \ldots, k\}$ the starting time of job j is determined by job i, i.e. $S_j^\pi = S_i^\pi + d_{ij}^\pi$ holds. Such a path always exists and it is easy to see that it corresponds to a longest path in the graph G^π introduced previously.

Based on this representation of feasible schedules for problem $1 \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$ by permutations, in the next subsections we describe two approaches for calculating heuristic solutions.

8.2 A priority-rule based heuristic

Heuristics based on priority rules are often used for combinatorial optimization problems since they even find solutions for large instances in a short amount of time and are easy to implement. However, especially for more complex problems, the quality of these simple heuristics is not satisfactory. Therefore, a priority-rule based heuristic is often only used to construct starting solutions quickly, which may then be improved by more sophisticated local search methods.

Heuristics based on priority rules construct a schedule iteratively, starting at time 0. Based on certain priority rules in each step a job from a specified subset of jobs is chosen to be scheduled next. This decision is final, i.e. the decision is not corrected later on. In the following we describe the details of our approach. At each decision point t (initially $t := 0$) we have

- a set A of jobs whose predecessors are all finished,
- a subset $A(t) \subseteq A$ consisting of all jobs in A which are available at time t, i.e. $A(t) := \{i \in A \mid r_i \leq t\}$.

Due to the release dates it may happen that if we plan a job $i \in A(t)$ at time t, a more important job becomes available shortly after time t. In this case it could be better to introduce a short idle time at time t and schedule the more important job before all jobs from set $A(t)$. Thus, we look a bit ahead and enlarge $A(t)$ to

- the set $A(t') := \{i \in A \mid r_i \leq t'\} \supseteq A(t)$ consisting of all jobs in A which are available at time t or become available not later than time

$$t' := t + \min_{i \in A(t)} \{p_i + \min_{j \in A \setminus \{i\}} s_{ij}\}.$$

In each step we choose a job i from the subset $A(t')$ according to some priority rule, start i at time $S_i := \max\{t, r_i\}$ and delete i from the set A. Scheduling job i influences the starting times of the unscheduled jobs. All these consequences can be taken into account by modifying the heads of the unscheduled jobs as follows. Each conjunctive successor j of i cannot start before time $S_i + p_i + l_{ij}$, i.e. we update the heads of all jobs j with $i \to j \in C$ according to $r_j := \max\{r_j, S_i + p_i + l_{ij}\}$ and test if j has to be inserted into the set A of all jobs whose predecessors are all scheduled. Although we do not know the scheduling order for the remaining jobs in A at this point, we may update the heads r_j of all other jobs $j \in A$ according to $r_j := \max\{r_j, S_i + p_i + s_{ij}\}$. This is correct since a job h which will be scheduled next cannot start before time $S_i + p_i + t_{ih}$ (again t_{ih} denotes either a time-lag l_{ih} or a setup s_{ih}) and all other jobs $j \in A$ which are scheduled after h cannot start before time $S_i + p_i + t_{ih} + p_h + s_{hj} \geq S_i + p_i + s_{ij}$ due to the triangle inequality $t_{ih} + p_h + s_{hj} \geq s_{ij}$ (cf. (6.1)). Then we proceed with the next decision point $t := \min_{i \in A}\{r_i\}$. Our approach is summarized in the following

`PROCEDURE Priority-rule heuristic`

Let $A := \{i \in V \mid i$ has no predecessor$\}$;
WHILE $A \neq \emptyset$ DO BEGIN
    $t := \min_{i \in A}\{r_i\}$;
    Let $A(t) := \{i \in A \mid r_i \leq t\}$;
    $t' := t + \min_{i \in A(t)} \{p_i + \min_{j \in A \setminus \{i\}} s_{ij}\}$;
    Let $A(t') := \{i \in A \mid r_i \leq t'\}$;
    Choose $i \in A(t')$ according to some priority rule;
    Start i at time $S_i := \max\{t, r_i\}$;
    $A := A \setminus \{i\}$;
    FOR ALL $j \in A$ DO
        $r_j := \max\{r_j, S_i + p_i + s_{ij}\}$;
    FOR ALL j with $i \to j \in C$ DO BEGIN
        $r_j := \max\{r_j, S_i + p_i + l_{ij}\}$;
        IF all predecessors of j are scheduled THEN
            $A := A \cup \{j\}$;
    END
END

The running time of this procedure is $O(n^2)$. There are several possibilities for a priority-rule in Step 7. We tested the following four rules:

- MIN-HEAD: a job i with the smallest head r_i is chosen,
- MAX-TAIL: a job i with the largest tail q_i is chosen,
- MAX-SUCC: a job i with the largest number of successors is chosen,
- MIN-DP: a job i is chosen which determines t', i.e. a job i for which we have

$$p_i + \min_{j \in A \setminus \{i\}} s_{ij} = \min_{i \in A(t)} \{p_i + \min_{j \in A \setminus \{i\}} s_{ij}\}.$$

Note that for the first rule MIN-HEAD the time t' is not needed, since the job with smallest release date is always chosen.

8.3 Foundations for local search methods

In this subsection we give some foundations for applying local search algorithms (like iterative improvement or tabu search) to problem $1 \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$. We assume that we are given a permutation π with the associated schedules S^π and $\bar{S}^\pi$. Our goal is to improve this solution by local search methods. These methods start with an initial solution and iteratively search through the solution space to find better solutions. In each step the current solution is replaced by a solution in some neighborhood of the solution. The choice of the neighborhood structure has an important influence on the efficiency of local search methods since it determines how we navigate through the solution space. In this subsection we first describe how neighborhoods can be defined based on structural properties of the single-machine problem. Then we present some techniques which accelerate the evaluation of solutions in the proposed neighborhoods to make the search process more efficient.

In order to incorporate problem-specific properties of the single-machine problem into the definition of neighborhood structures we use a so-called block approach. Such an approach was first proposed for the single-machine problem $1 \mid r_j \mid L_{\max}$ (Grabowski et al. [1986]). Later it was successfully adapted to some other scheduling problems (like the job-shop or flow-shop problem, cf. Brucker et al. [1994A], Nowicki & Smutnicki [1996], Nowicki & Smutnicki [1996A]). With the definition of blocks it can be stated that only certain changes of a solution π may improve the current makespan $C_{\max}(\pi)$ and in the search process only such solutions will be considered as candidates for neighbored solutions.

Let $P^\pi = (0 = i_0, i_1, \ldots, i_k, i_{k+1} = *)$ be a critical path associated with π. A subsequence $(i_b, \ldots, i_f)$ of at least two successive jobs (i.e. $f > b$) on P^π is called a **block** if

- the jobs of the subsequence are processed consecutively on the single machine without idle times (idle time is time which is not used for processing or setups),
- no conjunction exists between two consecutive jobs of the subsequence, and
- enlarging the subsequence by one job leads to a subsequence which does not fulfill both of the above properties.

The definition of a block implies that for all pairs $(i_\lambda, i_{\lambda+1})$ in a block $(i_b, \ldots, i_f)$ the equality $S^\pi_{i_{\lambda+1}} = S^\pi_{i_\lambda} + p_{i_\lambda} + s_{i_\lambda i_{\lambda+1}}$ holds.

The following theorem is the basis for defining suitable neighborhoods on the set of all permutations.

Theorem 8.1: Let π be an arbitrary permutation with makespan $C_{\max}(\pi)$ and let P^π be a critical path associated with π. If another permutation π' with $C_{\max}(\pi') < C_{\max}(\pi)$ exists, then in π' at least two jobs of a block on P^π are processed in the opposite order as in π.

Proof: Let $P^\pi = (0, u_1^1, u_2^1, ..., u_{m_1}^1, \ldots, u_1^k, u_2^k, ..., u_{m_k}^k, *)$, where $u_1^j, ..., u_{m_j}^j$ $(j = 1, \ldots, k)$ denotes a maximal number of jobs which are processed consecutively on the single machine and no conjunction exists between two consecutive jobs (i.e. a block if $m_j > 1$). Assume that a permutation π' with $C_{\max}(\pi') < C_{\max}(\pi)$ exists and all jobs of the blocks on P^π are processed in the same order as in π. Then the graph $G^{\pi'}$ contains a path which consists of the jobs of P^π in the same order as in π and which possibly contains some additional jobs in between. This means that we again have the arcs $0 \to u_1^1$, $u_{m_j}^j \to u_1^{j+1}$ for $j = 1, \ldots, k-1$ and $u_{m_k}^k \to *$ with the same arc lengths as in G^π. It remains to consider two arbitrary jobs $u_\lambda^j, u_{\lambda+1}^j$ which may be separated by some additional jobs $v_\lambda^j, \ldots, v_\mu^j$. By iteratively applying the triangle inequalities (6.1) $s_{uv} + p_v + s_{vw} \geq s_{uw}$, $l_{uv} + p_v + s_{vw} \geq s_{uw}$, $s_{uv} + p_v + l_{vw} \geq s_{uw}$ we obtain that the length of the path between u_λ^j and $u_{\lambda+1}^j$ in $G^{\pi'}$ is not smaller than the arc length $d^\pi_{u_\lambda^j u_{\lambda+1}^j} = p_{u_\lambda^j} + s_{u_\lambda^j u_{\lambda+1}^j}$ in G^π. Thus, we have $C_{\max}(\pi') \geq C_{\max}(\pi)$, which is a contradiction. □

Contrary to the job-shop problem the interchange of internal jobs in a block may improve the current objective value due to the setup times. Based on the theorem which gives necessary conditions for a permutation π' to be better than π, we introduce the following neighborhood $\mathcal{N}_1$.

Neighborhood $\mathcal{N}_1$: Let π be an arbitrary permutation and let P^π be a critical path associated with π. Neighborhood $\mathcal{N}_1(\pi)$ contains all feasible permutations π' (compatible with the precedence relations) which can be constructed by interchanging two adjacent jobs in a block on the critical path P^π.

Simple examples show that neighborhood $\mathcal{N}_1$ is not connected, i.e. it is not possible to transform two arbitrary permutations into each other by iteratively applying only changes according to $\mathcal{N}_1$. But the neighborhood $\mathcal{N}_1$ is opt-connected, which means that from each permutation an optimal solution can be reached by a finite sequence of moves in the neighborhood. This can be shown by an adaption of the proof in van Laarhoven et al. [1992] who considered $\mathcal{N}_1$ in connection with the job-shop problem. Furthermore, since due to the definition of a block between two adjacent jobs no precedence relation exists, an interchange of two adjacent jobs in a block always results in a feasible permutation π'.

A disadvantage of neighborhood $\mathcal{N}_1$ is that permutations are changed only slightly by the interchange of two adjacent jobs in a block. Thus, many interchanges may be necessary to change the structure of a given solution significantly. To decrease this number of moves, we extend $\mathcal{N}_1$ in the following way.

Neighborhood $\mathcal{N}_2$: Let π be an arbitrary permutation and let P^π be a critical path associated with π. Neighborhood $\mathcal{N}_2(\pi)$ contains all feasible permutations π' which can be constructed as follows:

- two adjacent jobs of a block on P^π are interchanged,
- in π' the first job of a block B on P^π is shifted to the right to another position in B, or
- in π' one job of a block B on P^π different from the last one is shifted to the end of B.

Since neighborhood $\mathcal{N}_2$ contains $\mathcal{N}_1$ as a subneighborhood (i.e. $\mathcal{N}_1(\pi) \subseteq \mathcal{N}_2(\pi)$ for each permutation π), $\mathcal{N}_2$ is also opt-connected. The size of both neighborhoods is polynomially bounded by $O(n)$ because the number of jobs in blocks on a critical path is bounded by n.

Formally, the two neighborhoods can be defined by the following operators which may be applied to every block B on a critical path P^π:

- $api(k)$ for $k = 1, \ldots, |B|-2$ interchanges the k-th job in block B with its immediate successor,
- $rshift1(k)$ for $k = 2, \ldots, |B|-1$ shifts the first job of block B to the k-th position in the same block, and
- $endshift(k)$ for $k = 1, \ldots, |B|-1$ shifts the k-th job in block B to the last position in B.

Note that the operator $api(1)$ results in the same permutation as $rshift1(2)$, the operator $rshift1(|B|)$ is the same as $endshift(1)$ and $api(|B|-1)$ equals $endshift(|B|-1)$. Since within the following considerations the operators $api(|B|-1)$ and $rshift1(|B|)$ cannot be treated in the same way as the other operators of the same type, we included them into the set of the endshift operators where they can be dealt with in a unique context.

Local search methods like iterative improvement or tabu search (cf. Brucker [1998]) usually evaluate all solutions in the neighborhood of the current solution and choose the best one as a new solution. For a straightforward calculation of the best neighbor of a permutation we have to calculate the objective value for $O(n)$ neighbored permutations. Although a neighbored permutation differs only slightly from the current permutation, the starting times for all jobs which appear after the shifted job have to be recalculated to calculate the makespan of a neighbored solution. These starting times may be changed by different amounts due to the minimal time-lags (according to (8.1)). Thus, the straightforward calculation of a best neighbor will need $O(n(n+|C|))$ time since calculating the objective value of a permutation needs $O(n+|C|) = O(n^2)$ time (cf. Subsection 8.1).

In the following we will describe an approach which calculates a hopefully good (but not necessarily the best) neighbor in a shorter amount of time. The whole process of evaluating all neighbors for the operators $api(k)$ and $rshift1(k)$ will be done in $O(n+|C|)$ time using estimations for the real objective values. Then the $O(n+|C|)$-procedure for calculating the correct objective value has to be applied only once for the determined neighbor. Thus, we are able to evaluate **all** *api*- and $rshift1$-neighbors with the same time complexity which is needed for calculating the objective value of a **single** neighbor. This means that the computational time for evaluating all *api*- and $rshift1$-neighbors of a solution is reduced by the factor n. Unfortunately, for the operator $endshift(k)$ we are not able to evaluate all $endshift$-neighbors with worst-case complexity $O(n+|C|)$. By using similar estimations as for the other two operators, we propose an approach which evaluates all $endshift$-neighbors of a solution π with worst-case complexity $O(|C| + \sum\limits_{B \in P^\pi} |B|^2) = O(|C| + n^2)$, which is nevertheless better than $O(n|C| + n^2)$.

We will describe the used estimates for neighbored permutations in dependence of the operators applied to generate the neighbors. At first we consider the operator $api(k)$ with $k \in \{1, \ldots, |B|-2\}$ which interchanges two adjacent jobs in a block B on a critical path P^π. According to the definition of a block between such two jobs no precedence relation exists. Thus, all operators $api(k)$ lead to feasible permutations $\pi' = api(k)(\pi)$. To apply $api(k)$ to a block B, B has to contain at least 3 jobs. W.l.o.g. let $B := (i_b, ..., 1, 2, 3, ..., i_f)$ and let 1 be the k-th job of B. Then $\pi' = api(k)(\pi)$ has the form $\pi' = (\ldots, i_b, ..., 2, 1, 3, ..., i_f, \ldots)$.

