Complex Scheduling Problems

Dissertation submitted to
The Hungarian Academy of Sciences
for the degree "MTA Doktora"

Tamás Kis

Institute for Computer Science and Control

Hungarian Academy of Sciences

2017

dc_1311_16

Contents

Chapter 1

Introduction

Scheduling theory is a flourishing field of operations research. This is indicated by the large number of papers published in scientific journals and conference proceedings, as well as by the many practical applications. It is a field, where practical needs frequently lead to new research avenues, which is the case for most of the problems studied in this book.

Briefly stated, in the course of solving a (project) scheduling problem, we allocate resources, and time intervals to a set of activities, while respecting a number of constraints and minimizing some objective function(s).

Resources in (project) scheduling can be classified in two main categories according to the type of constraints on their usage: (i) *renewable resources*, and (ii) *non-renewable resources*. The consumption of renewable resources is limited in any time moment t, whereas the total consumption of each non-renewable resource is bounded in every time interval $[0, t)$. We mention that if a resource has both type of bounds, then it is called *doubly constrained*.

First of all, the basic *resource constrained project scheduling problem* (RCPSP) will be defined, whose variants and extensions constitute the problems studied in the following chapters.

In RCPSP, there is a finite set of *activities*, $\mathcal{J}$, a finite set of resources, $\mathcal{R}$, containing only renewable resources, and a *precedence relation* $\mathcal{E} \subset \mathcal{J} \times \mathcal{J}$. Each activity $j \in \mathcal{J}$ has a processing time $p_j \geq 0$, and has some requirements $a_{ij} \geq 0$ from each resource $i \in \mathcal{R}$, and each resource $i \in \mathcal{R}$ has a capacity $b_i > 0$. The execution of the activities cannot be interrupted, i.e., if some activity $j \in \mathcal{J}$ starts at time S_j, then it will complete at time $S_j + p_j$. Throughout the execution, activity j reserves a_{ij} units from each resource $i \in \mathcal{R}$. The total amount reserved from any resource cannot exceed its capacity at any time moment, i.e., if $S \in \mathbb{R}_+^{\mathcal{J}}$ is the vector of starting times

of the activities, then the inequalities

$$\sum_{j \in \mathcal{J} \,:\, S_j \le t \le S_j + p_j} a_{ij} \le b_i, \qquad \forall i \in \mathcal{R},\ t \ge 0 \tag{1.1}$$

must be respected. The precedence relations must be satisfied as well:

$$S_j + p_j \le S_k, \qquad \forall (j,k) \in \mathcal{E} \tag{1.2}$$

Those vectors $S \in \mathbb{R}_+^{\mathcal{J}}$ that satisfy (1.1) and (1.2) are called *feasible schedules*. The most widely studied objective function is the minimization of maximum activity completion time, or makespan, defined as $C_{\max}(S) = \max_j C_j(S)$, where $C_j(S) = S_j + p_j$. Being an $\mathcal{NP}$-hard optimization problem, several exact and heuristic approaches have been proposed in the literature for solving the basic project scheduling problem, see e.g., [49, 105].

We will present several results for machine scheduling problems. *Machines* are renewable resources of unit capacity, and each activity requires 0 or 1 unit of them.

In this dissertation I will present modeling, complexity and algorithmic results on scheduling problems, where (i) resources play a central role either in the objective function, or in the constraints, and where (ii) two problems in a subordinate relationship have to be solved.

The dissertation is divided into 3 chapters:

- Chapter 2 deals with resource leveling problems, where we are given renewable resources, and the objective is to minimize a function of the resource usage of the activities over time.
- Chapter 3 is devoted to machine scheduling problems with additional non-renewable resources.
- Chapter 4 is concerned with scheduling problems in a subordinate relationship.

In each chapter I will introduce the problem area, the most important previous work, and present new results authored or co-authored by myself.

Notation of results

For all the results presented in this thesis, the source is indicated, such as [45], and with the consent of my co-authors, those results reached by myself alone are marked by an asterisk, e.g., [45]*.

Chapter 2

Resource leveling problems

In a *resource leveling problem* we are given the maximum job completion time T, and a feasible schedule S is sought which minimizes some function of the resource usage over time, i.e.,

$$\begin{aligned} &\min \sum_{i \in \mathcal{R}} f_i(A_i^S) \\ &\text{subject to } (1.1), (1.2),\ 0 \leq S_j \leq T - p_j,\ j \in \mathcal{J}, \end{aligned} \tag{2.1}$$

where $A_i^S : [0, T] \to \mathbb{Q}$ is a mapping with $A_i^S(t) := \sum_{j \in \mathcal{J}, S_j \leq t < S_j + p_j} a_{ij}$, and $f_i(\cdot)$ is a real-valued function for each $i \in \mathcal{R}$. Among the many type of functions studied in the literature, the most well-known are

$$f_{\text{lin}}(A_i^S) := \int_0^T w_{it} \max\{0, A_i^S(t) - L_i\} dt, \tag{2.2}$$

that is, minimize the weighted resource usage above given limit L_i, and

$$f_{\text{quad}}(A_i^S) := \int_0^T w_{it} (A_i^S(t) - L_i)^2 dt, \tag{2.3}$$

in words, minimize the weighted squared difference between given resource limit L_i and the resource usage determined by S. Neumann and Zimmermann [86] have devised methods to solve the problem with respect to various functions f while observing, as well as neglecting the resource constraints (1.1).

In Section 2.1 we will study a resource leveling problem with f equal to (2.2), but in which the utilization of resources, unlike in the basic RCPSP, may change from time period to time period. Such a model is called *resource constrained project scheduling with variable intensity activities* (RCPSVP).

In that model, each activity $j \in \mathcal{J}$ has a time interval $[r_j, d_j]$ in which it must be entirely processed. Further on, for each unit-length time period $t \in [r_j, d_j]$, it has an intensity x_{jt} to be determined. The intensity represents the fraction completed during a unit-length time period, and it may vary between 0 and a constant maximum intensity, denoted by h_j for activity j. The sum of the intensities must be 1 for each activity. The amount of resources reserved for the activities are proportional to their intensities, i.e., for $j \in \mathcal{J}$ it is $a_{it}x_{jt}$ for each resource $i \in \mathcal{R}$ and $t \in [r_j, d_j]$. Firstly, it is shown that the problem is $\mathcal{NP}$-hard in the strong sense. Then, a new mixed integer linear program (MIP) is introduced for modeling the problem, and new valid inequalities are derived to strengthen the LP relaxation. Finally, computational result are summarized.

In Section 2.2 the RCPSVP model is extended with feeding precedence constraints. A feeding precedence constraint is described by an ordered triple (j, k, ϕ), where $j, k \in \mathcal{J}$ and $0 \leq \phi \leq 1$ is a rational parameter expressing the fraction of j to be completed before k may be started. If $\phi < 1$, then k may start before j finishes, but it is required that the total fraction of j completed up to any time period t cannot be less than that of k, i.e., $\sum_{\tau=r_j}^{t} x_{jt} \geq \sum_{\tau=r_k}^{t} x_{kt}$ for all $t \in [r_j + p_j^{\phi}, d_j] \cap [r_k, d_k]$, where p_j^{ϕ} is the minimum number of time periods needed to finish the ϕ fraction of j, i.e., $p_j^{\phi} = \lceil \phi / h_j \rceil$. Notice that for $\phi = 1$ we get back the familiar end-to-start precedence constraints. For this problem the results of RCPSVP will be generalized and new computational results will be presented.

In Section 2.3 resource leveling problems in a machine environment are discussed. Recall that machines are renewable resources with constant 1 availability, and each activity may require 0 or 1 unit from them. We will consider a multiple-machine environment, where each job is dedicated to a machine, but the starting times of the jobs on each machine have to be determined in order to minimize the objective function (2.1) for linear (2.2) or quadratic (2.3) functions. We will study the complexity of the problem, and determine the borderline between $\mathcal{NP}$-hard and tractable cases.

The content of Section 2.1 is based on Kis [64], that of Section 2.2 on Kis [65], and Section 2.3 presents selected results from Drótos and Kis [29].

2.1 Project scheduling with variable intensity activities

Most results on the resource-constrained project scheduling problem (RCPSP) assume *fixed* activity durations and a *constant rate* of resource usage while

performing every activity. Extensions to RCPSP relaxing at least one of these assumptions comprise *preemption of activity execution*, the *discrete time/resource trade-off*, the *time/cost trade-off* and the *multi-mode* resource-constrained project scheduling problems (see e.g Herroelen et al. [50], Brucker et al. [14] and Demeulemeester and Herroelen [25]). We will study a further extension, called *RCPSVP*, in which the intensity of each activity may vary over time and the resource-usage is proportional to the intensity.

An instance of the problem is given by a finite set $\mathcal{J}$ of *activities*, a finite set R of *continuously divisible renewable resources* and a *precedence relation* $\mathcal{E} \subset \mathcal{J} \times \mathcal{J}$ among the activities. The time horizon is divided into T unit-length periods, which will be indexed as $t = 1, \ldots, T$. For each activity $j \in \mathcal{J}$ a release time r_j and a deadline d_j specify an interval of time periods $\{r_j, \ldots, d_j\}$ in which the activity must entirely be executed, where $1 \leq r_j \leq d_j \leq T$. In each time period $t \in \{r_j, \ldots, d_j\}$ at most an $h_j \leq 1$ fraction of activity j may be completed. Activity j requires a total of a_{ij} units of resource i, for each $i \in \mathcal{R}$. If the intensity of activity j is x_{jt} in time period t, where $0 \leq x_{jt} \leq h_j$ and $\sum_{t=r_j}^{d_j} x_{jt} = 1$ hold, then it requires $a_{ij} \cdot x_{jt}$ units of resource i in that period. Each resource $i \in \mathcal{R}$ has an *internal capacity* of b_{it} units that is available free of charge, and it has an additional *external capacity* of $\bar{b}_{it}$ units at the expense of c_{it} for each external unit used.