To obtain an estimate of the objective value of π', we will use estimates for the starting times $S_j^{\pi'}$ of the jobs in π'. These estimates are calculated using the current starting times S_j^π and the lower bounds L_j^π for the starting times of the jobs in π based on the time-lags (cf. (8.2)). For the sake of simplicity in the following we will denote the starting times of the jobs j in π by $S_j := S_j^\pi$ and the lower bounds by $L_j := L_j^\pi$. Again, t_{ij} denotes the amount of time which has to be considered between the completion time of job i and the starting time of j (i.e. as in (6.2) either a time-lag l_{ij} if $i \to j \in C$ or a setup s_{ij}).

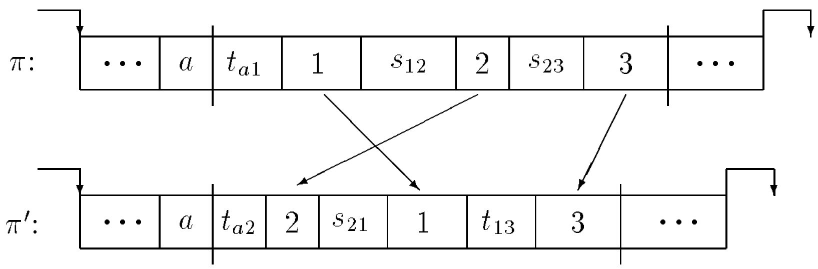


Figure 8.1: Permutations π and $\pi' = api(k)(\pi)$

In a first step we will consider the new starting times $S'_j := S_j^{\pi'}$ for the jobs $j \in \{1, 2, 3\}$ (cf. Figure 8.1). Let a denote the immediate predecessor of job 1 in π. Obviously, we have

$$S'_2 = \max\{S_a + p_a + t_{a2},\ L_2\} \tag{8.4}$$

since the L_2-value is not changed by the operator $api(k)$. For the same reason the new starting time of job 1 is given by

$$S'_1 = \max\{S'_2 + p_2 + s_{21},\ L_1\}. \tag{8.5}$$

The value L_3 may only be changed by the operator if the conjunction $1 \to 3 \in C$ exists. In this case L'_3 is given by

$$L'_3 = \max\left[\max_{\substack{\{j \mid j \neq 1;\\ j \to 3 \in C\}}} \{S_j + p_j + l_{j3}\},\ S'_1 + p_1 + l_{13}\right].$$

This expression equals $\max\{L_3,\ S'_1 + p_1 + l_{13}\}$ since next we will show that job 1 is never moved to the left (i.e. $S_1 \leq S'_1$ holds). Furthermore, due to the definition of blocks no conjunction $2 \to 3 \in C$ exists, i.e. we have

$$S'_3 = \max\{S'_1 + p_1 + t_{13},\ L_3\}. \tag{8.6}$$

Due to the weak triangle inequality for the setup times s_{ij} it is easy to show that the following inequalities hold:

$$\Delta_2 := S'_2 - S_2 \leq 0, \quad \Delta_1 := S'_1 - S_1 \geq 0, \quad \Delta_3 := S'_3 - S_3 \geq \Delta_2.$$

Thus, job 2 is not moved to the right and job 1 is not moved to the left. Furthermore, if job 3 is moved to the left, it cannot be moved by more than $|\Delta_2|$ units.

Since job 3 is critical w.r.t. π, we have $S_3 = \bar{S}_3^\pi$ and Δ_3 has the following meaning:

- If Δ_3 is non-negative, the sequence of critical jobs after job 3 has to be shifted to the right by at least Δ_3 time units to respect all time-lags l_{3j} and the machine capacity in π',
- if Δ_3 is negative, $|\Delta_3|$ defines an upper bound for the amount of time by which the whole sequence of jobs after job 3 can be shifted to the left (if we shift this sequence by $|\Delta_3|$ time units, all time-lags l_{3j} and the machine capacity are respected, but time-lags l_{ij} between jobs i, j may be violated where i is placed before job 3 and j is placed after job 3 in π).

To obtain an estimate of the new starting times of the jobs scheduled after job 3 we also estimate how the jobs j with $1 \to j \in C$ are influenced by the right-shift of job 1. Since such a job j cannot be started before time $S_1' + p_1 + l_{1j}$ in the schedule $S^{\pi'}$, the value $\theta_j := S_1' + p_1 + l_{1j} - \bar{S}_j^\pi$ has the following meaning:

- If θ_j is non-negative, some jobs after job j exist which have to be shifted to the right by at least θ_j time units to respect the time-lags l_{1j},
- if θ_j is negative, $|\theta_j|$ defines an upper bound for the amount of time by which the whole sequence of jobs scheduled after job j can be shifted to the left respecting only the time-lags l_{1j}.

Using the maximal θ_j-value of all conjunctive successors j of job 1,

$$\Delta_1^{succ} := \max_{\{j|1\to j\in C\}} \{\theta_j\} = \max_{\{j|1\to j\in C\}} \{S_1' + p_1 + l_{1j} - \bar{S}_j^\pi\} \tag{8.7}$$

and the value Δ_3 the following lemma can be proved:

Lemma 8.1: Let $\Delta := \max\{\Delta_3, \Delta_1^{succ}\}$. Then the objective value of the permutation $\pi' = api(k)(\pi)$ can be estimated as follows:

(a) If $\Delta \geq 0$ holds, we have $C_{\max}(\pi') = C_{\max}(\pi) + \Delta$, and

(b) if $\Delta < 0$ holds, we have $C_{\max}(\pi) \geq C_{\max}(\pi') \geq C_{\max}(\pi) - |\Delta|$.

Proof: Consider the schedule S' where the jobs j scheduled before the interchanged jobs start at their old starting times S_j, the starting times for the jobs $j \in \{1, 2, 3\}$ are given by (8.4) to (8.6), and the jobs j scheduled after job 3 start at time $S_j' = \bar{S}_j^\pi + \Delta$. Obviously, this schedule S' is feasible w.r.t. the setup times. Thus, it remains to consider the time-lags.

(a) $\Delta \geq 0$:
Since all jobs scheduled after job 3 are shifted by the same amount Δ to the right and $\Delta \geq \Delta_3$ holds, only time-lags between job 1 and the jobs scheduled after job 3 may be violated (job 1 is the only job which may have been shifted by more than Δ to the right). However, due to the definition of Δ_1^{succ} all these time-lags are respected. Thus, the schedule S' is a feasible schedule for sequence π' and has a makespan of length $C_{\max}(\pi) + \Delta$.

This schedule is also a best possible schedule for sequence π' since in the case $\Delta = \Delta_3$ job 3 remains a critical job and in the case $\Delta = \Delta_1^{succ}$ job 1 followed by job j determining the maximum in (8.7) are part of a critical path.

(b) $\Delta < 0$:
Since $\Delta \geq \Delta_3$ and all time-lags l_{1j} are respected due to the definition of Δ_1^{succ}, only time-lags between jobs scheduled before job 3 and jobs scheduled after job 3 may be violated. If no such time-lag is violated, the schedule S' is a best possible schedule for sequence π' (same argumentation as above). On the other hand, if some time-lags l_{ij} are violated, the makespan of S' is a lower bound on the best possible makespan (using again the critical job or critical path argument from above).

An upper bound for the minimal makespan of π' can be achieved by considering the schedule S'', which is the same as S', but the jobs j scheduled after job 3 start at their latest starting times (i.e. $S''_j = \bar{S}_j^{\pi}$). The resulting schedule is feasible since all time-lags l_{ij} with $i \neq 2$ were already satisfied in $\bar{S}$ and none of the time-lags l_{2j} are violated due to $\Delta_2 \leq 0$.

Summarizing, we get $C_{\max}(\pi) \geq C_{\max}(\pi') \geq C_{\max}(\pi) - |\Delta|$. □

From Lemma 8.1 the following conclusions can be drawn:

- In case (a) we know that π' does not improve π and the exact objective value of π' can be determined directly from the estimate Δ, and
- in case (b) we know that π' is not worse than π and the estimate Δ leads to an upper bound for the improvement.

The time complexity for the described process of evaluating all neighbors $api(k)$ for all blocks B and $k = 1, \ldots, |B| - 2$ can be determined as follows. Obviously, for each of the $|B| - 2$ neighbors for a block B we can calculate the three starting times according to (8.4) to (8.6) in constant time. Since during the calculation of all Δ_1^{succ}-values each conjunction in C is considered at most once and all blocks together contain at most n jobs, the whole process of evaluating all api-neighbors for all blocks can be done in $O(n + |C|)$ time.

Next we consider the operator $rshift1(k)$ with $k \in \{2, \ldots, |B| - 1\}$ which shifts the first job of a block B on a critical path P^{π} to the right. To apply $rshift1(k)$ to a block B, B has to contain at least 3 jobs. W.l.o.g. let $B := (1, 2, 3, ..., f)$. Then $\pi^{(k)} = rshift1(k)(\pi)$ is feasible if no conjunction $1 \rightarrow j \in C$ with $j \in \{3, \ldots, k\}$ exists and $\pi^{(k)}$ has the form $\pi^{(k)} = (\ldots, 2, ..., k, 1, k+1, ..., f, \ldots)$. Note that if a conjunction $1 \rightarrow h \in C$ with $h \in B$ exists, all permutations $\pi^{(k)} = rshift1(k)(\pi)$ for $k = h, \ldots, f$ are infeasible. Thus, we can stop evaluating the $rshift1$-operators for a block as soon as we detect the first conjunction $1 \rightarrow h \in C$ with $h \in B$.

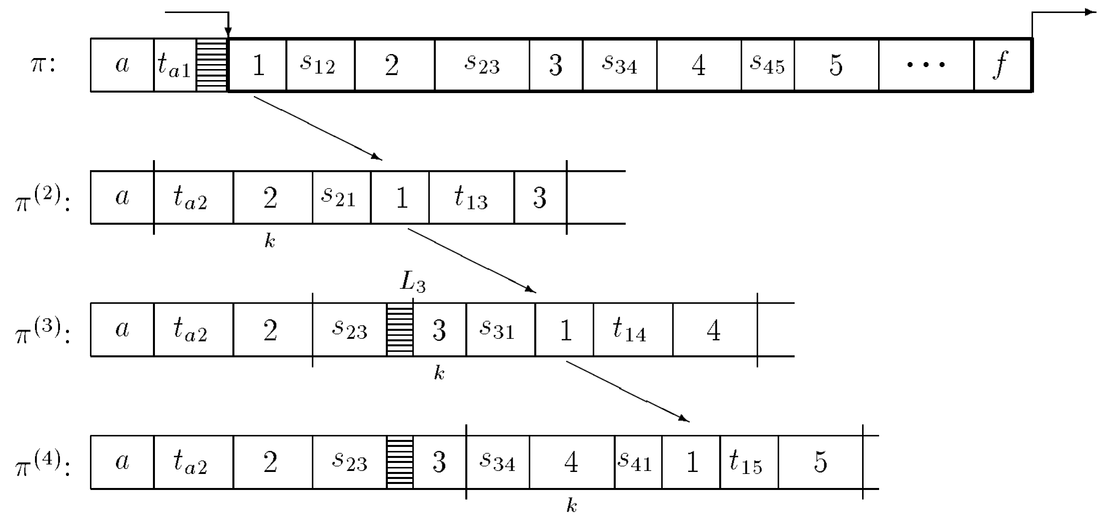


Figure 8.2: Permutations π and $\pi^{(k)} = rshift1(k)(\pi)$

To obtain an estimate of the makespan of $\pi^{(k)}$, we will again calculate the correct new starting times for the jobs involved in the shift and use these values to estimate the starting times of the remaining jobs. The new starting times $S_j^{(k)} := S_j^{\pi^{(k)}}$ for the jobs $j \in \{1, \ldots, k+1\}$ (see Figure 8.2) can be calculated as follows. Since the operator $rshift1(2)$ equals $api(1)$, for $k = 2$ the starting times $S_j^{(k)}$ with $j \in \{1, 2, 3\}$ are given by (8.4) to (8.6), i.e.

$$S_2^{(2)} = \max\{S_a + p_a + t_{a2},\ L_2\} \tag{8.8}$$

$$S_1^{(2)} = \max\{S_2^{(2)} + p_2 + s_{21},\ L_1\} \tag{8.9}$$

$$S_3^{(2)} = \max\{S_1^{(2)} + p_1 + t_{13},\ L_3^{(2)}\}, \tag{8.10}$$

where a denotes the direct predecessor of job 1 in π and the new value $L_3^{(2)}$ may again be replaced by the old value L_3.

For $k = 3, \ldots, |B| - 1$ we proceed as follows. By comparing $\pi^{(k-1)}$ and $\pi^{(k)}$ we see that the starting times for the jobs $j = 2, \ldots, k-1$ placed before job k are the same (i.e. $S_j^{(k)} = S_j^{(k-1)}$). Thus, only the three new starting times $S_k^{(k)}, S_1^{(k)}, S_{k+1}^{(k)}$ have to be calculated for $\pi^{(k)}$. They are given by

$$S_k^{(k)} = \max\{S_{k-1}^{(k)} + p_{k-1} + s_{k-1,k},\ L_k^{(k-1)}\} \tag{8.11}$$

$$S_1^{(k)} = \max\{S_k^{(k)} + p_k + s_{k1},\ L_1\} \tag{8.12}$$

$$S_{k+1}^{(k)} = \max\{S_1^{(k)} + p_1 + t_{1,k+1},\ L_{k+1}^{(k)}\}, \tag{8.13}$$

where the values $L^{(\nu)}_{\nu+1}$ in (8.11) and (8.13) denote the lower bounds for the starting time of job $\nu+1$ in $\pi^{(\nu)}$ w.r.t. the time-lags $l_{j,\nu+1}$ with $j \neq 1$, i.e.

$$L^{(\nu)}_{\nu+1} = \max_{\{j\neq 1 | j\to\nu+1\in C\}} \{S^{(\nu)}_j + p_j + l_{j,\nu+1}\}.$$

Note that contrary to the *api*-operator these lower bound values may not be replaced by the old values.

Due to the weak triangle inequality the following inequalities hold:

$$\Delta^{(k)}_k := S^{(k)}_k - S_k \leq 0, \quad \Delta^{(k)}_1 := S^{(k)}_1 - S_1 \geq 0, \quad \Delta^{(k)}_{k+1} := S^{(k)}_{k+1} - S_{k+1} \geq \Delta^{(k)}_k.$$

Thus, jobs $2,\ldots,k$ are not moved to the right and job 1 is not moved to the left. Furthermore, if job $k+1$ is moved to the left, it cannot be moved by more than $|\Delta^{(k)}_k|$ time units.

The lower bound values $L^{(k-1)}_k, L^{(k)}_{k+1}$ in (8.11) and (8.13) do not always have to be calculated from scratch for the new starting times $S^{(k)}_j$ in iteration k. The value $L^{(k-1)}_k$ is known from the previous iteration $k-1$ and the calculation of $L^{(k)}_{k+1}$ in (8.13) can be done as follows. If no job $2,\ldots,k-1$ defines the value L_{k+1} in π, (i.e. $S_j + p_j + l_{j,k+1} < L_{k+1}$ for $j = 2,\ldots,k-1$), then the L_{k+1}-value does not change and we have $L^{(k)}_{k+1} := L_{k+1}$, because jobs $2,\ldots,k-1$ are not moved to the right. Otherwise, if a job $j \in \{2,\ldots,k-1\}$ determines the L_{k+1}-value in π, (i.e. $S_j + p_j + l_{j,k+1} = L_{k+1}$), the L_{k+1}-value may be decreased if j is shifted to the left. Then we have to recalculate L_{k+1} according to the new starting times of the jobs $2,\ldots,k-1$ and the old starting times of the remaining jobs:

$$L^{(k)}_{k+1} := \max\left[\max_{\substack{\{j|j=2,\ldots,k-1;\\ j\to k+1\in C\}}} \{S^{(k)}_j + p_j + l_{j,k+1}\}, \max_{\substack{\{j|j\neq 2,\ldots,k-1;\\ j\to k+1\in C\}}} \{S_j + p_j + l_{j,k+1}\}\right]. \quad (8.14)$$

To estimate the change of the objective value we again use a measure for the influence of this move on the starting times of the jobs scheduled after job 1 in addition to the amount $\Delta^{(k)}_1$ by which job 1 is moved to the right:

$$\Delta^{(k)\,succ}_1 := \max_{\{j|1\to j\in C\}} \{S^{(k)}_1 + p_1 + l_{1j} - \bar{S}^{\pi}_j\}. \quad (8.15)$$

Analogously to Lemma 8.1 the following lemma can be proved:

Lemma 8.2: Let $\Delta := \max\{\Delta^{(k)}_{k+1}, \Delta^{(k)\,succ}_1\}$. Then the objective value of the permutation $\pi^{(k)} = rshift1(k)(\pi)$ can be estimated as follows:

(a) If $\Delta \geq 0$ holds, we have $C_{\max}(\pi') = C_{\max}(\pi) + \Delta$, and

(b) if $\Delta < 0$ holds, we have $C_{\max}(\pi) \geq C_{\max}(\pi') \geq C_{\max}(\pi) - |\Delta|$. □

Note that if $\Delta_1^{(k)\,succ} \geq 0$ holds for a permutation $\pi^{(k)}$, we also have $\Delta_1^{(h)\,succ} \geq \Delta_1^{(k)\,succ} \geq 0$ for all $h = k+1, \ldots, |B|-1$ since in these iterations the starting time of job 1 is not smaller than $S_1^{(k)}$. Thus, if we are only interested in a neighbor with a minimal Δ-value, we do not have to consider the remaining shifts for the current block.

The time complexity for evaluating all $rshift1$-neighbors can be determined as follows. If we evaluate the neighbors $\pi^{(k)}$ iteratively for all blocks B and $k = 2, 3, \ldots, |B|-1$, only three new starting times have to be determined for $\pi^{(k)}$, as mentioned above. Each of these calculations can be done in constant time if the $L_{k+1}^{(k)}$-values are available. The calculation of all relevant $L_{k+1}^{(k)}$-values according to (8.14) and all $\Delta_1^{(k)\,succ}$-values according to (8.15) can be done in $O(|C|)$ time since each conjunction has to be considered at most once. Hence, the whole process of evaluating all $rshift1$-neighbors for all blocks requires $O(n + |C|)$ time.

Finally, we consider the operator $endshift(k)$ with $k \in \{1, \ldots, |B|-1\}$ which shifts the k-th job of a block B on a critical path P^π to the end. Since a block contains at least 2 jobs by definition, $endshift(k)$ can be applied to any block B. W.l.o.g. let $B := (1, 2, ..., f)$, let b be the conjunctive successor of job f on P^π, and let c be the job directly processed after job f in π (see Figure 8.3 where $f = 5$). Then $\pi^{(k)} = endshift(k)(\pi)$ is feasible if no conjunction $k \to j \in C$ with $j \in \{k+2, \ldots, f\}$ exists and $\pi^{(k)}$ has the form

$$\pi^{(k)} = (\ldots, 1, ..., k-1, k+1, ..., f, k, c, \ldots, b, \ldots).$$

As for the previously considered operators, it can be shown that jobs $k+1, \ldots, f$ are not moved to the right and job k is not moved to the left.

At first, we only calculate the new starting times $S_j^{(k)} := S_j^{\pi^{(k)}}$ for the moved job k, the final job f of the considered block and its machine successor c. For $k = |B|-1, \ldots, 1$ we determine the new starting time of the final job f by calculating the amount of time by which it can be shifted to the left. Due to the removal of job k between jobs $k-1$ and $k+1$, the jobs after $k+1$ can be moved by at most $s_{k-1,k} + p_k + s_{k,k+1} - t_{k-1,k+1}$ time units to the left respecting the setup times and the machine capacity. On the other hand, the maximal amount of time by which the whole sequence $(k+1, \ldots, f)$ can be shifted to the left according to the time-lags is given by

$$V_{k+1}^f := \min_{h=k+1}^{f} \{S_h - L_h^{k-1}\}, \tag{8.16}$$

where L_h^{k-1} is a lower bound for the starting time of job h in $\pi^{(k)}$ w.r.t. the time-lags l_{jh} with $j \neq k+1, \ldots, h-2$, i.e. $L_h^{k-1} = \max\limits_{\substack{\{j \mid j \neq k+1, \ldots, h-2; \\ j \to h \in C\}}} \{S_j + p_j + l_{jh}\}$. Note that the other time-lags l_{jh} with $j = k+1, \ldots, h-2$ do not have to be considered for moving the whole sequence $(k+1, \ldots, f)$ to the left. Thus, the new starting time of job f is given by

$$S_f^{(k)} = S_f - \min\{V_{k+1}^f, s_{k-1,k} + p_k + s_{k,k+1} - t_{k-1,k+1}\}. \tag{8.17}$$

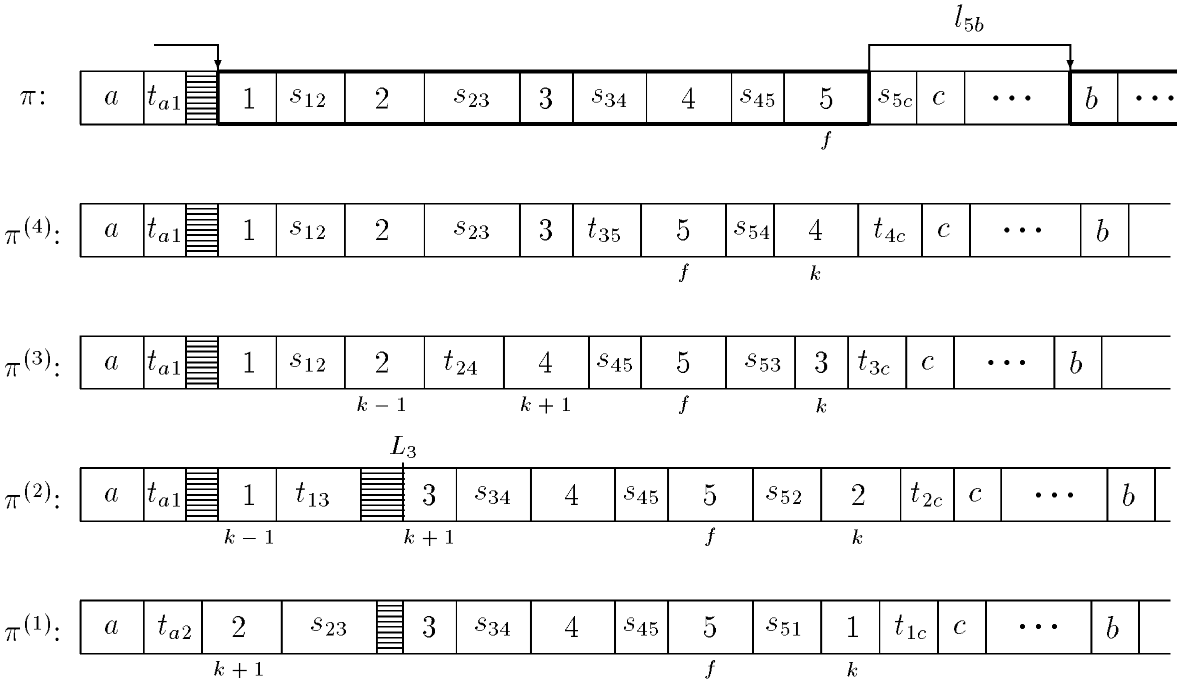


Figure 8.3: Permutations π and $\pi^{(k)} = endshift(k)(\pi)$

Since we assume that permutations π and $\pi^{(k)}$ are feasible, neither a conjunction $j \to k \in C$ nor a conjunction $k \to j \in C$ exists with $j \in \{k+2, \ldots, f\}$. Therefore, the new starting time of job k is only determined by the job directly processed before k on the machine, i.e.

$$S_k^{(k)} = S_f^{(k)} + p_f + s_{fk}. \tag{8.18}$$

Finally, the starting time of the new machine successor c of k is given by

$$S_c^{(k)} = \max \{S_k^{(k)} + p_k + t_{kc},\ L_c^{(k)}\}, \tag{8.19}$$

where the value $L_c^{(k)}$ in (8.19) again denotes a lower bound for the starting time of job c in $\pi^{(k)}$ w.r.t. the time-lags l_{jc} with $j \neq k$, i.e. $L_c^{(k)} = \max\limits_{\{j \neq k | j \to c \in C\}} \{S_j^{(k)} + p_j + l_{jc}\}$.

The calculation of the $L_c^{(k)}$-value can be done very similarly to the case of the $rshift1$-operator: If no job $k+1, \ldots, f$ defines the value L_c in π, (i.e. $S_j + p_j + l_{jc} < L_c$ for $j = k+1, \ldots, f$), then the L_c-value does not change and we have $L_c^{(k)} := L_c$, because jobs $k+1, \ldots, f$ are not moved to the right. Otherwise, if a job $j \in \{k+1, \ldots, f\}$ determines the L_c-value in π, (i.e. $S_j + p_j + l_{jc} = L_c$), the L_c-value may be decreased if j is shifted to the left. Then we have to recalculate L_c according to the new starting times of the jobs $k+1, \ldots, f$ and the old starting times of the remaining jobs:

$$L_c^{(k)} := \max \left[\max_{\substack{\{j | j = k+1, \ldots, f;\\ j \to c \in C\}}} \{S_j^{(k)} + p_j + l_{jc}\}, \max_{\substack{\{j | j \neq k+1, \ldots, f;\\ j \to c \in C\}}} \{S_j + p_j + l_{jc}\} \right]. \tag{8.20}$$

Unfortunately, at this point we may need the new starting times $S_j^{(k)}$ for some of the jobs $j = k+1, \ldots, f-1$ (we did not have to calculate them in (8.17) to (8.19)). They are given by

$$S_{k+1}^{(k)} = \max\{S_{k-1} + p_{k-1} + t_{k-1,k+1},\ L_{k+1}\} \tag{8.21}$$
$$S_j^{(k)} = \max\{S_{j-1}^{(k)} + p_{j-1} + s_{j-1,j},\ L_j^{k-1}\} \text{ for } j = k+2, \ldots, f-1, \tag{8.22}$$

with the same lower bound values L_j^{k-1} for $j = k+2, \ldots, f-1$ as in (8.16).

Again, we define a measure for the possible movements of jobs scheduled after job k, which is given by

$$\Delta_k^{(k)\,succ} := \max_{\{j \mid k \to j \in C\}} \{S_k^{(k)} + p_k + l_{kj} - \bar{S}_j^\pi\}.$$

Furthermore, we define measures for the movements of

- the machine successor c of f by $\Delta_c^{(k)} := S_c^{(k)} - \bar{S}_c$, and
- the conjunctive successor b of f on the critical path by $\Delta_b^{(k)} := S_f^{(k)} + p_f + l_{fb} - S_b$.

Analogously to the preceding lemmata the following lemma can be proved:

Lemma 8.3: Let $\Delta := \max\{\Delta_c^{(k)}, \Delta_b^{(k)}, \Delta_k^{(k)\,succ}\}$. Then the objective value of the permutation $\pi^{(k)} = endshift(k)(\pi)$ can be estimated as follows:

(a) If $\Delta \geq 0$ holds, we have $C_{\max}(\pi') = C_{\max}(\pi) + \Delta$, and

(b) if $\Delta < 0$ holds, we have $C_{\max}(\pi) \geq C_{\max}(\pi') \geq C_{\max}(\pi) - |\Delta|$. □

Contrary to the *api*- and *rshift*1-operators in the process of evaluating an $endshift(k)$-neighbor no information from the evaluation of the previous neighbor $endshift(k+1)$ can be used. Although all L_h^{k-1} -values in (8.16) can be determined in a preprocessing step in $O(n + |C|)$ time, calculating all minima in (8.16) for a block B requires $O(|B|^2)$ time. Furthermore, the lower bound values $L_c^{(k)}$ in (8.20) for a block B can only be calculated in $O(|B|^2 + |C|)$ time, since in each iteration k we may have to determine the new starting times $S_j^{(k)}$ in (8.22) for $j = k+1, \ldots, f-1$. Thus, the whole process for all blocks B on a critical path P^π needs $O(|C| + \sum_{B \in P^\pi} |B|^2) = O(|C| + n^2)$ time.

Summarizing, the conclusions of Lemmata 8.1, 8.2 and 8.3 can be used as follows in a local search algorithm. For a given permutation π we are able to determine an estimate Δ^* (the minimum of all considered Δ-values) and a solution $\pi' \in \mathcal{N}_2(\pi)$ by spending $O(n + |C|)$ time for all *api*- and *rshift*1-neighbors and $O(n^2 + |C|)$ time for all *endshift*-neighbors with the following meaning:

- If $\Delta^* \geq 0$ holds, we know that π' is the best neighbor of π in $\mathcal{N}_2(\pi)$ and that Δ^* is the amount by which π' is worse than π,

- if $\Delta^* < 0$ holds, we know that π' is not worse than π and that Δ^* defines an upper bound for the improvement of the best neighbor of π in $\mathcal{N}_2(\pi)$.