The scheduling problem consists of determining for each activity j an intensity x_{jt} in each time period $t \in \{r_j, \ldots, d_j\}$ such that $0 \leq x_{jt} \leq h_j$, $\sum_{t=r_j}^{d_j} x_{jt} = 1$, the precedence constraints among the activities are fulfilled, the resource demand does not exceed the resource availability (internal + external) in any time period, and the total cost of using external capacity is minimized.

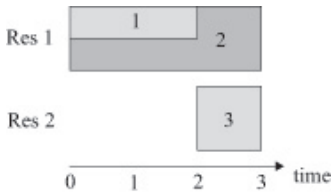

Figure 2.1: Resource usage of activities.

Example 1. To illustrate the above concepts consider a problem instance with 2 renewable resources and 3 activities. All activities have the same time window $r_j = 1, \ldots, d_j = 3$, while the maximum intensities are $h_1 = h_2 = 1/2$, and $h_3 = 1$. Activities 1 and 2 require $a_{11} = 1$, $a_{21} = 2$ unit of resource 1,

respectively, and activity 3 requires $a_{32} = 1$ unit of resource 2. The internal capacity of both resources is 1, and there is no external capacity available. As for precedence constraints, activity 1 must precede activity 3.

The unique solution to this instance is given by the intensity assignments $x_1 = (1/2, 1/2, 0)$, $x_2 = (1/4, 1/4, 1/2)$ and $x_3 = (0, 0, 1)$, respectively, where the t-th component of x_j represents the intensity of activity j in time period t. Fig. 2.1 depicts the resource usage of the activities with respect to x. □

2.1.1 Previous work

Project scheduling with variable intensity activities has been studied by several authors. Weglarz [104] proposes a continuous time model for allocating a single, doubly constrained resource to a set of activities over time, the objective being to minimize project duration. The single resource may be allocated to activities in arbitrary amounts within given intervals. The *performance-speed* or *intensity* of each activity is determined by a continuous, non-decreasing function of the amount of resource allocated to it at any moment. Weglarz provides several analytical results and discusses a few numerical examples. Leachman et al. [78] study a discrete time version of the model of Weglarz in which all the different types of resources required by an activity are applied proportionally to the (varying) intensity of the activity. The authors propose a heuristic algorithm for minimizing the makespan. For modeling and solving the special case where all lower bounds on activity intensities are 0, Tavares [99], [100] suggests a non-linear program, containing products of decision variables, which is solved by a standard non-linear optimization software. Finally, Hans [47] discusses in Chapter 6 of his Ph.D. thesis exactly the same model which is the topic of this section. His approach is branch-and-price, which is dual to our branch-and-cut approach. Hans obtains an initial upper bound by various constructive heuristics based on LP rounding techniques and by iterative improvement. In the column generation phase he uses a fast pricing algorithm and also rounding heuristics to obtain feasible solutions.

2.1.2 Summary of main contributions

We propose a new mixed integer-linear program formulation of RCPSVP, where the novelty lies in the modeling of the precedence constraints (Section 2.1.3). It is also shown how other objective functions fit in our framework. The pro's and con's of the alternative formulations are discussed in Section 2.1.4. In Section 2.1.5 we prove that RCPSVP is $\mathcal{NP}$-hard in the strong sense. In

Section 2.1.6, we analyze the polytope K_{jk} of all feasible intensity assignments to a pair of variable-intensity activities j and k with $(j,k) \in \mathcal{E}$. This polytope is decomposed into two smaller dimensional polytopes which, in fact, are instances of the same polytope K. Valid inequalities for K along with a separation algorithm is provided next. In Section 2.1.7 we show that the inequalities found previously completely describe the convex hull of K and also that of K_{jk} and establish conditions under which the inequalities represent facets of these polytopes. In Section 2.1.8 we describe a branch-and-cut algorithm based on our polyhedral results and summarize various computational experiments with it. In particular, we compare our method to that of Hans for RCPSVP and also a variant of our method to the approach of Tavares for minimizing the makespan.

2.1.3 New formulations

Since resources must be allocated to activities over discrete time periods, it is natural to use a time-indexed formulation with the following variables:

x_{jt} = intensity of activity j in time period t,
z_{jt} = mask of activity j in time period t. $z_{jt} = 1$ if activity j may be executed in time period t and 0 otherwise,
y_{it} = external capacity of resource $i \in \mathcal{R}$ used in time period t.

Let $p_j = \lceil 1/h_j \rceil$ denote the minimum time to complete activity j. Now suppose $(j,k) \in \mathcal{E}$, i.e., activity j must complete before activity j may start. Since activity j cannot complete before $r_j + p_j$, w.l.o.g. we may assume that $r_j + p_j \leq r_k$. Similarly, activity j must complete not later than $d_k - p_k$, that is, we may assume that $d_j \leq d_k - p_k$. After these preliminaries, the *weak formulation* for RCPSVP is as follows:

$$\text{Minimize} \sum_{i \in \mathcal{R}} \sum_{t \in \{1,\ldots,T\}} c_{it} \cdot y_{it}$$

subject to

$$\sum_{t=r_j}^{d_j} x_{jt} = 1, \quad j \in \mathcal{J}, \tag{2.4a}$$

$$x_{jt} \leq h_j \cdot z_{jt}, \quad j \in \mathcal{J},\ t \in \{r_j + p_j, \ldots, d_j\} \tag{2.4b}$$

$$x_{kt} \leq h_k \cdot (1 - z_{jt}), \quad (j,k) \in \mathcal{E},\ t \in \{r_k, \ldots, d_j\} \tag{2.4c}$$

$$z_{jt} \geq z_{j,t+1}, \quad j \in \mathcal{J},\ t \in \{r_j + p_j, \ldots, d_j - 1\}, \tag{2.4d}$$

$$\sum_{j:\ r_j \le t \le d_j} a_{ij} \cdot x_{jt} \le b_{it} + y_{it}, \quad i \in \mathcal{R},\ t \in \{1, \dots, T\}, \tag{2.4e}$$

$$0 \le y_{it} \le \bar{b}_{it}, \quad i \in \mathcal{R},\ t \in \{1, \dots, T\} \tag{2.4f}$$

$$0 \le x_{jt} \le h_j, \quad j \in \mathcal{J},\ t \in \{r_j, \dots, d_j\} \tag{2.4g}$$

$$z_{jt} \in \{0, 1\}, \quad j \in \mathcal{J},\ t \in \{r_j + p_j, \dots, d_j\} \tag{2.4h}$$

The objective is to minimize the weighted sum of external capacity used. By (2.4a), every activity must entirely be executed between its release time and deadline. The precedences between the activities are forced by (2.4b)–(2.4d). That is, activity j can be executed at time t only if $z_{jt} = 1$ (cf. (2.4b)). However, when $z_{jt} = 1$, no activity k with $(j,k) \in A$ may be executed, due to (2.4c). Finally, (2.4d) ensures that there is a time point $t_0 \in \{r_j + p_j, \dots, d_j + 1\}$ such that $z_{jt} = 1$ for all $t \in \{r_j + p_j, \dots, t_0 - 1\}$ and $z_{jt} = 0$ for all $t \in \{t_0, \dots, d_j\}$, hence, activity j has to complete before activity j may start. Since activity j cannot complete before $r_j + p_j$, we have no z_{jt} variables for $t \in \{r_j, \dots, r_j + p_j - 1\}$. The external capacity y_{it} of resource i used in time period t is determined by (2.4e) and it cannot be more than $\bar{b}_{it}$, due to (2.4f). Finally, the domains of the variables x_{jt} and z_{jt} are given by (2.4g) and (2.4h), respectively.

If activity j has no successors in $\mathcal{E}$, then z_{jt} can be set to 1 for all $t \in \{r_j + p_j, \dots, d_j\}$, and similarly y_{it} can be set to 0 if no activity may require resource i at time t, i.e., $t \notin \{r_j, \dots, d_j\}$ for all $j \in \mathcal{J}$ with $a_{ij} > 0$. A further reduction is possible for $(j,k) \in \mathcal{E}$. Since j precedes k, the difference $z_{kt} - z_{jt}$ must be 0 or 1 in any feasible solution. Hence, the inequalities $x_{kt} \le h_k \cdot (z_{kt} - z_{jt})$, $t \in \{r_k + p_k, \dots, d_j\}$, are satisfied by all feasible solutions and are stronger than (2.4b) for activity j and (2.4c) for (j,k) together. Therefore, we may replace (2.4b) and (2.4c) by the following set of inequalities:

$$x_{kt} \le \begin{cases} h_k \cdot (1 - z_{jt}), & t \in \{r_k, \dots, \min\{rk + p_k - 1, d_j\}\}, \\ h_k \cdot (z_{kt} - z_{jt}), & t \in \{r_k + p_k, \dots, d_j\}, \\ h_k \cdot z_{kt}, & t \in \{\max\{d_j + 1, r_k + p_k\}, \dots, d_k\}. \end{cases} \quad (j,k) \in \mathcal{E} \tag{2.4bc}$$

Notice that some of the intervals may be empty depending on the particular d_j and r_k values. We call the resulting program the *strong formulation*.