Based on these estimates a solution is chosen from the set of neighbors as a starting solution for the next iteration (usually a permutation π' for which its Δ-value equals Δ^*).

The neighborhoods introduced in this section and, thus, also the calculation of the presented estimates for the neighbors depend on the chosen critical path P^π for the solution S^π associated with the current solution π. However, in general for a given permutation π many critical paths P^π may exist. In the following we discuss some aspects about choosing a critical path P^π. We will show that we can always choose a critical path where all jobs on it belong to blocks or a path without any blocks.

Lemma 8.4: For each permutation π a critical path $P^\pi = (0 = i_0, i_1, \ldots, i_k, i_{k+1} = *)$ exists with the property that for no subsequence $(i_{\lambda-1}, i_\lambda, i_{\lambda+1})$ of P^π both conjunctions $i_{\lambda-1} \to i_\lambda \in C$ and $i_\lambda \to i_{\lambda+1} \in C$ exist.

Proof: Let $P^\pi = (0 = i_0, i_1, \ldots, i_k, i_{k+1} = *)$ be a critical path with $i_{\lambda-1} \to i_\lambda \in C$ and $i_\lambda \to i_{\lambda+1} \in C$ for an index $\lambda \in \{1, \ldots, k\}$. Then also the arc $i_{\lambda-1} \to i_{\lambda+1} \in C$ exists and $l_{i_{\lambda-1} i_{\lambda+1}} \geq l_{i_{\lambda-1} i_\lambda} + p_{i_\lambda} + l_{i_\lambda i_{\lambda+1}}$ holds, since we assume that the l_{ij}-distances are transitively adjusted. Since P^π is a longest path, we must have equality $l_{i_{\lambda-1} i_{\lambda+1}} = l_{i_{\lambda-1} i_\lambda} + p_{i_\lambda} + l_{i_\lambda i_{\lambda+1}}$. Thus, we may replace P^π by $(i_0, i_1, \ldots, i_{\lambda-1}, i_{\lambda+1}, \ldots, i_k, i_{k+1})$. Repeating this process yields a critical path with the requested property. $\square$

Note that if a critical path P^π without any blocks exists, we have $P^\pi = (0, *)$ and the permutation π is optimal, since $C_{\max}(\pi) = l_{0*}$ equals a trivial lower bound for the optimal makespan. Thus, we may assume that all jobs on the chosen critical path belong to some block.

The following lemma gives another possible restriction on the critical paths. It can be proved in a similar way as Lemma 8.4.

Lemma 8.5: For each permutation π a critical path $P^\pi = (0 = i_0, i_1, \ldots, i_k, i_{k+1} = *)$ exists with the property that for each pair of jobs i, j which belong to the same block and which are linked by a conjunction $i \to j \in C$ the inequality $S_i^\pi + p_i + l_{ij} < S_j^\pi$ holds. $\square$

As a consequence of this lemma we may assume that no starting time of a job in a block is determined by a conjunctive predecessor in the same block. If this situation were to occur (i.e. in a block $S_i^\pi + p_i + l_{ij} = S_j^\pi$ holds for a conjunction $i \to j \in C$), all changes in the subsequence between i and j would not improve $C_{\max}(\pi)$ since the length of the critical path containing the arc $i \xrightarrow{l_{ij}} j \in C$ is not shortened. Thus, by choosing the critical path according to Lemma 8.5, some non-improving solutions are not considered as candidates for neighbored permutations.

After having calculated the starting times S_j^π for a permutation π according to (8.1), a special critical path fulfilling the properties of Lemmata 8.4 and 8.5 denoted by CP^π can be determined by the following backward calculation: starting with $j := *$ in each step among the jobs i with $S_i^\pi + d_{ij}^\pi = S_j^\pi$ the job with the smallest starting time S_i^π is chosen. After adding the arc (i, j) to CP^π the process is repeated with $j := i$ in the same way until the starting job $i = 0$ is reached.

CP^π has the property that for all jobs $j \in \{i_{b+1}, \ldots, i_{f-1}\}$ of any block $(i_b, \ldots, i_f)$ on CP^π no job i exists with $S_i^\pi + p_i + l_{ij} = S_j^\pi$. This implies $S_j^\pi > L_j^\pi$, i.e. the starting times of all jobs different to the first in any block are determined by their machine predecessors. Thus, if we use CP^π to define the neighborhoods, some calculations can be simplified.

8.4 A tabu search algorithm

In this subsection we describe a tabu search algorithm for problem $1 \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$ which is based on the concepts of the previous subsection.

The **tabu search** method is a metaheuristic approach which was designed by Glover [1989, 1990]. In each iteration of this local search method the current solution is usually replaced by the best solution in its neighborhood. Due to the fact that (contrary to the iterative improvement method) also non-improving solutions are accepted, it is possible to leave local minima during the search process. But on the other hand, solutions may be visited more than once and therefore it may be possible that the method cycles. One strategy to avoid this problem would be to store all visited solutions in a so-called **tabu list** TL and to accept a neighbor of the current solution only if it is not contained in this list. Due to memory restrictions in general it is not possible to store all visited solutions. Therefore, the list contains only solutions which have been visited in the last q iterations. Then only cycles with a length greater than the tabu list length $|TL|$ may occur and if $|TL|$ is sufficiently large, the probability of cycling becomes very small. Furthermore, not complete descriptions, but only typical properties (attributes) of the solutions are stored in the list. All solutions having one of the properties stored in the tabu list are declared as tabu. The properties are chosen such that it is guaranteed that a visited solution will not be reached again as long as the corresponding property is in the tabu list. A disadvantage of this procedure is that solutions which have never been visited may be also forbidden by the tabu list. To overcome this difficulty so-called **aspiration criteria** are used which allow the acceptance of neighbors even if they are forbidden due to the tabu status. For example, solutions which improve the best solution found so far should always be accepted.

Besides the tabu list, which has the function of a short-term memory, often also a long-term memory is kept which is used for **diversification**. In this long-term memory promising solutions or properties of promising solutions which are visited during the search process are stored. If within the search process the best solution found so far is not improved in a certain number of iterations, the search process is stopped and restarted with a new starting solution (diversification). Since a restart from a randomly generated solution would neglect all information of the previous search process, the long-term memory is used to restart in a hopefully good region. Either a promising solution from the long-term memory list is selected or, in the case where only properties have been stored, a solution having the properties in the long-term memory is generated. The whole tabu search procedure stops if a given time-limit is exceeded or a maximal number of restarts has been completed.

In the following we will describe how these general concepts are applied in our tabu search algorithm for problem $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$. We use the neighborhood $\mathcal{N}_2$ defined by the operators $api(k)$, $rshift1(k)$, and $endshift(k)$. The properties stored in the tabu list depend on the current solution and the operator which will be applied to it.

More precisely, if an operator is applied to a block $B := (i_1, i_2 \ldots, i_{k-1}, i_k, i_{k+1}, \ldots, i_f)$ on a critical path P^π associated with a permutation π, besides the objective value $C_{\max}(\pi)$ we store a pair of jobs as an attribute, which characterizes the order in π before applying the operator:

- for the operator $api(k)$ we use the pair of interchanged jobs as an attribute, i.e. we store (i_k, i_{k+1}),
- for the operator $rshift1(k)$ we use the shifted job i_1 and its new predecessor (an old successor) i_k as an attribute, i.e. we store (i_1, i_k), and
- for the operator $endshift(k)$ we use the shifted job i_k and its old immediate successor i_{k+1} as an attribute, i.e. we store (i_k, i_{k+1}).

A neighbored solution $\pi' \in \mathcal{N}_2(\pi)$ of a permutation π is defined as tabu if π' results from π by reconstructing the order of a pair of jobs belonging to the tabu list. More specifically, π' is tabu if

- $\pi' = api(k)(\pi)$ and the pair (i_{k+1}, i_k) of interchanged jobs is contained in the tabu list,
- $\pi' = rshift1(k)(\pi)$ and a pair (i_λ, i_1) with $\lambda \in \{2, \ldots, k\}$ is contained in the tabu list, or
- $\pi' = endshift(k)(\pi)$ and a pair (i_λ, i_k) with $\lambda \in \{k+1, \ldots, f\}$ is contained in the tabu list.

If the tabu list contains a large number of solutions, it may be time-consuming to test whether a solution is tabu or not. Therefore, instead of a tabu list we use a tabu matrix TM of size $n \times n$. If an operator is applied to a permutation π in iteration h, we store both the objective value $C_{\max}(\pi)$ and the number h in the entry belonging to the corresponding pair of jobs (i, j), i.e. we set $TM(i, j) := (C_{\max}(\pi), h)$. The matrix representation may use more memory (n^2) than the tabu list, but the test whether a solution is tabu or not can always be done in time linear in the number of jobs interchanged by the operator. Applying an operator to a permutation π in iteration h yielding the new permutation π' is allowed if it is not tabu or an aspiration criterion is satisfied. An operator

- is tabu if for at least one pair (i, j) corresponding to the operator the entry $\hat{h}$ in $TM(i, j) = (C_{\max}(\hat{\pi}), \hat{h})$ satisfies the condition $h - \hat{h} \leq |TL|$,
- satisfies the aspiration criterion if the objective value of the new solution is smaller than the objective values of all entries in the tabu matrix which declare the new solution tabu (i.e. if for all entries $TM(i, j) = (C_{\max}(\hat{\pi}), \hat{h})$ of pairs (i, j) corresponding to the operator $C_{\max}(\pi') < C_{\max}(\hat{\pi})$ holds whenever $h - \hat{h} \leq |TL|$).

Since we do not calculate the correct objective values for all neighbored solutions but only estimates, we cannot apply the above approach directly. However, a straightforward adaption is to also test the aspiration criteria based on the estimates. If Δ denotes the estimate for the operator we want to apply to π, instead of using the correct objective value $C_{\max}(\pi')$ we test the condition $C_{\max}(\pi) + \Delta < C_{\max}(\hat{\pi})$. If $\Delta > 0$ holds, the use of the estimate does not change the situation since in this case the estimate equals the correct objective value. In the other case, where $\Delta \leq 0$, using the estimate may lead to another decision than using the correct value since the estimate only defines an upper bound for the improvement.

Following Dell' Amico & Trubian [1993] we use a variable length $|TL|$ for the tabu list (a so-called dynamic tabu list) according to the following rules:

- If in some iteration a solution π is found which improves the best solution found so far, we empty the tabu list (by defining $|TL| := 0$) since we know that this solution has never been visited before.
- If we are in an improving phase of the search (i.e. the objective value of the current solution is better than the value of the previous iteration) and the length of the tabu list is greater than a certain constant $TL_{\min}$, we decrease the length by one.
- If we are not in an improving phase of the search and the length of the tabu list is smaller than a certain constant $TL_{\max}$, we increase the length by one.

In the process of choosing a neighbored solution $\pi' \in \mathcal{N}_2(\pi)$ of π two different strategies can be used:

- "best-fit": an allowed neighbor π' is chosen with minimal $C_{\max}(\pi')$-value (or with minimal estimate Δ),
- "first-fit": the first allowed neighbor π' is accepted with $C_{\max}(\pi') < C_{\max}(\pi)$ (or with $\Delta < 0$). If no such neighbor exists, the same solution as in the first strategy is chosen.

Furthermore, we partition the neighborhood $\mathcal{N}_2(\pi)$ into three different subsets $\mathcal{N}_2(\pi) = \mathcal{N}_2^{api}(\pi) \cup \mathcal{N}_2^{rshift1}(\pi) \cup \mathcal{N}_2^{endshift}(\pi)$ according to the three types of operators. Instead of evaluating all neighbors from the set $\mathcal{N}_2(\pi)$ in each iteration, we restrict the choice of the next neighbor to one of these subsets. In this way the evaluation of the neighborhood can be further accelerated and some additional strategies may be used to intensify or diversify the search process. We guide the choice of the next operator according to the folllowing three types of strategies:

- depending on the iteration number: the choice of the next operator type is only determined by the current number of iterations, e.g. we may always repeat the sequence $api, rshift1, api, endshift, api, \ldots$,
- depending on the tabu status: the operator type is changed when all neighbors belonging to the last operator type are tabu,
- depending on the objective values: the operator type is changed if the objective value is not improved for a certain number of iterations.

Further details and computational results for the described tabu search algorithm can be found in Section 10.3.

9 The calculation of heuristic solutions for problem $J, R1 \mid t_{jkl}, t'_{kl} \mid C_{\max}$

In this section we present how heuristic solutions can be calculated for the job-shop problem with transportation and empty moving times where only a single robot is available for all transportations. This problem is a special case of problem $J, MPR \mid prec, t_{jkl}, t'_{kl} \mid C_{\max}$ which was introduced in Subsection 4.1. We have to determine a complete selection defined by orders for all operations processed on the same machine (machine selection) and an order for all transport operations on the robot (robot selection).

As described in Subsection 4.1, this problem may also be formulated as a special case of the general-shop problem with sequence-dependent setup times and, thus, may be solved by the tabu search algorithm developed by Küsters [1997]. In this approach the robot is considered as an additional machine and the empty moving times correspond to setup times on the robot. The search space of the tabu search algorithm is the set of all complete selections. In each iteration a neighbored solution is generated either by moving an operation on a machine or by moving a transport operation on the robot to another position.

In this section we present another local search algorithm which tries to deal with the different situations on the machines and the robot more individually. This algorithm is organized in two stages. While in in the outer stage an operation on a machine is moved to another position, in the inner stage the robot is optimized according to the new machine selection. In order to obtain a good robot solution we use the single-machine tabu search procedure of the previous subsection as a subroutine.

9.1 A tabu search algorithm for problem $J, R1 \mid t_{jkl}, t'_{kl} \mid C_{\max}$

As described in Section 4.1, feasible solutions may be represented by complete selections $S = C \cup S_M \cup S_R$ in the disjunctive graph $G = (V, C \cup D_M \cup D_R)$, where all machine disjunctions D_M and all robot disjunctions D_R are fixed in such a way that the resulting graph $G(S) = (V, S)$ is acyclic. We consider the situation in which a complete selection S with makespan $C_{\max}(S)$ is given and we want to improve this solution. We will define neighborhoods based on a block-approach which combines the block-approach for the classical job-shop problem with our approach of Section 8.3 for the single-machine problem arising for the robot. For this purpose we define blocks as follows. Let P^S be a critical path in $G(S)$. A sequence $u_1, \ldots, u_k$ of at least two successive nodes (i.e. $k > 1$) on P^S is called

- a **machine-block** if the operations of the sequence are processed consecutively on the same machine, and enlarging the subsequence by one operation leads to a subsequence which does not fulfill this property,

- a **robot-block** if the sequence consists of transport operations consecutively processed on the robot without idle times, and enlarging the subsequence by one operation would violate this property.