Besides minimizing the cost of using external resources, our model is suitable for other criteria as well. In the following three problems the external resource capacities are neglected. The *project duration* or *makespan* can be minimized by finding the smallest T (using dichotomic search between some lower and upper bounds), such that when setting all activity deadlines to T, the linear program (2.4) has a feasible solution. When a due-date $\tilde{d}_j$ is specified for each activity j, we may minimize the *maximum tardiness* by

finding the smallest $T_{\max}$ such that system (2.4) admits a feasible solution with activity deadlines $d_j = \tilde{d}_j + T_{\max}$. Again, the minimum value of $T_{\max}$ can be found by dichotomic search. In addition, when weights w_j are also given, we can minimize the *weighted tardiness* $\sum_j w_j(\max\{0, C_j - \tilde{d}_j\})$, where C_j is the completion time of activity j. That is, define new weights w_{jt} as follows: $w_{jt} = 0$ if $t \leq \tilde{d}_j$ and $w_{jt} = w_j$ for all $t \in \{\tilde{d}_j + 1, \ldots, T\}$. Then, the minimum value of $\sum w_{jt} \cdot z_{jt}$ is the minimum weighted tardiness with respect to time horizon T.

The polyhedral results and the branch-and-cut approach presented in this paper can be used to solve the RCPSVP with respect to any of the above optimization criteria.

2.1.4 Comparison of alternative formulations

In this section we discuss the advantages and disadvantages of the alternative formulations for RCPSVP, the emphasis being on integer-linear programming approaches. As all approaches model the resource constraints in essentially the same way, we focus on the modeling of precedence constraints.

Suppose activity j must precede activity k, i.e., $(j,k) \in \mathcal{E}$. Tavares [100] models this situation by a set of constraints equivalent to the following:

$$\left(\sum_{t=t_0}^{d_j} x_{jt} \right) x_{k,t_0} = 0, \quad \forall\, t_0 \in \{r_k, \ldots, d_k\}.$$

Tavares shows that the project duration can be minimized by defining new auxiliary variables z_t with $0 \leq z_t \leq 1$, $t \in \{1, \ldots, T\}$, and adding the constraints

$$\left(\sum_{j \in \mathcal{J}} \sum_{t=t_0}^{T} x_{jt} \right) z_{t_0} = 0, \quad \forall\, t_0 \in \{1, \ldots, T\}.$$

to the model. Notice that if $z_{t_0} > 0$, then no activity can be performed after t_0. Therefore, the above constraints along with the objective $\min(T - \sum_t z_t)$ express the makespan minimization problem. Since the constraints are clearly non-linear in the variables x and z, Tavares used a non-linear optimization software for solving the problem w.r.t. the makespan objective.

In contrast, Hans [47] models the precedence constraints using a set of binary vectors $\{\beta^\pi \in \{0,1\}^{|\mathcal{J}| \times T} \mid \pi \in \Pi\}$ consisting of the supports of all feasible intensity assignments to the activities. Although Hans considered distinct projects among which there were no precedence constraints, to simplify notation we assume that all activities belong to the same project. Notice that a binary vector $\beta \in \{0,1\}^{|\mathcal{J}| \times T}$ is the support of a feasible intensity assignment

if and only if $\sum_{t=1}^{T} \beta_{jt} \geq p_j$, $\min\{t \mid \beta_{jt} = 1\} \geq r_j$, $\max\{t \mid \beta_{jt} = 1\} \leq d_j$, and if $(j,k) \in \mathcal{E}$, then $\max\{t \mid \beta_{jt} = 1\} < \min\{t \mid \beta_{kt} = 1\}$. Clearly, precisely one vector β^π must be chosen. To this end, Hans has introduced new binary variables z_h, $h \in \Pi$, together with the following constraints:

$$\begin{aligned} \sum_{\pi \in \Pi} z_\pi &= 1, \\ z_\pi &\in \{0,1\}, \quad \pi \in \Pi, \\ 0 \leq x_{jt} &\leq h_j \left(\sum_{\pi \in \Pi} \beta^\pi_{i,t} z_\pi \right), \quad \forall j \in \mathcal{J},\ t \in \{r_j, \ldots, T\} \end{aligned}$$

The first two constraints ensure that exactly one vector β^π is chosen. The third one specifies that x_{jt} is either 0, or is between 0 and h_j, depending on whether β^h_{it} is 0 or 1. In addition, Hans' formulation also contains constraints equivalent to (2.4a) and (2.4e), and instead of (2.4f) it has $y_{it} \geq 0$, and $\sum_{i \in \mathcal{R}} y_{it} \leq \bar{b}_t$, for all t, where $\bar{b}_t$ is a cumulative upper bound for all resources. As the size of Π can be enormous, column generation is the only viable approach to handle this formulation.

The primary advantage is that any objective function which depends linearly on the cost associated with the vector β^π, whatever this cost be, can be optimized by the same solver, provided that the pricing problem can be solved efficiently. The main drawback is that a huge number of columns must be handled, and enlarging all activity deadlines by only one time period may multiply the size of Π which may increase considerably the running time of the pricing algorithm.

In contrast, if all activity deadlines are increased by one time period in our formulation, the number of variables grows only by $O(|\mathcal{J}| + |\mathcal{R}|)$ and the increase in the number of constraints is in $O(|\mathcal{J}| + |\mathcal{R}| + |\mathcal{E}|)$. The drawback is that good cutting planes must be supplied to the solver. Nevertheless, for the objective function studied in this paper our method is competitive with that of Hans, see Section 2.1.8.

2.1.5 Computational complexity

In this section we will prove that RCPSVP is NP-complete in the strong sense. To this end, we will show that RCPSVP contains the preemptive flowshop scheduling problem (PFSP) as a special case. As the latter problem has been shown NP-complete in the strong sense by Gonzalez and Sahni [39], our claim follows. For fundamental definitions and results of scheduling theory see e.g. Graham et al. [40] and Blazewicz et al. [8].

Recall that in a preemptive flowshop a finite set of *jobs*, JB, must be processed by a finite set of m *processors*, $\mathcal{M}$. A processor can process at most one job at a time. The *processing time* of job k on processor i is given by a positive integer number π_{ik} specifying that job k must be processed for exactly p_{ik} time units on processor i. If $p_{ik} = 0$ then job k is not processed by processor i. The processing of a job may be interrupted at any (integral) time point and resumed later. Each job k must visit the processors in the same order, that is, if $p_{i_1,k} > 0$, $p_{i_2,k} > 0$, and i_1 precedes i_2 in the order of processors, then job k must be processed for a total of $p_{i_1,k}$ time units on processor i_1 before its processing may be started on processor i_2. All jobs can be started at time 1 and the question is whether all jobs can be completed by a given time point C.

A solution σ to the PFSP specifies the time points in which a job is being processed by a processor. That is, $\sigma(k,i,t) = 1$ if job k is processed by processor i in the interval $[t, t+1]$ (t is an integer), and 0 otherwise. A solution is *feasible* if and only if it satisfies the following conditions:

(i) $\sum_t \sigma(k,i,t) = p_{ik}$ for all jobs k and processors i,

(ii) $\sum_k \sigma(k,i,t) \in \{0,1\}$ for all processors i and time periods t,

(iii) $\max\{t \mid \sigma(k,i_1,t) = 1\} < \min\{t' \mid \sigma(k,i_2,t') = 1\}$ for all jobs k and pairs of processors i_1, i_2 such that i_1 precedes i_2 in the order of processors, and $p_{i_1,k}, p_{i_2,k} > 0$ hold.

Below we describe a transformation f from PFSP to RCPSVP. Let I be an instance of PFSP, in the corresponding instance $f(I)$ of RCPSVP the set of resources $\mathcal{R}$ coincides with the set of processors $\mathcal{M}$. The set of activities $\mathcal{J}$ contains an activity for each job-processor pair such that the job has a positive processing time on the processor. The precedence relation $\mathcal{E}$ specifies the processing order of the activities, i.e., $\mathcal{E}$ consists of all the pairs (j, j') of activities such that j and j' represents the processing of some job k on consecutive processors. If activity j represents the processing of job k on processor i (when $p_{ik} > 0$), then the maximum intensity of activity j is $h_j = 1/p_{ik}$, and it requires $a_{ij} = p_{ik}$ units of resource (processor) i and no other resources. All activities can be started in time period 1 and they must all be completed at latest in time period C. The internal capacity of each resource is 1 in each time period and its external capacity is always 0. We have the following:

Lemma 1. *[64]*An instance I of PFSP admits a feasible solution if and only if the corresponding instance $f(I)$ of RCPSVP admits a feasible solution.*

Proof. Suppose first that the instance of PFSP has a solution σ. If activity j represents the processing of job k on processor i then set $x_{jt} = \sigma(k,i,t)/\pi_{ik}$. Moreover, set $z_{jt} = 1$ if and only if there exists $t' \geq t$ with $x_{jt'} > 0$, and 0 otherwise. Finally, set $y_{it} = 0$ for all resources (processors) i and time periods t. One can verify that the above (x,y,z) satisfies system (2.4).

Conversely, suppose system (2.4) admits a feasible solution (x',y',z'). We will prove that in this case there exists a possibly different feasible solution (x,y,z) of the system such that $x_{jt} = h_j$ or 0, $y = y'$ and $z = z'$. From this claim it follows that the schedule σ, defined by $\sigma(k,i,t) = x_{jt}/h_j$ for each k, i, t and j such that activity j represents the processing of job k by processor i, is a feasible solution for the PFSP instance. It is enough to verify property (ii), as the other two requirements of feasible schedules easily follow from the properties of feasible solutions of system (2.4). We have the following estimation on the number of jobs requiring a processor i in time period t:

$$\sum_k \sigma(k,i,t) = \sum_j x_{jt}/h_j = \sum_{j:a_{ij}>0, t\in\{r_j,\ldots,d_j\}} a_{ij} \cdot x_{jt} \leq 1.$$

Here, the second summation is over all activities j representing the processing of a job k by processor (resource) i. The second equation follows from the definition of h_j and a_{ij} and the inequality is due to the fact that (x,y,z) satisfies (2.4e) and y must be all 0 since all external resource capacities are 0.