Note that the definition of a robot-block is compatible with the definition of a block in Section 8.3 for the single-machine problem $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$ resulting for the robot from a fixed machine selection. The following theorem is the basis for defining suitable neighborhoods and can easily be proved by combining the corresponding proofs for the classical job-shop problem (cf. Brucker et al. [1994A]) and Theorem 8.1.

Theorem 9.1: Let S be a complete selection with makespan $C_{\max}(S)$ and let P^S be a critical path in $G(S)$. If another complete selection S' with $C_{\max}(S') < C_{\max}(S)$ exists, then in S'

- at least one operation of some machine-block B on P^S has to be processed before the first or after the last operation of B, or
- at least two transport operations of a robot-block on P^S are processed in the opposite order.

Based on this result we introduce the following neighborhood $\mathcal{N}_3$. Let $S = C \cup S_M \cup S_R$ be a complete selection and let P^S be a critical path in $G(S)$. **Neighborhood $\boldsymbol{\mathcal{N}_3}$** contains all feasible selections $S' = C \cup S'_M \cup S'_R$ which can be constructed by one of the following moves:

- in S'_M one operation (different from the first one) of a machine-block B on P^S is moved before all other operations in B, or
- in S'_M one operation (different from the last one) of a machine-block B on P^S is moved after all other operations in B, or
- in S'_R two adjacent transport operations of a robot-block on P^S are interchanged.

Using this neighborhood, in each iteration a neighbored solution S' is constructed in which either the machine selection S_M or the robot selection S_R is changed, but not both. Since the robot corresponds to a bottleneck machine, the old robot selection S_R is often a bad solution for the new machine selection S'_M or S_R is even infeasible w.r.t. S'_M (i.e. $G(S')$ contains a cycle). To overcome this disadvantage we may proceed in two stages, i.e. we define a neighborhood where after changing the machine selection the robot selection is adapted before the makespan of the new solution is determined. This **neighborhood $\boldsymbol{\mathcal{N}_4}$** contains all feasible selections $S' = C \cup S'_M \cup S'_R$ which can be constructed as follows:

- in S'_M one operation (different from the first one) of a machine-block B on P^S is moved before all other operations in B, or one operation (different from the last one) of a machine-block B on P^S is moved after all other operations in B, and
- S'_R is obtained from S_R by calculating a heuristic solution for the single-machine problem associated with the new machine selection S'_M.

While in the outer stage we change the machine selection by the same operators as in $\mathcal{N}_3$, in the inner stage we use a good version of the tabu search algorithm described in Section 8.4.

Neighborhood $\mathcal{N}_4$ is defined in a hierarchical way where first the machine selection is changed and then the robot selection is adapted. For instances with large transportation times it may happen that no machine-blocks exist for a considered solution S and, thus, neighborhood $\mathcal{N}_4(S)$ is empty. To improve such a solution S, according to Theorem 9.1 some robot-block has to be destroyed. Since in the outer stage of our hierarchical approach we only want to change the machine selection, but not the robot selection, we allow some additional moves on the level of the machine selections. For this purpose we do not only move critical operations belonging to machine-blocks, but also single operations on the chosen critical path. According to Theorem 9.1 interchanging a single critical operation (not belonging to a machine-block) with its machine predecessor or successor cannot reduce the objective value, but in combination with the change of the robot selection an improving solution may be obtained (since time-lags in the robot problem are cancelled by moving critical operations on the machines).

Thus, we enlarge $\mathcal{N}_4(S)$ by all feasible selections $S' = C \cup S'_M \cup S'_R$, where in S'_M a single critical operation on P^S is interchanged with its machine predecessor or successor, and S'_R is again a solution associated with the new machine selection S'_M. This resulting **neighborhood $\boldsymbol{\mathcal{N}_5}$** is used in our tabu search algorithm, which will be described in more detail next. As in the tabu search procedure from Section 8.3, we store attributes which characterize visited solutions during the search process in a tabu list. We use one kind of attributes suggested by Küsters [1997]: If in a complete selection $S' = C \cup S'_M \cup S'_R$ the machine selection S'_M results from S_M by moving an operation u, we store the objective value $C_{\max}(S)$ besides the triple consisting of u, its old machine predecessor and successor in S_M. A solution S is defined to be tabu if $C_{\max}(S)$ equals the objective value of an element in the tabu list and the associated triple of this element in the tabu list is reconstructed in S. Since the objective values are already taken into account in the attributes of the solutions, no additional aspiration criteria are used.

During the search process we use two versions of the single-machine tabu search algorithm for the robot. First we evaluate neighbored solutions in $\mathcal{N}_5(S)$ with a fast version. The resulting makespan is used to test whether the neighbored solution is tabu or not. Then we choose the best neighbored solution which is not tabu and try to improve the new robot selection within some more iterations of a second tabu search version.

In order to obtain starting solutions for problem $J, R1 \mid t_{jkl}, t'_{kl} \mid C_{\max}$ we use two versions of a priority-based heuristic described in Brucker et al. [1994A] for the classical job-shop problem. In the first version we only calculate a machine selection S_M and determine a corresponding robot selection S_R afterwards. In the second version we calculate a complete selection by considering the machines and the robot simultaneously.

In both versions of this priority-based heuristic in each iteration an operation is determined to be scheduled next from set A of all operations for which all predecessors are scheduled. Initially, A contains only the source 0 of the disjunctive graph. From the set A we choose an operation which is scheduled next as follows.

1. Let $u \in A$ be an operation with minimal value $r_u + p_u$, i.e. $r_u + p_u = \min_{v \in A}\{r_v + p_v\}$ and let M_k be the machine which has to process operation u. Define the set $\bar{A}$ by
$$\bar{A} := \{v \in A \mid r_v < r_u + p_u \text{ and } v \text{ has to be processed on } M_k\}.$$
2. Choose an operation $v \in \bar{A}$ according to some priority rule.
3. Update set A by inspecting all successors w of v. If all predecessors of w have already been scheduled, set $A := A \cup \{w\}$. Furthermore, scheduling operation v is taken into account in the heads of its successors and all unscheduled operations on M_k by setting $r_w := \max\{r_w, r_v + p_v\}$ for all these operations w. After this, delete v from the set A and proceed with Step 1.

In Step 2 we tested the following priority rules:

- MIN-HEAD: Choose an operation $v \in \bar{A}$ with minimal head r_v.
- MAX-TAIL: Choose an operation $v \in \bar{A}$ with maximal tail q_v.
- JPS: Choose an operation $v \in \bar{A}$ with a minimal lower bound value for the makespan assuming that v is scheduled next. The lower bound is based on the calculation of Jackson's preemptive schedule for problem $1 \mid pmtn, prec, r_j \mid \max\{C_j + q_j\}$ (Jackson [1955]) with the set $U \supseteq \bar{A}$ of all unscheduled operations on machine M_k, assuming that v is scheduled first.

In the first version we only calculate a machine selection S_M in this way and use the single-machine tabu search algorithm of Section 8.4 to obtain a good robot selection S_R for S_M afterwards. In the second version, instead of proceeding in two steps we calculate a complete selection $S = C \cup S_M \cup S_R$ by considering the robot as an additional machine (which has to process all transport operations). The empty moving times t'_{kl} corresponding to setup times s_{ij} on the robot (cf. Section 4.1) can be taken into account as follows. If in Step 3 a transport operation v is scheduled, we update the heads of all unscheduled transport operations w by $r_w := \max\{r_w, r_v + p_v + s_{vw}\}$, which is valid due to the triangle inequality for the setup times (cf. Section 8.2).

Computational results for the described tabu search algorithm can be found in Section 10.4.

10 Implementation and computational results

In this section we describe some implementational details of the proposed algorithms and report computational results for them. We implemented the algorithms in C and used the callable library of CPLEX 4.0 to solve the linear programs. The algorithms were tested on a Sun Ultra 2 work station (167 MHz) with operating system Solaris 2.5 and 320 MB general storage.
After describing the used test data in Subsection 10.1 we present results for the algorithms developed in Sections 7-9. First we tested the single-machine algorithms which calculate lower bounds and heuristic solutions for problem $1 \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$ in Subsections 10.2 and 10.3, respectively. Then in Subsection 10.4 we report results for the algorithms which calculate heuristic solutions for the job-shop problem with transportation times and a single robot.

10.1 Test data

Since no test instances for the job-shop problem with transportation times are available from the literature, we modified $m \times n$ benchmark problems for the classical job-shop problem, where m denotes the number of machines and n the number of jobs. We used the well-known 6×6 and 10×10 instances $P1$ and $P2$ from Muth & Thompson [1963]. In both instances the number of operations per job is equal to the number of machines (i.e. $n_j = m$ for $j = 1, \ldots, n$) and each job is processed on each machine exactly once. The processing times of the operations in the instance $P1$ are from the interval $[1, 10]$ and in $P2$ from the interval $[1, 100]$. Various test instances were obtained by adding transportation and empty moving times with different characteristics.

We distinguish the following four cases for the transportation times t_{jkl}:

- t_{jkl}: all transportation times are randomly generated from an interval $[1, T_{\max}]$ and then adjusted in such a way that the triangle inequality $t_{jkh} + t_{jhl} \geq t_{jkl}$ holds,
- t_{kl}: the transportation times are job-independent and randomly generated from an interval $[1, T_{\max}]$ and then adjusted in such a way that the triangle inequality $t_{kh} + t_{hl} \geq t_{kl}$ holds,
- $t_{jkl} = D|k - l|$: all transportation times are job-independent and proportional to the distance between the corresponding machines (which are assumed to be ordered in a single line),
- $t_{jkl} = T$: all transportation times are equal to a constant T.

Analogously we distinguish the following three cases for the empty moving times t'_{kl}:

- t'_{kl}: all empty moving times are randomly generated from the interval $[1, \min_{j=1}^{n} t_{jkl}]$ and then adjusted in such a way that the triangle inequality $t'_{kh} + t'_{hl} \geq t'_{kl}$ holds,
- $t'_{kl} = d|k-l|$: all empty moving times are proportional to the distance between the corresponding machines, where $d \leq D$,
- $t'_{kl} = t$: all empty moving times are equal to a constant $t \leq T$ (or $t \leq D$ if combined with the second case for the transportation times).

Test instances for the single-machine problem $1 \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$ were generated from the job-shop instances with transportation times. As described in Subsection 4.2, this problem arises for a single robot when a machine selection is given. In order to generate some instances where the optimal values are known, we calculated the optimal schedule for the job-shop problem with transportation times and a single robot where all empty moving times are zero. Since this problem is a classical job-shop problem (where the robot is considered as an additional machine), we were able to use the branch-and-bound algorithm of Brucker et al. [1994A] to calculate an optimal schedule. We then ignored the calculated order of transport operations for the robot and used only the complete machine selection for the m machines as input for the single-machine problem. In this way we obtained several instances with $6 \cdot 5 = 30$ or $10 \cdot 9 = 90$ transport operations (arising from the 6×6 and 10×10 job-shop instances $P1$ and $P2$, respectively). The maximal value for the randomly generated transportation times was set to $T_{\max} := 10$, while for the instances generated with the distance factor D the maximal value is $D(m-1)$. Since the processing times in instance $P2$ are very large (from the interval $[1, 100]$), the time horizon for the modified instances is often also very large ($C_{\max} \in [1000, 3000]$). Therefore, we also generated some instances in which the processing times are scaled by a factor $0 < f < 1$, i.e. we replace the processing times p_{ij} by $\lceil f \cdot p_{ij} \rceil$.

After some computational tests with a large test set we tried choosing interesting instances which are not easy to solve. In the following we will only report results for these instances. For the job-shop problem with transportation times we report results for 10 instances with $n = m = 6$ and 20 instances with $n = m = 10$. Concerning the single-machine problem we report results for 30 instances with 30 transport operations and 10 instances with 90 transport operations.

The names of the instances Pi_α_β with $i \in \{1,2\}$, $\alpha \in \{t_{jkl}, t_{kl}, Dx, Tx\}$, $\beta \in \{t'_{kl}, dx, tx\}$ and $x \in \mathbb{N}$ indicate the types of transportation and empty moving times as described above (e.g. $P1\_D3\_t2$ denotes the modified instance $P1$ where the transportation times satisfy $t_{jkl} = D|k-l|$ with $D = 3$ and the empty moving times are all equal to $t = 2$; $P1\_t_{jkl}\_t'_{kl}.y$ with $y \in \{1, \ldots, 8\}$ denotes instances with randomly generated transportation and empty moving times). If the processing times are scaled by a factor f, we write the factor behind the problem number $i \in \{1,2\}$, (e.g. $P2f0.2\_t_{kl}\_t'_{kl}$ denotes the modified $P2$-instance where the processing times are scaled with the factor $f = 0.2$ and the randomly generated transportation times are job-independent).

10.2 Lower bounds for $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$

As described in Section 7, we calculated lower bounds for problem $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$ in a destructive way. On the one hand, we used different immediate selection techniques and, on the other hand, we solved linear programs to detect infeasibility for a given threshold value T.

Concerning the implementation of the immediate selection procedures, the dynamic programming procedure of Subsection 6.2 is one of the most important subroutines which has to be implemented very efficiently due to its exponential worst-case complexity. In the following we briefly describe how the calculation of the $F(S, j)$-values for feasible subsets $S \subseteq V^0$ and jobs $j \in S$ is organized. During the dynamic programming procedure all triples $(S, j, F(S, j))$ with $F(S, j) < \infty$ are stored using hashing techniques. Additionally, a queue Q is held which contains all elements (S, j) with finite F-values exactly once during the calculations. Initially, all feasible elements $(\{j\}, j)$ are put into the queue. In each iteration we dequeue the first element (S, i) from Q and consider all feasible extensions $(S \cup \{j\}, j)$ with jobs $j \notin S$ whose predecessors are all contained in S. If $F(S, i) + p_i + t_{ij}$ is smaller than the current $F(S \cup \{j\}, j)$-value, the corresponding value is improved. If, additionally, the element $(S \cup \{j\}, j)$ is considered for the first time, it is put into the queue. In order to reduce the number of considered elements (S, i) we apply the following test. If during the consideration of an element (S, i) a job j with $F(S, i) + p_i + t_{ij} > d_j - p_j$ is detected, we may conclude that no feasible extension of (S, i) exists since in all other extensions $(S \cup \{h\}, h)$ job j cannot be started earlier than time $F(S, i) + p_i + t_{ij}$. In the case $t_{ij} = l_{ij}$ this is clear by the definition of the minimal time-lags. Otherwise, if $t_{ij} = s_{ij}$ holds, we have $F(S, i) + p_i + t_{ih} + p_h + t_{hj} \geq F(S, i) + p_i + s_{ij}$ due to the triangle inequalities (6.1). The case $l_{ih} + p_h + l_{hj} \geq s_{ij}$ does not occur since the time-lags are assumed to be transitively adjusted. Thus, in each extension $(S \cup \{h\}, h)$ job j violates its deadline and we may discard all these extensions of the current element. Some computational experiments showed that the number of considered elements and the running time were considerably reduced by this test. The dynamic programming algorithm terminates if the value $F(V^0, *)$ is calculated or if the queue becomes empty during the calculations. Then no feasible solution exists.