To prove our claim, observe that in any feasible solution of (2.4), z is a 0/1 vector. Let U^j be the subset of time points where activity j may be executed, i.e., $z_{jt} = 1$ and $z_{j',t} = 0$ for all j' such that $(j',j) \in \mathcal{E}$ (cf. constraints (2.4b) and (2.4c)). Since all components of z are fixed to 0 or 1, we may rewrite the system (2.4) as follows:

$$\sum_{t=r_j}^{d_j} x_{jt} = 1, \quad \forall j \in \mathcal{J}, \tag{2.5a}$$

$$\sum_{j:a_{ij}>0} (1/h_j) \cdot x_{jt} \leq 1, \quad \forall i \in \mathcal{R}, t \in \{1,\ldots,T\} \tag{2.5b}$$

$$0 \leq x_{jt} \leq h_j, \quad \forall j \in \mathcal{J},\ t \in U^j \tag{2.5c}$$

$$x_{jt} = 0, \quad j \in \mathcal{J},\ t \notin U^j. \tag{2.5d}$$

Multiply the constraints (2.5a),(2.5c) and (2.5d) by $1/h_j$ and replace $(1/h_j) \cdot x_{jt}$ by a new variable $\tilde{x}_{jt}$. The resulting system, depicted below,

is a transportation problem.

$$\sum_{t=r_j}^{d_j} \tilde{x}_{jt} = 1/h_j, \quad j \in \mathcal{J} \tag{2.6a}$$

$$\sum_{j:a_{ij}>0} \tilde{x}_{jt} \leq 1, \quad \forall i \in \mathcal{R},\ t \in \{1,\ldots,T\} \tag{2.6b}$$

$$0 \leq \tilde{x}_{jt} \leq 1, \quad i \in J,\ t \in U^j \tag{2.6c}$$

$$\tilde{x}_{jt} = 0, \quad j \in \mathcal{J},\ t \notin U^j. \tag{2.6d}$$

Since the constraint matrix of (2.6) is totally unimodular and the right hand side is integral (since every h_j is the inverse of some $p_{ik} \in \mathbb{Z}^+$), the Hoffman-Kruskal theorem on unimodular matrices [51] implies that there exists an integral solution $\overline{x}$. Since $\overline{x}_{jt}$ is between 0 and 1, $\overline{x}$ is a 0-1 vector. Then, $x_{jt} = h_j \cdot \overline{x}_{jt}$ is a solution of (2.5) such that $x_{jt} = 0$ or h_j, as claimed. □

Before proving strong NP-completeness of RCPSVP notice that the maximum numbers $\mathrm{Max}[I]$ and $\mathrm{Max}'[f(I)]$ in an instance I of PFSP and in the corresponding instance $f(I)$ of RCPSVP, respectively, are the same, i.e., both have the value $\max\{\max p_{ij}, C\}$.

Corollary 1. *[64]*RCPSVP is NP-complete in the strong sense.*

Proof Membership in NP is straightforward. To show strong NP-completeness it suffices to verify that f is a *pseudo-polynomial transformation* (Garey and Johnson [36], p. 101) from PFSP to RCPSVP. Namely, an instance I of PFSP admits a feasible solution if and only if the corresponding instance $f(I)$ of RCPSVP has a feasible solution, by Lemma 1. Moreover, f can be computed in time polynomial in the length of I and $\mathrm{Max}[I]$. The length of $f(I)$ is not smaller than the length of I and $\mathrm{Max}'[f(I)]$ is bounded by a two-variable polynomial in the length of I and $\mathrm{Max}[I]$, since they are the same. □

As a consequence, RCPSVP cannot be solved in pseudo-polynomial time, unless P=NP. Hence, we will propose a branch-and-cut algorithm using strong valid inequalities which is the topic of the next section.

2.1.6 Polyhedral results for RCPSVP

If we omit the resource capacity constraints (2.4e) from the weak (or from the strong) formulation of RCPSVP, then the feasible solutions of the remaining system are all the intensity assignments to activities respecting the release

times, deadlines, maximum intensities and precedence constraints. We may consider this system as a collection of subsystems each describing the feasible intensity assignments to a *pair of activities* connected by a precedence constraint. In this and the next section we will obtain a complete description of the polytope associated with such a subsystem.

Feasible intensity assignments to a pair of activities

Let j, k be a pair of activities with $(j,k) \in \mathcal{E}$. If $d_j < r_k$, then this precedence constraint is meaningless, so we may assume that $r_k \leq d_j$. Recall also that $r_j + p_j \leq r_k$ and $d_j \leq d_k - p_k$. We define the polytope K_{jk} as the convex hull of all feasible intensity assignments to j and k such that j completes before k starts. That is, K_{jk} is the convex hull of all points $(x_j, x_k, z_j) \in \mathbb{R}^{s_j} \times \mathbb{R}^{s_j} \times \{0,1\}^{s_j - p_j}$, where $s_j = d_j - r_j + 1$ and $s_k = d_k - r_k + 1$, satisfying the following constraints:

$$\sum_{t=r_j}^{d_j} x_{jt} = 1, \tag{2.7a}$$

$$0 \leq x_{jt} \leq h_j, \quad t \in \{r_j, \ldots, r_j + p_j - 1\} \tag{2.7b}$$

$$0 \leq x_{jt} \leq h_j \cdot z_{jt}, \quad t \in \{r_j + p_j, \ldots, d_j\} \tag{2.7c}$$

$$z_{jt} \geq z_{j,t+1}, \quad t \in \{r_j + p_j, \ldots, r_k - 1\} \tag{2.7d}$$

$$z_{jt} \geq z_{j,t+1}, \quad t \in \{r_k, \ldots, d_j - 1\} \tag{2.7e}$$

$$\sum_{t=r_k}^{d_k} x_{kt} = 1, \tag{2.7f}$$

$$0 \leq x_{kt} \leq h_k \cdot (1 - z_{jt}), \quad t \in \{r_k, \ldots, d_j\} \tag{2.7g}$$

$$0 \leq x_{kt} \leq h_k, \quad t \in \{d_j + 1, \ldots, d_k\} \tag{2.7h}$$

In order to find a linear representation of K_{jk} consider the polytopes K_{j*} and K_{*k} derived from K_{jk} as follows:

$$K_{j*} = \operatorname{conv}\left\{(x_j, z_j) \in \mathbb{R}^{s_j} \times \{0,1\}^{s_j - p_j} \mid (x_j, z_j) \text{ satisfies } (2.7a) - (2.7e)\right\},$$
$$K_{*k} = \operatorname{conv}\left\{(x_k, \tilde{z}_j) \in \mathbb{R}^{s_k} \times \{0,1\}^{d_j - r_k + 1} \mid (x_k, \tilde{z}_j) \text{ satisfies } (2.7e) - (2.7h)\right\}.$$

A feasible intensity assignment to a pair of acitivities is shown in Figure 2.2. The main result of this section is the following:

Lemma 2. *[64]*Let (x_j, x_k, z_j) be any point in $\mathbb{R}^{s_j} \times \mathbb{R}^{s_k} \times \mathbb{R}^{s_j - p_j}$. Then $(x_j, x_k, z_j) \in K_{jk}$ if and only if $(x_j, z_j) \in K_{j*}$ and $(x_k, \tilde{z}_j) \in K_{*k}$, where $\tilde{z}_{jt} = z_{jt}$ for all $t \in \{r_k, \ldots, d_j\}$.*

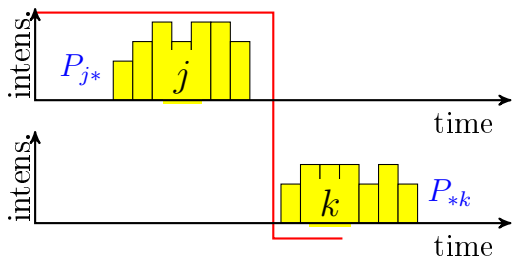

Figure 2.2: Feasible intensity assignment to a pair of activities $(j,k) \in \mathcal{E}$.

Before the proof we derive some common properties of K_{j*} and K_{*k}. In fact, K_{j*} and K_{*k} are both equivalent to the polytope K defined next. Let $1 \leq m < n$ be integer numbers, and $0 < h \leq 1$ a real number such that $(n-m)h \geq 1$. Then K is the convex hull of all points $(x,z) \in \mathbb{R}^n \times \{0,1\}^m$ satisfying the following linear constraints:

$$\sum_{t=1}^{n} x_t = 1 \tag{2.8a}$$

$$x_t \leq h \cdot (1 - z_t), \quad t \in \{1,\dots,m\} \tag{2.8b}$$

$$x_t \leq h, \quad t \in \{m+1,\dots,n\} \tag{2.8c}$$

$$z_t \geq z_{t+1}, \quad t \in \{1,\dots,m-1\} \tag{2.8d}$$

$$x_t \geq 0, \quad t \in \{1\dots,n\}. \tag{2.8e}$$

We obtain a polytope equivalent to K_{j*} by the substitutions $m = s_j - p_j$, $n = s_j$, $h = h_j$, $z_t = 1 - z_{j,d_j-t+1}$, $t \in \{1,\dots,m\}$, and $x_t = x_{j,d_j-t+1}$, $t \in \{1,\dots,n\}$. We get a polytope equivalent to K_{*k} by setting $m = d_j - r_k + 1$, $n = s_k$, $h = h_k$, $z_t = \tilde{z}_{j,t+r_k-1}$, $t \in \{1,\dots,m\}$, and $x_t = x_{k,t+r_k-1}$, $t \in \{1,\dots,n\}$.