Concerning the implementation of the procedures based on linear programming, the decomposition of the precedence constraints into different subsets of chains is important. Since we only generated single-machine instances arising from job-shop problems with transportation times, we could use the decomposition into chains which is naturally induced by the job-shop structure (i.e. chain Γ_j consists of the $n_j - 1$ transport operations belonging to job $j \in \{1, \ldots, n\}$). In order to obtain a further decomposition for the q-chain-packing approach into λ subsets Λ_ϕ (each consisting of q chains), we consider the first q chains as the first subset, the second q chains as the second subset, etc. If n is not divisible by q, we partition the set into $\lfloor \frac{n}{q} \rfloor$ subsets consisting of q chains and one subset consisting of the remaining $n - q\lfloor \frac{n}{q} \rfloor$ chains.

Some difficulties arise due to the very large number of constraints in the chain-packing linear programming formulation ($O(n^2T)$ in the worst case). For instances with a large time horizon T it was not possible to include all these constraints into the linear programs (we included at most 15 000 rows). For example, for $q = 1$ (i.e. only single chains are considered), approximately 20 000 - 300 000 constraints exist for the instances with 30 transport operations. On the other hand, for $q = 6$ (i.e. all chains are considered simultaneously in a single subset), we only have 250-400 rows since in this case no disjunctive constraints (7.6) have to be considered in the linear programs.

Thus, for smaller values of q we decided to choose a subset of disjunctive constraints as follows. Each disjunction $i - j \in D$ may be considered in the constraints (7.6) in two different directions: on the one hand, as $(i,j) \in \vec{D}$ and, on the other hand, as $(j,i) \in \vec{D}$, where $\vec{D} := \{(i,j) \in V \times V \mid i - j \in D\}$. For each triple (i,j,t) with $(i,j) \in \vec{D}$ and $t \in [r_i + p_i, d_i]$ we determine the quotient $\frac{|I_t^{i,j}|}{d_j - (r_j + p_j)}$ indicating how many time periods in the time window of operation j are blocked by the corresponding disjunctive constraint. If this quotient exceeds a certain value and the maximal number of rows is not yet achieved, the corresponding disjunctive constraint is included into the linear program.

Further technical details and additional computational results for the different lower bound procedures will be found in Seibert [1999].

The lower bounds obtained by the different algorithms for the 30 single-machine test instances with 30 operations and the 10 chosen instances with 90 operations are presented in Table 10.1. This table contains the following information:

- Problem: the benchmark problem,
- $\sum t_{jkl}$: the sum of all transportation times of the transport operations,
- UB: the best known upper bound for the problem,
- JPS: the objective value of the relaxation $1 \mid pmtn, prec, r_j \mid \max\{C_j + q_j\}$ where the heads and tails are modified according to the time-lags,
- IS_2: the lower bound obtained by procedure `Immediate_Selection_2(T)`,
- HT_1, HT_2, HT_3: the lower bounds obtained by first applying for each threshold value T the procedure `Immediate_Selection_2(T)` and then the procedure `Improve_heads_tails_i(T)` for i=1,2,3, respectively,
- CP_q: the lower bounds obtained by first applying for each threshold T the procedure `Immediate_Selection_2(T)` and then the chain-packing bound procedure for a decomposition into subsets of q chains; if the lower bound value is equal to UB, it is marked with a *,

By testing some other combinations of the lower bound procedures we obtained the result that the procedure `Immediate_Selection_1(T)` always provided the same bound as `Immediate_Selection_2(T)`, and applying `Improve_heads_tails_3(T)` prior to calculating CP_6 did not improve the bounds.

Problem	$\sum t_{jkl}$	UB	JPS	IS_2	HT_1	HT_2	HT_3	CP_1	CP_2	CP_4	CP_6
$P1\_t_{jkl}\_t'_{kl}.1$	124	138	126	126	127	132	133	128	128	128	138 *
$P1\_t_{jkl}\_t'_{kl}.2$	118	131	121	121	122	123	124	121	121	122	131 *
$P1\_t_{jkl}\_t'_{kl}.3$	132	145	134	134	135	142	143	136	137	137	145 *
$P1\_t_{jkl}\_t'_{kl}.4$	96	111	100	100	102	108	108	102	102	102	109
$P1\_t_{jkl}\_t'_{kl}.5$	114	126	116	116	117	122	123	118	118	118	126 *
$P1\_t_{jkl}\_t'_{kl}.6$	119	132	121	121	122	123	125	122	122	122	131
$P1\_t_{jkl}\_t'_{kl}.7$	117	129	119	119	119	121	122	121	121	121	129 *
$P1\_t_{jkl}\_t'_{kl}.8$	123	136	126	126	126	130	131	127	129	129	136 *
$P1\_D1\_d1$	66	87	71	75	83	86	87	76	77	77	87 *
$P1\_D1\_t1$	66	82	71	71	79	81	81	73	74	74	81
$P1\_D2\_d1$	132	153	134	134	137	149	149	136	136	136	151
$P1\_D2\_d2$	132	169	134	134	146	158	158	136	136	138	164
$P1\_D2\_t1$	132	144	134	134	136	143	143	135	135	136	144 *
$P1\_D2\_t2$	132	153	134	134	138	150	152	136	136	136	152
$P1\_D3\_d1$	198	217	200	200	200	203	206	201	201	201	217 *
$P1\_D3\_d2$	198	233	200	200	205	216	222	202	202	202	230
$P1\_D3\_d3$	198	246	200	200	215	224	234	203	203	203	243
$P1\_D3\_t1$	198	210	200	200	200	200	200	201	201	201	210 *
$P1\_D3\_t2$	198	219	200	200	201	205	207	202	202	202	218
$P1\_D3\_t3$	198	227	200	200	203	214	216	203	203	203	227 *
$P1\_D5\_d5$	330	397	332	332	338	344	348	334	334	334	397 *
$P1\_t_{kl}\_t'_{kl}.1$	121	143	123	123	131	137	138	125	127	128	141
$P1\_t_{kl}\_t'_{kl}.2$	127	152	129	129	133	141	144	132	132	133	151
$P1\_t_{kl}\_t'_{kl}.3$	114	132	116	116	117	120	121	117	118	119	130
$P1\_T2\_t2$	60	82	63	67	76	79	80	68	68	73	81
$P1\_T3\_t1$	90	102	92	92	93	97	98	94	95	96	102 *
$P1\_T3\_t2$	90	110	92	92	96	105	107	95	96	96	110 *
$P1\_T3\_t3$	90	117	92	92	102	112	113	95	96	96	117 *
$P1\_T5\_t2$	150	168	152	152	152	154	156	154	154	155	168 *
$P1\_T5\_t5$	150	190	152	152	160	170	175	155	155	155	188
$P2\_D2\_d2$	446	1005	973	993	997	997	1002	—	—	—	—
$P2\_D3\_d1$	669	1149	1141	1145	1146	1146	1146	—	—	—	—
$P2\_D3\_d2$	669	1176	1141	1154	1165	1165	1165	—	—	—	—
$P2f0.1\_D1\_d1$	223	336	262	262	277	287	288	—	—	—	—
$P2f0.2\_D1\_d1$	223	370	313	313	336	336	334	—	—	—	—
$P2f0.5\_D1\_d1$	223	577	567	571	573	573	573	—	—	—	—
$P2f0.2\_t_{jkl}\_t'_{kl}$	302	374	339	339	341	343	343	—	—	—	—
$P2f0.2\_t_{kl}\_t'_{kl}$	350	462	400	400	400	400	400	—	—	—	—
$P2f0.3\_t_{jkl}\_t'_{kl}.1$	357	445	415	415	421	421	421	—	—	—	—
$P2f0.3\_t_{jkl}\_t'_{kl}.2$	302	433	413	416	417	417	417	—	—	—	—

Table 10.1: Lower bounds for the 40 test instances

	IS_2	HT_1	HT_2	HT_3	CP_1	CP_2	CP_4	CP_6
verified	0	0	0	1	0	0	0	16
$\Delta(UB)_{av}$	12.1	9.3	5.3	4.3	10.9	10.5	10.1	0.6
$\Delta(UB)_{max}$	21.4	15.8	13.4	12.3	19.5	19.5	18.4	3.0
CPU aver.	0:17	0:20	0:34	0:36	10:31	45:22	1.26:32	1:47
CPU max.	0:20	0:25	0:49	0:51	1.07:54	3.37:29	6.57:10	4:30

Table 10.2: Comparison of the lower bound procedures

The results of the lower bound calculations for the 30 instances with 30 operations are summarized in Table 10.2. For each lower bound procedure we determined the number of instances for which the best known heuristic solution UB could be verified by the corresponding value LB (i.e. for which $LB = UB$ holds) and the relative deviation $\Delta(UB) = \frac{UB-LB}{UB}$. Besides the average and maximal relative deviations $\Delta(UB)_{av}$ and $\Delta(UB)_{max}$ (in %) we report the average and maximal computation times (in hours.min:sec) for the whole binary search process (as an initial lower bound we used JPS and an initial upper bound was calculated by a fast version of our tabu search heuristic).

While the procedures `Improve_heads_tails_i (T)` required 100-200 MB storage, the chain-packing procedures required 100-300 MB for $q \in \{1, 2, 4\}$ and even up to 700 MB for $q = 6$ in our current implementation (since in the pricing procedure up to $O(qm^qT)$ real numbers are stored). For this reason it was not possible to calculate the chain-packing bounds with $q > 4$ for the instances with 90 operations. Since for the smaller instances the chain-packing bounds with $q \neq n$ only produced bad results, we did not calculate these bounds for the larger instances. In order to apply the chain-packing procedures to larger instances, the storage requirement has to be reduced in our implementation. This could be done by similar techniques which we already used in the dynamic programming procedure of Subsection 6.2 (generation of the tuples according to increasing cardinalities of the corresponding subsets, hashing techniques).

For the 10 instances with 90 operations no instance could be verified by the lower bound HT_3, the mean (maximal) relative deviations were $\Delta(UB)_{av} = 5.7\%$ ($\Delta(UB)_{max} = 14.3\%$). The average (maximal) computation times increased to 28:08 (57:03) minutes.

The results show that good lower bounds can only be calculated with much effort. While the quality of the bounds obtained by polynomial procedures (JPS and IS_2) is not satisfactory, strong lower bounds can only be determined with exponential algorithms (HT_3 or CP_6). At first sight it seems to be surprising that the chain-packing procedure for $q = 6$ needs much less time than the procedures for $q \in \{1, 2, 4\}$ which produce much weaker bounds. But this can be explained by the fact that for $q = 6$ the pricing procedure does not have to be called very often. In many cases this procedure has to be called only once for a threshold value T: either it determines a feasible q-chain-schedule which decreases the objective of the linear program to zero in one step, or it finds no improving column at all, which means that infeasibility can be stated. On the other hand, for smaller q-values often many iterations (100-300 in the average case, up to 2000 in the worst case) with only very small improvements in the objective are necessary until the linear program is solved to optimality.

10.3 Heuristic solutions for $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$

In this subsection we compare the heuristic procedures which have been described in Section 8. The simplest procedure is the priority-based heuristic which has been tested for four different priority rules, as mentioned in Subsection 8.2. Additionally, each of the resulting four permutations is used as a starting solution for an iterative improvement procedure and the tabu search algorithm described in Subsection 8.4. The iterative improvement procedure is based on neighborhood $\mathcal{N}_2$ and stops if no improving neighbor of the current solution exists.
Several parameters may be varied concerning the tabu search algorithm. Since some preliminary tests showed that the "best-fit" strategy outperforms the "first-fit" strategy in most cases, we only used the "best-fit" strategy. Furthermore, we tested 13 strategies for choosing the next operator which is applied to the current solution. Since no strategy clearly outperformed the others and the computation times were small, we decided to include them all into the algorithm (i.e. we start with each starting solution 13 times). We performed two restarts from the best solution found so far (where additionally the strategy was changed). Finally, we tested different parameter values for the minimal and maximal tabu list lengths ($TL_{\min}$ and $TL_{\max}$), and the maximal number of iterations after which the search process is stopped when the best solution is not improved within these iterations ($maxiter$). Further technical details and additional computational results for the different heuristics will be found in Bendfeld [1999].