A necessary and sufficient condition for a vector $(x,z) \in \mathbb{R}^n \times \mathbb{R}^m$ to be in K is provided next. First of all, if $(\hat{x},\hat{z})$ is a vertex of K, then $\hat{z}$ is one of the following vectors $z^\ell \in \{0,1\}^m$:

$$z^\ell_t = \begin{cases} 1 & \text{if } t \in \{1,\dots,\ell-1\}, \\ 0 & \text{if } t \in \{\ell,\dots,m\}, \end{cases} \qquad \ell \in \{1,\dots,m+1\}.$$

Moreover, if $\hat{z} = z^\ell$ for some vertex $(\hat{x},\hat{z})$ and some ℓ, then $\hat{x}_t = 0$ for all $t \in \{1,\dots,\ell-1\}$, $0 \leq \hat{x}_t \leq a$ for all $t \in \{\ell,\dots,n\}$ and $\sum_{t=1}^{n} \hat{x}_t = 1$. After these preparations, we can prove the following:

Lemma 3. *[64]* Let (x,z) be any point in $\mathbb{R}^n \times \mathbb{R}^m$. Then $(x,z) \in K$ if and only if the numbers $\lambda_1 = 1 - z_1$, $\lambda_\ell = z_{\ell-1} - z_\ell$ $(\ell \in \{2,\dots,m\})$ and $\lambda_{m+1} = z_m$*

are all non-negative and there exist vectors $x^\ell \in \mathbb{R}^n$, $\ell \in \{1, \ldots, m+1\}$, *such that* $\sum_{\ell=1}^{m+1} \lambda_\ell x^\ell = x$ *and every* (x^ℓ, z^ℓ) *belongs to* K.

Proof. (Necessity) If (x, z) is a point in K and the points $(\hat{x}^\theta, \hat{z}^\theta)$, $\theta \in \Theta$, constitute the set of vertices of K, then there exist reals $\omega_\theta \geq 0$ such that $(x, z) = \sum_{\theta\in\Theta} \omega_\theta(\hat{x}^\theta, \hat{z}^\theta)$, $\sum_{\theta\in\Theta} \omega_\theta = 1$.

Let $\lambda_\ell = \sum_{\theta:\hat{z}^\theta = z^\ell} \omega_\theta$, $\ell \in \{1, \ldots, m+1\}$. Then $\lambda_\ell \geq 0$ and $\sum_\ell \lambda_\ell = 1$. If $\lambda_\ell > 0$, let $x^\ell = \sum_{\theta:\hat{z}^\theta = z^\ell} (\omega_\theta/\lambda_\ell)\hat{x}^\theta$. Otherwise, if $\lambda_\ell = 0$, choose an arbitrary x^ℓ such that $(x^\ell, z^\ell) \in K$. Then every (x^ℓ, z^ℓ) belongs to K and $\sum_\ell \lambda_\ell(x^\ell, z^\ell) = (x, z)$. Using this and the fact that $z^\ell_m = 1$ if and only if $\ell = m+1$, $\lambda_{m+1} = z_m$ follows. An inductive argument proves that the rest of the λ_ℓ must also equal the specified values.

(Sufficiency) Observe that $\sum_{\ell=1}^{m+1} \lambda_\ell = 1$ and also $\sum_{\ell=1}^{m+1} \lambda_\ell z^\ell = z$. Consequently, if there exist vectors x^ℓ satisfying the conditions of the lemma, then (x, z) is a convex combination of the points (x^ℓ, z^ℓ). Since the vectors (x^ℓ, z^ℓ) all belong to K, $(x, z) \in K$. □

Proof of Lemma 2 By the definitions, if $(x_j, x_k, z_j) \in K_{jk}$, then $(x_j, z_j) \in K_{j*}$ and $(x_k, \tilde{z}_j) \in K^{*k}$, where $\tilde{z}_{jt} = z_{jt}$ for all $t \in \{r_k, \ldots, d_j\}$.

Conversely, suppose $(x_j, z_j) \in K_{j*}$ and $(x_k, \tilde{z}_j) \in K_{*k}$, where $\tilde{z}_{jt} = z_{kt}$ for all $t \in \{r_k, \ldots, d_j\}$. In order to apply the previous lemma to K_{j*} and K_{*k}, define the vectors $z^\ell_j \in \{0,1\}^{s_j - p_j}$ and $z^\ell_k \in \{0,1\}^{d_j - r_k + 1}$ as follows.

$$z^\ell_{jt} = \begin{cases} 1 & \text{if } t \in \{r_j + p_j, \ldots, \ell - 1\}, \\ 0 & \text{if } t \in \{\ell, \ldots, d_j\} \end{cases} \qquad \ell \in \{r_j + p_j, \ldots, d_j + 1\}.$$

For each $\ell \in \{r_k, \ldots, d_j + 1\}$, let $z^\ell_{kt} = z^\ell_{jt}$ for all $t \in \{r_k, \ldots, d_j\}$.

Applying Lemma 3 to K_{j*} yields vectors x^ℓ_j and coefficients λ_ℓ, $\ell \in \{r_j + p_j, \ldots, d_j + 1\}$ such that $\sum_{\ell=r_j+p_j}^{d_j+1} \lambda_\ell(x^\ell_j, z^\ell_j) = (x_j, z_j)$, $(x^\ell_j, z^\ell_j) \in K_{j*}$ for each ℓ, $\lambda_{r_j+p_j} = 1 - z_{r_j+p_j}$, $\lambda_\ell = z_{j,\ell-1} - z_{j\ell}$, $\ell \in \{r_j + p_j + 1, \ldots, d_j\}$, and $\lambda_{d_j+1} = z_{j,d_j}$.

For K_{*k} we get vectors x^ℓ_k and coefficients α_ℓ, $\ell \in \{r_k, \ldots, d_j + 1\}$ such that $\sum_{\ell=r_k}^{d_j+1} \alpha_\ell(x^\ell_k, z^\ell_k) = (x_k, \tilde{z}_j)$, $(x^\ell_j, z^\ell_j) \in K_{*k}$ for each ℓ, $\alpha_{r_k} = 1 - \tilde{z}_{j,r_k}$, $\alpha_\ell = \tilde{z}_{j,\ell-1} - \tilde{z}_{j,\ell}$, $\ell \in \{r_k + 1, \ldots, d_j\}$, and $\alpha_{d_j+1} = \tilde{z}_{j,d_j}$.

Since $z_{jt} = \tilde{z}_{jt}$, $t \in \{r_k, \ldots, d_j\}$ by assumption, it follows that $\alpha_\ell = \lambda_\ell$, $\ell \in \{r_k + 1, \ldots, d_j + 1\}$, and $\alpha_{r_k} = 1 - \sum_{\ell=r_k+1}^{d_j+1} \lambda_\ell = \sum_{\ell=r_j+p_j}^{r_k} \lambda_\ell$.

Letting $x^\ell_k = x^{r_k}_k$ for each $\ell \in \{r_j + p_j, \ldots, r_k - 1\}$, we have $(x_j, x_k, z_j) = \sum_{\ell=r_j+p_j}^{d_j+1} \lambda_\ell(x^\ell_j, x^\ell_k, z^\ell_j)$. Since each $(x^\ell_j, x^\ell_k, z^\ell_j)$ belongs to K_{jk} by the definition of the polytopes K_{j*} and K_{*k}, $(x_j, x_k, z_j) \in K_{jk}$ as well. □

Corollary 2. *[64]* If* $K_{j*} = \{(x_j, z_j) \mid A^{j*}x_j + B^{j*}z_j \leq b^{j*}\}$ *and* $K_{*k} = \{(x_k, \tilde{z}_j) \mid A^{*k}x_k + B^{*k}\tilde{z}_j \leq b^{*k}\}$, *then* $K_{jk} = \{(x_j, x_k, z_j) \mid A^{j*}x_j + B^{j*}z_j \leq$

$b^{j*}, A^{*k}x_k + [0, B^{*k}]z_j \leq b^{*k}\}$, *where 0 is a null matrix with* $r_k - r_j - p_j$ *columns and appropriate number of rows.*

We will provide a minimal linear representation of K in Section 2.1.7, which can trivially be transformed to one for K_{j*} and for K_{*k}, respectively. We will prove that putting together these two representations yields a minimal linear representation of K_{jk}.

Relation to fixed-charge network flows

In [89], the fixed charge network (or variable upper-bound) flow model is defined. That is, $\mathcal{P}_= = \text{conv}\{(x,z) \in \mathbb{R}^n \times \{0,1\}^n \mid \sum_{t=1}^n x_t = g, 0 \leq x_t \leq h_t z_t, t = 1, \ldots, n\}$, where g and the h_ts are constants. The system defining the polytope $\mathcal{P}_=$ models a fragment of a network consisting of a node u and arcs entering u. The total flow through the arcs has to meet a specified demand g. The upper bounds on the arcs are variable as they are determined by the binary variables z_t. In the definition of polytope K, there is an ordering on the arcs by the constraints $z_t \geq z_{t+1}$. That is, if z_{t_0} is set to 1, then for all $t \in \{1, \ldots, t_0\}$ the upper bound on x_t is fixed at h and if $z_{t_0} = 0$ then the upper bound is 0 on all arc flows x_t with $t \in \{t_0, \ldots, m\}$. As we will see, these extra constraints yield a new set of facets, different to the flow-cover inequalities of [89].