The upper bounds obtained by the different algorithms for the 40 test instances are presented in Table 10.3. This table contains the following information:

- Problem: the benchmark problem,
- $\sum t_{jkl}$: the sum of all transportation times of the transport operations,
- LB: the best known lower bound for the problem,
- UB^{pr}: the best objective value obtained by the priority-based heuristic,
- II: the objective value obtained by iterative improvement,
- TS_1: the objective value obtained by a version of the tabu search algorithm with $TL_{\min} = 3$, $TL_{\max} = 16$ and $maxiter = 500$,
- TS_2: the objective value obtained by a version of the tabu search algorithm with $TL_{\min} = 4$, $TL_{\max} = 14$ and $maxiter = 1500$,
- TS: the best objective value found by a version of the tabu search algorithm,

Problem	$\sum t_{jkl}$	LB	UB^{pr}	II	TS_1	TS_2	TS
$P1\_t_{jkl}\_t'_{kl}.1$	124	138	155	144	139	139	138
$P1\_t_{jkl}\_t'_{kl}.2$	118	131	151	140	132	132	131
$P1\_t_{jkl}\_t'_{kl}.3$	132	145	159	151	146	146	145
$P1\_t_{jkl}\_t'_{kl}.4$	96	109	124	116	111	111	111
$P1\_t_{jkl}\_t'_{kl}.5$	114	126	141	133	127	126	126
$P1\_t_{jkl}\_t'_{kl}.6$	119	131	144	142	132	132	132
$P1\_t_{jkl}\_t'_{kl}.7$	117	129	145	139	130	131	129
$P1\_t_{jkl}\_t'_{kl}.8$	123	136	147	141	137	136	136
$P1\_D1\_d1$	66	87	96	94	87	87	87
$P1\_D1\_t1$	66	81	87	85	82	82	82
$P1\_D2\_d1$	132	151	165	162	153	153	153
$P1\_D2\_d2$	132	164	189	188	174	171	169
$P1\_D2\_t1$	132	144	160	150	144	144	144
$P1\_D2\_t2$	132	152	170	165	154	153	153
$P1\_D3\_d1$	198	217	240	232	219	217	217
$P1\_D3\_d2$	198	230	281	262	233	234	233
$P1\_D3\_d3$	198	243	276	276	251	249	246
$P1\_D3\_t1$	198	210	220	216	210	211	210
$P1\_D3\_t2$	198	218	237	231	220	220	219
$P1\_D3\_t3$	198	227	252	246	230	231	227
$P1\_D5\_d5$	330	397	454	454	407	409	397
$P1\_t_{kl}\_t'_{kl}.1$	121	141	147	144	144	143	143
$P1\_t_{kl}\_t'_{kl}.2$	127	151	188	175	156	153	152
$P1\_t_{kl}\_t'_{kl}.3$	114	130	156	141	133	133	132
$P1\_T2\_t2$	60	81	97	87	82	83	82
$P1\_T3\_t1$	90	102	111	109	102	102	102
$P1\_T3\_t2$	90	110	122	122	110	112	110
$P1\_T3\_t3$	90	117	149	137	120	118	117
$P1\_T5\_t2$	150	168	182	174	168	168	168
$P1\_T5\_t5$	150	188	218	218	190	192	190
$P2\_D2\_d2$	446	1002	1134	1061	1007	1007	1005
$P2\_D3\_d1$	669	1146	1283	1178	1149	1149	1149
$P2\_D3\_d2$	669	1165	1363	1267	1180	1180	1176
$P2f0.1\_D1\_d1$	223	288	413	377	346	347	336
$P2f0.2\_D1\_d1$	223	334	435	414	377	374	370
$P2f0.5\_D1\_d1$	223	573	624	596	577	577	577
$P2f0.2\_t_{jkl}\_t'_{kl}$	302	343	412	400	377	375	374
$P2f0.2\_t_{kl}\_t'_{kl}$	350	400	531	499	468	467	462
$P2f0.3\_t_{jkl}\_t'_{kl}.1$	357	421	490	472	448	445	445
$P2f0.3\_t_{jkl}\_t'_{kl}.2$	302	417	481	451	435	434	433

Table 10.3: Heuristic results for the 40 test instances

The results of the upper bound calculations are summarized in Tables 10.4 and 10.5. For each heuristic solution value UB we determined the relative deviation $\Delta(LB) = \frac{UB-LB}{LB}$ from the best known lower bound value LB and report the average and maximal relative deviations $\Delta(LB)_{av}$ and $\Delta(LB)_{max}$ (in %).

		UB^{pr}	II	TS_1	TS_2	TS
30	$\Delta(LB)_{av}$	12.8	8.3	1.3	1.2	0.6
	$\Delta(LB)_{max}$	27.4	17.1	6.1	4.3	3.0
90	$\Delta(LB)_{av}$	20.9	13.8	7.3	7.1	6.4
	$\Delta(LB)_{max}$	43.4	30.9	20.1	20.5	16.7

Table 10.4: Comparison of the quality of the heuristic procedures

Table 10.4 shows that the quality of the tabu search algorithm is quite good. While for the instances with 30 operations the mean relative deviation from the best known lower bound is $\Delta(LB)_{av} = 0.6\%$, for the instances with 90 operations we have $\Delta(LB)_{av} = 6.4\%$ (note that for the larger instances only the lower bounds HT_3 were considered, since the chain-packing bounds could not be calculated). On average the tabu search algorithm improved the best starting solution by approximately 10%.

In our implementation of the tabu search algorithm we used the techniques to accelerate the evaluation of neighbored solutions as described in Subsection 8.3. From the theoretical point of view, the complexity of the evaluation process was reduced approximately by the factor $O(n)$. In order to study the improvement from the practical point of view, we implemented a second version of the tabu search procedure in which for each neighbor the correct objective value is determined. With respect to the quality of the calculated solutions no significant differences could be observed, but the computational times increased. A comparison of the average and maximal CPU times (in min:sec) for the fast implementations TS_1^f and TS_2^f, and the slow versions TS_1^s and TS_2^s for the instances with 30 and 90 operations can be found in Table 10.5. The computational times could be reduced up to the factor 3 with our fast versions, which shows that our accelerations are also practically worthwhile.

	30				90			
	TS_1^f	TS_1^s	TS_2^f	TS_2^s	TS_1^f	TS_1^s	TS_2^f	TS_2^s
average CPU	0:37	1:03	1:47	2:57	2:09	6:34	5:43	17:30
maximal CPU	0:43	1:17	2:07	3:39	2:59	8:58	8:09	24:30

Table 10.5: Comparison of the computational times for the two implementations

Summarizing, we can conclude that the tabu search procedures calculate good upper bounds in a short amount of time.

10.4 Heuristic solutions for $J, R1 \mid t_{jkl}, t'_{kl} \mid C_{\max}$

In this subsection we report some results for the job-shop problem with transportation and empty moving times and a single robot. Since no good lower bound procedures are available for this problem, we compare our tabu search algorithm described in Section 9 with the tabu search procedure of Küsters [1997] for the general-shop problem with sequence-dependent setup times. This tabu search procedure is based on a neighborhood similar to $\mathcal{N}_3$ (cf. Subsection 9.1). In each iteration either the machine selection is changed by a move of an operation belonging to a machine-block or the robot sequence is changed by moving a transport operation belonging to a robot-block to another position.

Some preliminary tests showed that the behavior of Küsters' algorithm does not strongly depend on the starting solution, but on the tabu list length. On the other hand, our algorithm produces different solutions for different starting solutions, but only slightly depends on the tabu list length. Thus, we decided to run Küsters' algorithm three times for each instance (with the same starting solution, but three different static tabu list lengths) and stopped each run when the best solution had not been improved within the last 10 000 iterations. On the other hand, our tabu search algorithm was started with 6 different starting solutions and a static tabu list of length 8, and was stopped when the best solution had not been improved within the last 10 iterations in the outer stage.

The upper bounds obtained by the different algorithms for the 10 job-shop test instances with $n = m = 6$ and the 20 instances with $n = m = 10$ are presented in Table 10.6. This table contains the following information:

- Problem: the benchmark problem,
- $\sum t_{jkl}$: the sum of all transportation times of the transport operations,
- UB^0: the best objective value obtained by a version of the initial priority-based heuristic,
- UB^{K97}: the best objective value calculated with one of the three versions of Küsters' tabu search algorithm,
- UB^{TS}: the best objective value calculated with a version of the tabu search algorithm in Section 9,
- CPU: the computational times (in min:sec) for the version of the corresponding algorithm which produced the best objective value.

The maximal machine load (i.e. the sum of processing times of all operations processed on the same machine) is 43 for the $P1$-instances, 631 for the $P2$-instances, and 357 for the $P2f0.5$-instances.

Problem	$\sum t_{jkl}$	UB^0	UB^{K97}	CPU	UB^{TS}	CPU
$P1\_t_{jkl}\_t'_{kl}.1$	124	137	134	1:34	135	3:15
$P1\_t_{jkl}\_t'_{kl}.2$	118	134	131	1:10	133	2:40
$P1\_t_{jkl}\_t'_{kl}.3$	132	146	144	2:29	145	2:12
$P1\_D1\_d1$	66	90	87	0:48	87	4:12
$P1\_D1\_t1$	66	83	81	1:07	82	3:13
$P1\_D2\_d1$	132	155	148	2:32	151	7:12
$P1\_D3\_d1$	198	218	217	0:59	218	1:32
$P1\_t_{kl}\_t'_{kl}.1$	121	139	137	1:22	138	1:43
$P1\_T2\_t1$	60	76	74	0:46	75	2:56
$P1\_T3\_t0$	90	94	93	1:41	92	2:13
$P2\_D1\_d1$	223	1088	1044	1:33	994	12:35
$P2\_D1\_t0$	223	1088	1049	0:42	1013	2:06
$P2\_D1\_t1$	223	1088	1016	2:03	1012	2:31
$P2\_D2\_d1$	446	1116	1070	1:16	1019	32:37
$P2\_D3\_d1$	669	1141	1094	6:36	1085	30:41
$P2\_D5\_t2$	1115	1424	1412	17:03	1381	50:28
$P2\_T1\_t1$	90	1092	1080	1:13	1053	2:13
$P2\_T2\_t1$	180	1094	1082	0:52	1058	2:14
$P2\_T5\_t2$	450	1102	1102	1:05	1073	5:25
$P2\_t_{jkl}\_t'_{kl}.1$	338	1101	1082	1:15	1066	2:12
$P2\_t_{jkl}\_t'_{kl}.2$	333	1097	1104	0:53	1063	2:19
$P2\_t_{jkl}\_t'_{kl}.3$	298	1093	1081	1:12	1065	3:51
$P2\_t_{jkl}\_t'_{kl}.4$	368	1096	1084	1:34	1060	5:01
$P2\_t_{kl}\_t'_{kl}.1$	337	1098	1086	1:42	1067	4:03
$P2\_t_{kl}\_t'_{kl}.2$	385	1102	1088	2:36	1069	5:54
$P2f0.5\_D1\_d1$	223	595	563	2:48	547	59:53
$P2f0.5\_D1\_t1$	223	595	545	1:20	543	11:35
$P2f0.5\_D2\_d1$	446	689	681	8:22	675	56:05
$P2f0.5\_D2\_t0$	446	607	603	13:17	590	51:56
$P2f0.5\_D2\_t1$	446	630	628	11:08	623	44:17

Table 10.6: Heuristic solutions for the job-shop instances with a single robot

The results of the upper bound calculations are summarized in Table 10.7. For each heuristic solution value UB we determined the relative improvement $\Delta(UB^0) = \frac{UB^0 - UB}{UB^0}$ w.r.t. the best starting solution UB^0 and report the average and maximal relative improvements $\Delta(UB^0)_{av}$ and $\Delta(UB^0)_{max}$ (in %). Furthermore, we compare both algorithms according to the average and maximal CPU times (in min:sec).

		$\Delta(UB^0)_{av}$	$\Delta(UB^0)_{max}$	CPU_{av}	CPU_{max}
$n = 6$	UB^{K97}	2.2	4.5	1:27	2:32
	UB^{TS}	1.4	3.3	3:07	7:12
$n = 10$	UB^{K97}	2.3	8.4	3:55	17:03
	UB^{TS}	4.5	8.7	19:23	59:53

Table 10.7: Comparison of the two heuristic procedures

Table 10.7 shows that the two tabu search algorithms behave differently for different test instances. While Küsters' algorithm produces better results for the smaller instances with $n = 6$, our method is superior for the larger instances with $n = 10$. These two types of instances do not only differ in their problem size, but also in the proportion between transportation and processing times. In the smaller instances the transportation stage is dominant (while the maximal machine load is 43, the robot load varies in [60, 198]), whereas in the larger instances both the machines and the transportation stage are important. The bad behaviour of our procedure for the first type of instances can be explained by the fact that critical paths for these instances mainly contain transport operations and only few (or even no) machine-blocks or single critical operations. In this case the used neighborhood $\mathcal{N}_5$ is very small and it is very difficult to escape from local minima using only moves according to $\mathcal{N}_5$. Thus, we conjecture that for these instances some additional diversification techniques have to be integrated into our algorithm.

According to the computational times the tabu search of Küsters is considerably faster than our tabu search method. This is caused by the fact that in the first algorithm lower bounds are used for evaluating neighbored solutions, while we calculate exact objective values for each neighbor using a version of our single-machine tabu search procedure. But since in a first step we mainly wanted to compare both concepts according to their quality, we allowed larger computational times for our algorithm. If similar acceleration techniques were incorporated, we conjecture that the computational times could also be reduced drastically without changing the quality significantly. In both algorithms the computation times increase for larger transportation times.

Summarizing, we can conclude that our two-stage approach seems to be a good foundation for solving the job-shop problem with transportation times and a single robot. The first computational results are promising and we conjecture that they can be improved by incorporating some additional techniques.

11 Concluding remarks

We studied several scheduling problems with transportation aspects which generalize classical shop-scheduling models. Two types of problems were distinguished: on the one hand, problems without transportation conflicts (i.e. each transportation can be performed without delay), and on the other hand, problems where transportation conflicts may arise due to a limited capacity of transport robots. The problems were considered from a theoretical and a practical point of view.

First we classified which problems are polynomially solvable and which are $\mathcal{NP}$-hard. While for shop-problems without transportation conflicts several complexity results were known from the literature, for shop-problems with transportation conflicts only one $\mathcal{NP}$-hardness proof was available. Therefore, we studied the second type of problems more intensively and derived new complexity results for flow-shop problems with a single robot. Concerning such problems with two machines and the makespan criterion the borderline between easy and hard problems was completely determined. Since very special cases of these problems are already $\mathcal{NP}$-hard, it is very unlikely that they can be solved in polynomial time.

Thus, in the second part of this thesis we tried to develop some techniques for dealing with these hard problems in practice. We extended the disjunctive graph model in which feasible schedules are represented by sequences on the machines and the robots. In this context we considered the subproblem of scheduling a single robot which arises when the sequences on the machines are fixed. For this single-machine problem (a traveling salesman problem with time windows and minimal time-lags) we adapted immediate selection techniques used for other scheduling problems and calculated lower bounds with these methods. Another lower bound procedure, based on linear programming and the technique of column generation, provided very good results. On the other hand, to determine upper bounds for the single-machine problem we developed an efficient local search algorithm which finds good solutions in reasonable time. This algorithm was integrated into a local search algorithm for the job-shop problem with a single robot. Finally, we tested the algorithms on different test data and presented computational results.

There are still many interesting research topics in the area of scheduling problems with transportation issues. With respect to the computational complexity, the status of many problems is still open (e.g. flow-shop problems with a single robot and other criteria than the makespan objective, problems in other environments like open- or job-shops with a single robot, or problems with a limited number of robots greater than one).
From the practical point of view the presented algorithms may be a foundation for further research and additional improvements. Especially the lower bound calculations have shown that the single-machine problem for a single robot is very difficult. Good results could only be obtained by exponential algorithms which are not applicable to larger instances. Thus, some faster procedures with a satisfactory quality should be developed.

Concerning the job-shop problem with a single robot lower bounds should be calculated in order to estimate the quality of the presented heuristics. For this purpose the single-machine immediate selection procedures could be integrated into a destructive approach which combines the single-machine methods with immediate selection techniques known for the classical job-shop problem.
According to the calculation of heuristic solutions we have seen that different neighborhood structures are appropriate for different data configurations. Thus, it should be investigated in more detail how the advantages of different approaches can be combined in order to deal with a larger variety of instances.