Valid inequalities and a separation algorithm for K

Clearly, the equation (2.8a) and the inequalities (2.8b)-(2.8e) are all valid for K. Moreover, the inequality

$$z_m \geq 0 \tag{2.8f}$$

is also valid for K. We derive a new class of valid inequalities below.

Denote $p = \lceil 1/h \rceil$ and $h_{\text{rem}} = 1 - (p-1)h$. Since $(n-m)h \geq 1$ by definition, $m + p \leq n$.

Lemma 4. *[64]Let* $\emptyset \neq S_1 \subseteq \{1, \ldots, m\}$ *and* $S_2 \subseteq \{m+1, \ldots, n\}$ *be such that* $|S_1| + |S_2| = p$ *and let* t_1 *be the smallest element of* S_1. *The* (S_1, S_2) *inequality*

$$h_{\text{rem}} z_{t_1} + h \sum_{t \in S_1 - \{t_1\}} z_t \leq \sum_{t \in \{t_1, \ldots, n\} - (S_1 \cup S_2)} x_t \tag{2.8g}$$

is valid for K.

Proof As K is a convex polytope, it suffices to show that any vertex of K satisfies (2.8g). Since the inequalities are valid for any vertex $(\hat{x}, \hat{z})$ with

$\hat{z}_{t_1} = 0$, assume $\hat{z}_{t_1} = 1$. Therefore, $\hat{x}_{t_1} = 0$ by (2.8b), and the total processing in $\{t_1, \ldots, n\} - (S_1 \cup S_2)$, that is, the right hand side of (2.8g), is at least 1 minus the maximum processing in $S_1 \cup S_2 - \{t_1\}$. The latter is $h|S_2| + h\sum_{t \in S_1 - \{t_1\}}(1 - \hat{z}_t)$. Plugging all this together, the statement follows. □

The usefulness of the (S_1, S_2) inequalities is illustrated by the following:

Example 2. Suppose $m = 4$, $n = 7$ and $h = 2/5$. Then $p = 3$ and $h_{\rm rem} = 1/5$. The vector (x, z), where $x = (1/10, 0, 0, 3/10, 2/5, 1/5, 0) \in \mathbb{R}^7$ and $z = (3/4, 1/2, 1/2, 1/4) \in \mathbb{R}^4$, satisfies (2.8a)-(2.8f), but violates (2.8g) for $S_1 = \{1, 4\}$ and $S_2 = \{5\}$. Namely, we have

$$h_{\rm rem} z_1 + h z_4 = \frac{1}{5} \cdot \frac{3}{4} + \frac{2}{5} \cdot \frac{1}{4} = \frac{1}{4} > \frac{1}{5} = 0 + 0 + \frac{1}{5} + 0$$
$$= x_2 + x_3 + x_6 + x_7 = \sum_{t \in \{1,\ldots,7\} - (S_1 \cup S_2)} x_t. \qquad \square$$

We close this section by a separation algorithm for the (S_1, S_2) inequalities. For each $t_1 \in \{1, \ldots, m\}$, the procedure tries to find a violated (S_1, S_2) inequality with t_1 being the smallest element of S_1. Namely, rewrite (2.8g) as follows:

$$\sum_{t \in S_1 - \{t_1\}} (h \cdot z_t + x_t) + \sum_{t \in S_2} x_t \le \left(\sum_{t \in \{t_1+1,\ldots,n\}} x_t \right) - h_{\rm rem} z_{t_1}.$$

Now consider the following optimization problem:

$$\max \sum_{t \in S_1 - \{t_1\}} (h \cdot z_t + x_t) + \sum_{t \in S_2} x_t, \tag{2.9}$$

where the max is over all set pairs (S_1, S_2) such that $t_1 \in S_1 \subseteq \{t_1, \ldots, m\}$, $S_2 \subset \{m+1, \ldots, n\}$ and $|S_1| + |S_2| = p$. The maximum is greater than the constant $\sum_{t \in \{t_1+1,\ldots,n\}} x_t - h_{\rm rem} z_{t_1}$ if and only if at least one (S_1, S_2) inequality with t_1 being the smallest element of S_1 is violated by (x, z).

In order to solve problem (2.9), define a set of items $I(t_1) = \{t_1+1, \ldots, n\}$ with item weights $w(t) = h \cdot z_t + x_t$ if $t_1 + 1 \le t \le m$, and x_t if $m + 1 \le t \le n$. Then the $p - 1$ largest-weight items constitute an optimal solution to (2.9). In fact, this problem can be seen as finding a maximum weight basis in the uniform matroid over $I(t_1)$ in which every $p - 1$ items is a basis. Repeating this procedure for each $t_1 \in \{1, \ldots, m\}$ we can find a violated (S_1, S_2) inequality or conclude that none exists.

For efficient implementation notice that the item weights do not depend on t_1 and thus it is enough to sort the elements of the set $I(1)$ in decreasing

order of their weights. Then, after increasing t_1 by 1, eliminate from the ordered list t_1. By using appropriate data structures, the time complexity of the entire separation procedure is $O(n \log n)$.

Proposition 1. *[64] Inequalities (2.8g) can be separated in $O(n \log n)$ time.*

2.1.7 Minimal linear representations for K and K_{jk}

In this section we will prove that the inequalities (2.8a)-(2.8g) constitute a linear representation of K. Moreover, we will establish conditions under which these inequalities represent facets of K. Finally, we will extend these results to K_{jk}.

A linear representation of K

Let P be the polytope consisting of all points $(x, z) \in \mathbb{R}^n \times \mathbb{R}^m$ satisfying the system (2.8a)-(2.8g). Since the equation (2.8a) and the inequalities (2.8b)-(2.8g) are all valid for K, $K \subseteq P$. Our main goal is to prove the following:

Theorem 1. *[64]* $P \subseteq K$.

Proof Assuming the contrary, fix some $(x, z) \in P \setminus K$. Since $(x, z) \notin K$, it is not a convex combination of points in K. We will show that then (2.8g) is violated for a pair of sets (S_1, S_2) which contradicts the assumption $(x, z) \in P$.

To this end, define the capacitated network $G(x, z) = (V, E)$ with node set V consisting of a source s, a sink q, a node v_ℓ for each $\ell \in \{1, \ldots, m+1\}$ and a node w_t for each $t \in \{1, \ldots, n\}$. The source s is connected to every v_ℓ by one arc (s, v_ℓ) with capacity $c(s, v_\ell) = \lambda_\ell$, where the λ_ℓ are defined in Lemma 3. Moreover, there is one arc from each w_t to sink q with capacity $c(w_t, q) = x_t$. Finally, for each $\ell \in \{1, \ldots, m+1\}$ and $t \in \{\ell, \ldots, n\}$ there is an arc (v_ℓ, w_t) with capacity $c(v_\ell, w_t) = h \cdot \lambda_\ell$. This construction is illustrated in Fig. 2.3. Since $(x, z) \in P$, all arc capacities in $G(x, z)$ are non-negative.

Let $c(\delta(S)) = \sum_{(u,v)\in\delta(S)} c(u, v)$ denote the capacity of an $s-q$ cut $\delta(S) = \{(u, v) \in E \mid u \in S, v \in \overline{S}\}$, where $s \in S \subseteq V \setminus \{q\}$ and $\overline{S} = V \setminus S$. The minimum capacity of an $s - q$ cut in $G(x, z)$ is at most 1, as $c(\delta(\{s\})) = \sum_\ell \lambda_\ell = 1$.

Claim 1. *The minimum capacity of an $s-q$ cut in $G(x, z)$ is strictly smaller than 1.*

Proof Assuming the contrary, it follows that there exists a compatible flow f in $G(x, z)$ of value 1, due to the MAX-FLOW MIN-CUT Theorem of Ford and Fulkerson [31]. For each $\ell \in \{1, \ldots, m+1\}$ define a vector $x^\ell \in \mathbb{R}^n$ as

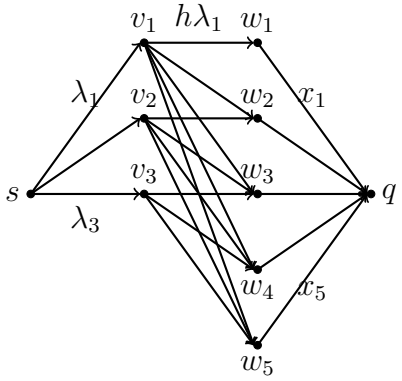

Figure 2.3: The construction of $G(x, z)$.

follows. If $\lambda_\ell > 0$, then let $x_t^\ell = f(v_\ell, w_t)/\lambda_\ell$ for $t \in \{\ell, \ldots, n\}$, and $x_t^\ell = 0$ for $t \in \{1, \ldots, \ell - 1\}$. On the other hand, when $\lambda_\ell = 0$, choose any x^ℓ such that $(x^\ell, z^\ell) \in K$. Hence, every (x^ℓ, z^ℓ) belongs to K and $\sum_\ell \lambda_\ell x_t^\ell \leq x_t$ for each t. But $\sum_t \sum_\ell \lambda_\ell x_t^\ell = 1$, since f is feasible, and $\sum_t x_t = 1$ as $(x, z) \in P$, whence, $\sum_\ell \lambda_\ell x^\ell = x$. Then Lemma 3 ensures that $(x, z) \in K$, a contradiction. □

Let $N(v_\ell) = \{w_t \in V \mid (v_\ell, w_t) \in E\}$ denote the set of nodes reachable from v_ℓ by one arc. If S is a subset of nodes, then let $N_S(v_\ell) = N(v_\ell) \cap S$.

Proposition 2. *Let S be an $s - q$ cut of $G(x, z)$. If $\ell < \kappa$ then $N_{\overline{S}}(v_\kappa) \subseteq N_{\overline{S}}(v_\ell)$.*

Claim 2. *There exists a minimum capacity $s - q$ cut $\delta(S)$ in $G(x, z)$ with the following structure:*

(i) there exists $t_1 \in \{1, \ldots, m\}$ such that $v_\ell \in \overline{S}$ for all $\ell \in \{1, \ldots, t_1\}$ and $v_\ell \in S$ for all $\ell \in \{t_1 + 1, \ldots, m + 1\}$,

(ii) $w_t \in \overline{S}$ for all $t \in \{1, \ldots, t_1\}$,

(iii) for every $v_\ell \in S$, $|N_{\overline{S}}(v_\ell)| \leq p - 1$,

(iv) $|N_{\overline{S}}(v_{t_1+1})| = p - 1$.

Proof Let $\delta(S)$ be a minimum capacity $s-q$ cut of $G(x, z)$. Clearly, $c(\delta(S)) < 1$ holds by assumption. First we claim that there exists $\ell \in \{1, \ldots, m + 1\}$ such that $v_\ell \in S$. Indeed, otherwise the arcs (s, v_ℓ) all leave S, whence $c(\delta(S)) \geq \sum_{\ell=1}^{m+1} c(s, v_\ell) = \sum_{\ell=1}^{m+1} \lambda_\ell = 1$, contradicting $c(\delta(S)) < 1$.

Proof of part (iii): Suppose there exists $v_\ell \in S$ with $|N_{\overline{S}}(v_\ell)| \geq p$. If $\lambda_\ell = 0$, we can replace S by $S - \{v_\ell\}$ without changing the capacity of the

cut. So assume $\lambda_\ell > 0$. From flow theory (see e.g., Ahuja et al. [1]) we know that any maximum value $s-q$ flow f saturates all arcs leaving S, since $\delta(S)$ is of minimum capacity. In particular, $f(v_\ell, w_t) = c(v_\ell, w_t) = h \cdot \lambda_\ell$ holds for all arcs $(v_\ell, w_t) \in E$ with $w_t \in N_{\overline{S}}(v_\ell)$. Using conservation of flow we can derive that

$$\lambda_\ell \geq f(s, v_\ell) = \sum_{(v_\ell, w_t) \in E} f(v_\ell, w_t) \geq |N_{\overline{S}}(v_\ell)| \cdot h \cdot \lambda_\ell.$$

Since $p \cdot h \geq 1$ by the definition of p and $|N_{\overline{S}}(v_\ell)| \geq p$ by assumption, it must be the case that $|N_{\overline{S}}(v_\ell)| = p$ and $p \cdot h = 1$, otherwise there is a contradiction. Hence, we may replace S by $S - \{v_\ell\}$ without changing the capacity of the cut.

Proof of part (i): Since S contains at least one of the nodes v_ℓ, there exists a unique $t_1 \in \{0, \ldots, m\}$ such that $v_\ell \notin S$ for all $\ell \in \{1, \ldots, t_1\}$ and $v_{t_1+1} \in S$. Suppose there exists $\kappa > t_1 + 1$ such that $v_\kappa \notin S$. If $\lambda_\kappa = 0$, we can add v_κ to S yielding a cut with the same capacity. Assume $\lambda_\kappa > 0$. Since $N_{\overline{S}}(v_\kappa) \subseteq N_{\overline{S}}(v_{t_1+1})$ by Property 2 and $|N_{\overline{S}}(v_{t_1+1})| \leq p-1$ by part (iii), we also have $|N_{\overline{S}}(v_\kappa)| \leq p-1$. Since the capacity of the cut determined by $S \cup \{v_\kappa\}$ is

$$\begin{aligned} c(\delta(S \cup \{v_\kappa\})) &= c(\delta(S)) - c(s, v_\kappa) + \sum_{w_t \in N_{\overline{S}}(v_\kappa)} c(v_\kappa, w_t) \\ &\leq c(\delta(S)) - \lambda_\kappa + (p-1) \cdot h \cdot \lambda_\kappa < c(\delta(S)), \end{aligned} \tag{2.10}$$

$\delta(S)$ is not of minimum capacity, which is a contradiction. It remains to show that $t_1 \geq 1$. Suppose $t_1 = 0$. To simplify notation, we introduce dummy z_t symbols with value 0, for $t \in \{m+1, \ldots, n\}$. Then S contains the source node s, all nodes v_ℓ and some of the nodes w_t. Denoting by U the indices t such that $w_t \notin S$, we determine the capacity of $\delta(S)$ below:

$$\begin{aligned} c(\delta(S)) = \sum_{t \in U} \sum_{\ell=1}^{\min(m+1,t)} h \cdot \lambda_\ell + \sum_{t \in \{1,\ldots,n\} - U} x_t = \\ = \sum_{t \in U} (1 - z_t)h + \sum_{t \in \{1,\ldots,n\} - U} x_t \geq \sum_{t \in \{1,\ldots,n\}} x_t, \end{aligned}$$

where we exploited that $\sum_{\ell=1}^{\min(m+1,t)} \lambda_\ell = 1 - z_t$ and that $(1 - z_t)h \geq x_t$, for (x, z) satisfies (2.8b) by assumption. But $\sum_{t=1}^{n} x_t = 1$, hence $c(\delta(S)) \geq 1$, a contradiction.

Proof of part (ii): Suppose there exists $w_t \in S$ with $t \in \{1, \ldots, t_1\}$. Then $(w_t, q) \in E$ is an arc leaving S. Since w_t is not connected to any other node

in S, the capacity of $\delta(S)$ can be decreased by x_t unless $x_t = 0$. Since $\delta(S)$ is of minimum capacity by assumption, it must be the case that $x_t = 0$. However, in this case w_t can be removed from S yielding a cut with the same capacity.

Proof of part (iv): Consider the node v_{t_1}. Since $w_{t_1} \notin S$ by part (ii), it follows that $N_{\overline{S}}(v_{t_1}) = N_{\overline{S}}(v_{t_1+1}) \cup \{w_{t_1}\}$. Consequently, if $|N_{\overline{S}}(v_{t_1+1})| < p-1$, then $|N_{\overline{S}}(v_{t_1})| < p$. If $\lambda_{t_1} = 0$, then $c(\delta(S \cup \{v_{t_1}\})) = c(\delta(S))$, whence we can replace S by $S \cup \{v_{t_1}\}$ and repeat the whole analysis. Assume $\lambda_{t_1} > 0$. Then adding v_{t_1} to S would decrease the capacity of the cut, a contradiction. □

To finish the proof of the theorem, define the sets $S_1 = \{t_1\} \cup \{t \in \{t_1 + 1, \ldots, m\} \mid w_t \in \overline{S}\}$ and $S_2 = \{t \in \{m+1, \ldots, n\} \mid w_t \in \overline{S}\}$, where S is the node set found in Claim 2. Notice that $\emptyset \neq S_1 \subseteq \{1, \ldots, m\}$, $S_2 \subseteq \{m+1, \ldots, n\}$ and $|S_1 \cup S_2| = p$, by part (iv) of Claim 2. To simplify the following presentation, define new symbols z_t with value 0 for all $t \in \{m+1, \ldots, n\}$. Since the capacity of $\delta(S)$ is strictly smaller than 1, we have

$$1 > \sum_{t=1}^{t_1} c(s, v_t) + \sum_{\ell=t_1+1}^{m} \sum_{t \in \{\ell, \ldots, n\} \cap (S_1 \cup S_2)} c(v_\ell, w_t) + \sum_{t \in \{t_1, \ldots, n\} \setminus (S_1 \cup S_2)} c(w_t, q).$$

Using the definition of arc capacities and that of the λ_ℓ, this can be rewritten as

$$\begin{aligned} &= 1 - z_{t_1} + h \sum_{t \in (S_1 \cup S_2) \setminus \{t_1\}} \sum_{\ell=t_1+1}^{t} (z_{\ell-1} - z_\ell) + \sum_{t \in \{t_1, \ldots, n\} \setminus (S_1 \cup S_2)} x_t \\ &= 1 - z_{t_1} + h \sum_{t \in (S_1 \cup S_2) \setminus \{t_1\}} (z_{t_1} - z_t) + \sum_{t \in \{t_1, \ldots, n\} \setminus (S_1 \cup S_2)} x_t. \end{aligned}$$

Since $z_t = 0$ for all $t \in \{m+1, \ldots, n\}$ and $|S_1 \cup S_2| = p$, this is equivalent to

$$= 1 - z_{t_1} + h(p-1)z_{t_1} - h \sum_{t \in S_1 \setminus \{t_1\}} z_t + \sum_{t \in \{t_1, \ldots, n\} \setminus (S_1 \cup S_2)} x_t.$$

Now, as $-z_{t_1} + h(p-1)z_{t_1} = -h_{\text{rem}} z_{t_1}$, it follows that (2.8g) is violated for the sets (S_1, S_2), that is, $(x, z) \notin P$, a contradiction. □

Facets of K

In order to find a minimal linear representation of polytope K, it suffices to consider only the system (2.8a)-(2.8g), by Theorem 1. To show that a valid inequality $\alpha x + \beta z \leq \gamma$ from (2.8b)-(2.8g) represents a facet of K we will use the standard proof technique consisting in exhibiting a point (x, z) in K such that $\alpha x + \beta z = \gamma$ and (x, z) satisfies all other inequalities with strict inequality, see e.g., Schrijver [94].