With respect to extensions of the proposed algorithms the situation with more than one robot could be studied next. It would be sufficient to replace the tabu search algorithm for the single-robot problem by an algorithm for the corresponding parallel robot problem in our decomposition approach, while the neighborhood structures on the level of the machines could be retained.

References

[1] **Achugbue, J.O., Chin, F.Y. [1982]** Scheduling the open shop to minimize mean flow time, SIAM J. on Computing 11, 709-720.

[2] **Adams, J., Balas, E., Zawack, D. [1988]** The shifting bottleneck procedure for job-shop scheduling, Management Science 34, 391-401.

[3] **Agnetis, A. [1996]** Planning the routing mix in FASs to minimize total transportation time, International Journal of Flexible Manufacturing Systems 8.

[4] **Agnetis, A. [1997]** Scheduling no-wait robotic cells with two and three machines, Technical Report 21.97 Universita Romà "La Sapienza".

[5] **Balas, E., Lenstra, J.K., Vazacopoulos, A. [1995]** One-machine scheduling with delayed precedence constraints and its use in job shop scheduling, Management Science 41, 94-109.

[6] **Baptiste, P. [1998]** Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine when processing times are equal, Technical Report 98-138, Université de Technologie de Compiègne, France.

[7] **Baptiste, P., Le Pape, C. [1996]** Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling problems, Proc. 15th Workshop of the U.K. Planning Special Interest Group, Liverpool.

[8] **Bendfeld, S. [1999]** Lokale Suchverfahren für Job-Shop Probleme mit einem Transportroboter, Diplomarbeit, Universität Osnabrück, Fachbereich Mathematik/Informatik, in preparation.

[9] **Bilge, Ü., Ulusoy, G. [1995]** A time window approach to simultaneous scheduling of machines and material handling system in an FMS, Operations Research 43, 1058-1070.

[10] **Blażewicz, J., Burkard, R.E., Finke, G., Wöginger, G.J. [1994]** Vehicle scheduling in two-cycle flexible manufacturing systems, Math. Comput. Modelling 20, 19-31.

[11] **Brauner, N., Finke, G. [1998]** Final results on the one-cycle conjecture in robotic cells, Internal Note, Laboratoire Leibniz, Institut IMAG, Grenoble.

[12] **Brauner, N., Finke, G. [1999]** On a conjecture about robotic cells: new simplified proof for the three-machine case, Journal of Information Systems and Operational Research - INFOR 37.

[13] **Brucker, P. [1998]** Scheduling algorithms, 2nd revised and enlarged edition, Springer Verlag, Berlin.

[14] **Brucker, P., Hilbig, T., Hurink, J. [1999]** A branch & bound algorithm for a single machine scheduling problem with positive and negative time-lags, Discrete Applied Mathematics 94, 77-99.

[15] **Brucker, P., Jurisch, B., Krämer, A. [1994]** The job-shop problem and immediate selection, Annals of Operations Research 50, 73-114.

[16] **Brucker, P., Jurisch, B., Sievers, B. [1994A]** A branch & bound algorithm for the job-shop problem, Discrete Applied Mathematics 49, 107-127.

[17] **Brucker, P., Knust, S. [1998]** Complexity results for single-machine problems with positive finish-start time-lags, Osnabrücker Schriften zur Mathematik, Reihe P, Nr. 202, to appear in Computing.

[18] **Brucker, P., Knust, S., Schoo, A., Thiele, O. [1998A]** A branch & bound algorithm for the resource-constrained project scheduling problem, European J. Oper. Res. 107, 272-288.

[19] **Brucker, P., Thiele, O. [1996]** A branch & bound method for the general-shop problem with sequence dependent setup-times, OR Spektrum 18, 145-161.

[20] **Carlier, J. [1982]** The one machine sequencing problem, European J. Oper. Res. 11, 42-47.

[21] **Carlier, J., Pinson, E. [1989]** An algorithm for solving the job-shop problem, Management Science 35, 164-176.

[22] **Chvátal, V. [1983]** Linear Programming, W.H. Freeman and Company, New York.

[23] **Crama, Y., Klundert, J. van de [1997]** Cyclic scheduling of identical parts in a robotic cell, Operations Research 45, 952-965.

[24] **Crama, Y., Kats, V., Klundert, J. van de, Levner, E. [1997]** Cyclic scheduling in robotic flowshops, Working paper, University of Maastricht, RM/98/001, to appear in Annals of Operations Research.

[25] **Dell' Amico, M. [1996]** Shop problems with two machines and time lags, Operations Research 44, 777-787.

[26] **Dell' Amico, M., Trubian, M. [1993]** Applying tabu search to the job-shop scheduling problem, Annals of Operations Research 41, 231-252.

[27] **Desrochers, M., Lenstra, J.K., Savelsbergh, M.W., Soumis, F. [1988]** Vehicle routing with time windows: optimization and approximation, in Golden, Assad (eds.): Vehicle routing: methods and studies, Elsevier.

[28] **Dorndorf, U., Phan Huy, T., Pesch, E. [1998]** A survey of interval capacity consistency tests for time- and resource-constrained scheduling, in J.Węglarz (ed.): Handbook on recent advances in project scheduling, Kluwer.

[29] **Dumas, Y., Desrosiers, J., Gelinas, E. [1995]** An optimal algorithm for the traveling salesman problem with time windows, Operations Research 43, 367-371.

[30] **Finke, G., Gueguen, C., Brauner, N. [1996]** Robotic cells with buffer space, Conference Proceedings of ECCO IX, Dublin.

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

[32] **Garey, M., Johnson, D.S., Sethi, R. [1976]** The complexity of flow-shop and job-shop scheduling, Mathematics of Operations Research 1, 117-129.

[33] **Gilmore, P.C., Gomory, R.E. [1961]** A linear programming approach to the cutting-stock problem, Operations Research 9, 849-859.

[34] **Glass, C.A., Shafransky, Y.M., Strusevich, V.A. [1995]** Scheduling for parallel dedicated machines with a single server, CASSM R&D Paper 8, University of Greenwich.

[35] **Glover, F. [1989, 1990]** Tabu Search, Part I and II, ORSA Journ. Computing 1, 190-206, ORSA Journ. Computing 2, 4-32.

[36] **Gonzalez, T., Sahni, S. [1976]** Open shop scheduling to minimize finish time, J. Assoc. of Comput. Mach. 23, 665-679.

[37] **Grabowski, J., Nowicki, E. Zdrzalka, S. [1986]** A block approach for single machine scheduling with release dates and due dates, European J. Oper. Res. 26, 278-285.

[38] **Graham, R., Lawler, E.L., Lenstra, J.K., Rinnooy-Kan, A.H.G. [1979]** Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics 5, 287-326.

[39] **Hall, N., Kamoun, H., Sriskandarajah, C. [1997]** Scheduling in robotic cells: Classification, two and three machine cells, Operations Research 45, 421-439.

[40] **Hall, N., Potts, C., Sriskandarajah, C. [1994]** Parallel machine scheduling with a common server, Report, Faculty of Mathematical Studies, University of Southampton, UK, to appear in Discrete Applied Mathematics.

[41] **Hall, N., Sriskandarajah, C. [1996]** A survey of machine scheduling problems with blocking and no-wait in process, Operations Research 44, 510-525.

[42] **Hurink, J., Knust, S. [1998]** Flow-shop problems with transportation times and a single robot, Osnabrücker Schriften zur Mathematik, Reihe P, Nr. 201, submitted to Discrete Applied Mathematics.

[43] **Jackson, J.R. [1955]** Scheduling a production line to minimize maximum tardiness, Research Report 43, Management Science Research Project, University of California at Los Angeles.

[44] **Johnson, S.M. [1954]** Optimal two- and three-stage production schedules with setup times included, Naval Res. Logist. Quart. 1, 61-68.

[45] **Kats, V., Levner, E. [1997]** A strongly polynomial algorithm for no-wait cyclic robotic flowshop scheduling, OR Letters 21, 171-179.

[46] **Kats, V., Levner, E. [1997A]** Minimizing the number of robots to meet a given schedule, Annals of Operations Research 69, 209-226.

[47] **King, R.E., Hodgson, T., Chafee, F.W. [1993]** Robot task scheduling in a flexible manufacturing cell, IIE Transactions 25, 80-87.

[48] **Kise, H. [1991]** On an automated two-machine flowshop scheduling problem with infinite buffer, Journal of the Oper. Res. Soc. Japan 34, 354-361.

[49] **Kise, H., Shioyama, T., Ibaraki, T. [1991]** Automated two-machine flowshop scheduling: a solvable case, IIE Transactions 23, 10-16.

[50] **Klein, R., Scholl, A. [1999]** Computing lower bounds by destructive improvement - an application to resource-constrained project scheduling, European J. Oper. Res. 112, 322-346.

[51] **Klundert, J. van de [1996]** Scheduling problems in automated manufacturing, Ph.D. thesis, Faculty of Economics and Business Administration, University of Limburg, Maastricht.

[52] **Küsters, K. [1997]** Tabu-Suche für General-Shop Probleme mit gruppenabhängigen Rüstzeiten, Diplomarbeit, Fachbereich Mathematik/Informatik, Universität Osnabrück.

[53] **Laarhoven, P.J.M. van, Aarts, E.H.L., Lenstra, J.K. [1992]** Job shop scheduling by simulated annealing, Operations Research 40, 113-125.

[54] **Langston, M.A. [1987]** Interstage transportation planning in the deterministic flow-shop environment, Operations Research 35, 556-564.

[55] **Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. [1981,1982]** Minimizing maximum lateness in a two-machine open shop, Math. Oper. Res. 6, 153-158; Erratum, Math. Oper. Res. 7, 635.

[56] **Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P. [1977]** Complexity of machine scheduling problems, Annals of Discrete Mathematics 1, 343-362.

[57] **Lenstra, J.K., Rinnooy Kan, A.H.G. [1979]** Computational complexity of discrete optimization problems, Annals of Discrete Mathematics 4, 121-140.

[58] **Lenstra, J.K. [-]** Unpublished.

[59] **Levner, E., Kogan, K., Levin, I. [1995]** Scheduling a two-machine robotic cell: a solvable case, Annals of Operations Research 57, 217-232.

[60] **Martin, P.B. [1996]** A time-oriented approach to computing optimal schedules for the job shop scheduling problem, Ph.D. thesis, Graduate School of Cornell University, USA.

[61] **Mitten, L.G. [1958]** Sequencing n jobs on two machines with arbitrary time lags, Management Science 5, 293-298.

[62] **Muth, J.F., Thompson, G.L. [1963]** Industrial Scheduling, Englewood Cliffs, N.J., Prentice Hall.

[63] **Nowicki, E., Smutnicki, C. [1996]** A fast tabu search algorithm for the job shop problem, Management Science 42, 797-813.

[64] **Nowicki, E., Smutnicki, C. [1996A]** A fast tabu search algorithm for the permutation flow-shop problem, European J. Oper. Res. 43, 160-175.

[65] **Panwalkar, S.S. [1991]** Scheduling of a two-machine flowshop with travel time between machines, Journal Operational Research Society 42, 609-613.

[66] **Papadimitriou, C.H., Steiglitz, K. [1982]** Combinatorial Optimization, Prentice Hall, Englewood Cliffs, N.J.

[67] **Queyranne, M., Schulz, A. [1994]** Polyhedral approaches to machine scheduling, Preprint Nr. 408, TU Berlin.

[68] **Rayward-Smith, V.J., Rebaine, D. [1992]** Open shop scheduling with delays, Theoretical Informatics and Applications 26, 439-448.

[69] **Rebaine, D., Strusevich, V.A. [1998]** Two-machine open shop scheduling problem with special transportation times, CASSM R&D Paper 15, University of Greenwich, to appear in Journal Operational Research Society.

[70] **Roy, B., Sussmann, B. [1964]** Les problemes d'ordonnancement avec contraintes disjonctives, Note DS no. 9 bis, SEMA, Paris.

[71] **Seibert, S. [1999]** Untere Schranken für das Traveling Salesman Problem mit Zeitfenstern und verallgemeinerten Nachfolgebeziehungen, Diplomarbeit, Universität Osnabrück, Fachbereich Mathematik/Informatik, in preparation.

[72] **Sethi, S., Sriskandarajah, C., Sorger, G., Blażewicz, J., Kubiak, W. [1992]** Sequencing of parts and robot moves in a robotic cell, International Journal of Flexible Manufacturing Systems 4, 331-358.

[73] **Simons, B. [1978]** A fast algorithm for single processor scheduling, Proc. 19th Annnual Symp. Foundations of Computer Science, 246-252.

[74] **Simons, B. [1983]** Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines, SIAM J. Comput. 12, 294-299.

[75] **Sriskandarajah, C., Hall, N., Kamoun, H. [1998]** Scheduling large robotic cells without buffers, Annals of Operations Research 76, 287-321.

[76] **Stern, H., Vitner, G. [1990]** Scheduling parts in a combined production-transportation work cell, Journal Operational Research Society 41, 625-632.

[77] **Strusevich, V.A. [1999]** A heuristic for the two-machine open shop scheduling problem with transportation times, Discrete Applied Mathematics 93, 287-304.

[78] **Yu, W. [1996]** The two-machine flow shop problem with delays and the one-machine total tardiness problem, Ph.D. thesis, Technische Universiteit Eindhoven.

Danksagung

Bedanken möchte ich mich an dieser Stelle bei allen, die mir geholfen haben, diese Arbeit zu erstellen.

Herrn Prof. Dr. Brucker danke ich nicht nur für die Betreuung dieser Arbeit, sondern insbesondere auch für die Beteiligung am wissenschaftlichen Leben in unserem Arbeitsgebiet auf nationaler und internationaler Ebene.

Desweiteren danke ich Herrn Dr. Johann Hurink, der jederzeit zu fachlichen Diskussionen bereit war und mir immer wieder neue Anregungen gegeben hat.

Für die Hilfe bei den umfangreichen Implementierungsarbeiten bedanke ich mich bei den studentischen Hilfskräften Sandra Seibert, Sandra Bendfeld und Christian Strotmann. Ferner danke ich Teresa Gehrs für ihre Mühe, die Arbeit zu lesen und sprachlich zu korrigieren.

Der Deutschen Forschungsgemeinschaft bin ich zu Dank verpflichtet für die Finanzierung meiner Stelle im Projekt "Komplexe Maschinen-Schedulingprobleme", in dessen Umfeld diese Arbeit entstanden ist.

Ein besonderer Dank gilt meinen Eltern, die mich während der Erstellung der Arbeit im privaten Bereich in jeder Hinsicht unterstützt haben.