Lemma 5. *[64]* The inequalities (2.8b), (2.8d), (2.8f) always represent facets of K. The inequalities (2.8c) represent facets iff $h < 1$. For $1 \leq t \leq n$ inequality (2.8e)$_t$ represents a facet iff $h > 1/(n-1)$ and if $t \geq m+1$, $p < n-m$. Finally, for each $\emptyset \neq S_1 \subseteq \{1,\ldots,m\}$, $S_2 \subset \{m+1,\ldots,n\}$ with $|S_1|+|S_2| = p$ the corresponding inequality in (2.8g) represents a facet unless $t_1 = 1$ and $h = 1/p$.*

Proof Firstly, we show that there exists a point in K satisfying all inequalities (2.8b)-(2.8g) with strict inequality. One may verify that if $\varepsilon > 0$ is a sufficiently small positive number, the vector (x, z) given by

$$x_t = 1/n,\ t \in \{1,\ldots,n\} \text{ and } z_t = \varepsilon^t,\ t \in \{1,\ldots,m\} \tag{2.11}$$

is in K satisfying every inequality in (2.8b)-(2.8g) with strict inequality.

The construction of an appropriate point in K for (2.8b), (2.8d) and (2.8f) being straightforward, we turn directly to the inequalities (2.8c). If $h = 1$ then $x_t = 1$ implies $x_{t'} = 0$ for all $t' \in \{1,\ldots,n\} - \{t\}$, proving the necessity of the condition. Conversely, if $h < 1$, the vector (x, z) with $x_t = h$, $x_{t'} = (1-h)/(n-1)$, $t' \in \{1,\ldots,n\} - \{t\}$, and z as in (2.11) satisfies $x_t = h$ with equality, while all other inequalities are strict if ε is sufficiently small.

Concerning (2.8e), if $h = 1/(n-1)$ then $x_t = 0$ implies $x_{t'} = h$ for all $t' \in \{1,\ldots,n\} - \{t\}$, proving the necessity of the condition. Moreover, if $t \geq m+1$ and $m+p = n$, then $x_t \geq 0$ is implied by the (S_1, S_2) inequality with $t_1 = m$, $S_1 = \{m\}$, $S_2 = \{m+1,\ldots,n\} - \{t\}$. Now suppose $h > 1/(n-1)$. The vector (x, z) with $x_t = 0$, $x_{t'} = 1/(n-1)$, $t' \in \{1,\ldots,n\} - \{t\}$, and z as in (2.11) satisfies $x_t = 0$ with equality while all other inequalities are strict if ε is sufficiently small and when $t \geq m+1$, $m+p < n$.

Finally, consider an inequality in (2.8g). If $t_1 = 1 \in S_1$ and $h = 1/p$, then this inequality is implied by the inequalities $x_t \leq h(1-z_t)$, $t \in S_1$, and $x_t \leq a$, $t \in S_2$. Namely, since $h = 1/p$, $h_{\text{rem}} = h$ follows. Therefore, we have

$$\begin{aligned}\sum_{t\in S_1} az_t &\leq \sum_{t\in S_1} hz_t + \sum_{t\in S_2}(h - x_t) \leq \sum_{t\in S_1}(h-x_t) + \sum_{t\in S_2}(h-x_t)\\ &= hp - \sum_{t\in S_1\cup S_2} x_t = 1 - \sum_{t\in S_1\cup S_2} x_t = \sum_{\{1,\ldots,n\}-(S_1\cup S_2)} x_t.\end{aligned}$$

On the other hand, when $t_1 \geq 2$ or $hp > 1$, we will define a point (x, z) in K such that

$$h_{\text{rem}} z_{t_1} + h \sum_{t\in S_1-\{t_1\}} z_t = \sum_{t\in\{t_1,\ldots,n\}-(S_1\cup S_2)} x_t \tag{2.12}$$

while all other inequalities will be strict. Let ε and ϕ be small positive numbers to be chosen later. Denote $U = \{t_1,\ldots,n\} - (S_1 \cup S_2)$ and $k =$

$|U| = n - t_1 + 1 - p$. Notice that $1 \notin U$ even if $t_1 = 1$. Therefore, $k \leq n-1$. Finally, $k \geq 1$, since $t_1 \leq m$, $|S_1 \cup S_2| = p$ and $m + p \leq n$. Observe that $h(n-k) = h(t_1 - 1 + p) > 1$, since $t_1 \geq 2$ or $hp > 1$ by assumption.

Define z as in (2.11), while let x be given by

$$x_t = \begin{cases} \phi, & t \in U \\ (1 - k\phi)/(n-k), & t \in \{1, \ldots, n\} - U. \end{cases}$$

Now, by substituting the definitions for x and z in (2.12) we obtain the following relation between ε and ϕ:

$$h_{\text{rem}}\varepsilon^{t_1} + h \sum_{t \in S_1 - \{t_1\}} \varepsilon^t = k\phi. \tag{2.13}$$

Observe that the left hand side is polynomial in terms of ε and that all coefficients are positive. Consequently, the left hand side is a monotone increasing and continuous function of ε. Hence, for any $\phi > 0$ there exists a unique $\varepsilon > 0$ satisfying eq. (2.13) and vice versa.

Clearly, (x, z) satisfies equation (2.8a) and the inequalities (2.8d)-(2.8f) with strict inequality. Concerning (2.8c), for each $t \in U$, $x_t < a$ if ϕ is small. If $t \in \{1, \ldots, n\} - U$, then $x_t = (1-k\phi)/(n-k) < 1/(n-k) = 1/(t_1 - 1 + p) \leq 1/p \leq a$. Checking (2.8b) is a bit tricky. For each $t \in U$ strict inequality holds if ε and ϕ are sufficiently small. It remains to show that for each $t \notin U$, $x_t = (1 - k\phi)/(n-k) < h(1 - \varepsilon^t) = h(1 - z_t)$. This can be rewritten as $h(n-k)\varepsilon^t < h(n-k) - 1 + k\phi$. Since $h(n-k) > 1$, any $\varepsilon < 1 - 1/(h(n-k))$ will do for arbitrary ϕ.

Now consider (2.8g) for any $(S_1', S_2') \neq (S_1, S_2)$. Let $U' = \{t_1', \ldots, n\} - (S_1' \cup S_2')$. Since $|S_1'| + |S_2'| = p = |S_1| + |S_2|$, at least one of the sets $U - U'$ and $U' - U$ is not empty. First suppose $U' - U$ is not empty. The left hand side of (2.8g) on (S_1', S_2') is at most

$$h_{\text{rem}}\varepsilon + h \sum_{t=2}^{\min(p,m)} \varepsilon^t. \tag{2.14}$$

We claim that the right hand side is greater than (2.14) if ε is sufficiently small. Namely, choose $t' \in U' - U$ arbitrarily. As $t' \notin U$, $x_{t'} = (1 - k\phi)/(n-k)$. Since $(1-k\phi)/(n-k) \geq 1/n$ when $0 \leq \phi \leq 1/n$, it follows that $x_{t'} \geq 1/n$ for ϕ sufficiently small. Hence, the right hand side of (2.8g) on (S_1', S_2') is at least $1/n$. So, choose $\varepsilon > 0$ independently from the particular (S_1', S_2') so that the quantity (2.14) is strictly smaller than $1/n$. The chosen ε determines ϕ by (2.13). If $\phi > 1/n$, decrease ϕ and proportionally also ε.

If $U' - U$ is empty, then $U - U'$ cannot be empty. These conditions along with $|S_1| + |S_2| = p = |S_1'| + |S_2'|$ imply that $t_1 < t_1'$. Since $\varepsilon^{t_1'}/\varepsilon^{t_1}$ can be made arbitrarily close to 0 by decreasing ε, the left hand side of (2.8g) on (S_1', S_2') can be made smaller than ϕ while maintaining eq. (2.13). Since the right hand side of (2.8g) is at least ϕ, we have shown that the inequality is strict.

Finally, our demonstration also shows that every facet inducing inequality in (2.8b)-(2.8g) represents a distinct facet of K. □

A minimal linear representation of K_{jk}

To obtain a minimal linear representation of K_{jk} we first determine one for K_{j*} and K_{*k}, respectively, as follows. Define the polytope K with respect to K_{j*} (using appropriate substitutions). Take the minimal linear representation of K and convert it back to one for K_{j*}, let $A^{j*}x_j + B^{j*}z_j \leq b^{j*}$ denote this system. We can get a minimal linear representation $A^{*k}x_k + B^{*k}\tilde{z}_j \leq b^{*k}$ for K_{*k} by a similar procedure. We will show that almost every inequality in the combined system $A^{j*}x_j + B^{j*}z_j \leq b^{j*}$, $A^{*j}x_k + [0, B^{*k}]z_j \leq b^{*k}$ represents a facet of K_{jk}.

First observe that (2.8f) becomes $1 - z_{j,r_j+p_j} \geq 0$ in polytope K_{j*} and it becomes $z_{j,d_j} \geq 0$ in polytope K_{*k}. Clearly $z_{j,d_j} \geq 0$ is implied by the facet-representing inequalities $x_{j,d_j} \geq 0$ and $x_{j,d_j} \leq h_j z_{j,d_j}$ for K_{j*}. Therefore, $z_{j,d_j} \geq 0$ is superfluous in the linear representation of K_{jk}. Now consider $z_{j,r_j+p_j} \leq 1$. If $r_j + p_j = r_k$, then this inequality is implied by the facet-representing inequalities $x_{k,d_k} \geq 0$ and $x_{k,r_k} \leq h_k(1 - z_{j,r_k})$ for K_{*k}. As a by-product, $z_{j,r_k} \leq 1$ never represents a facet of K_{jk}. We have the following:

Theorem 2. *[64]* *The system $A^{j*}x_j + B^{j*}z_j \leq b^{j*}$, $A^{*k}x_k + [0, B^{*k}]z_j \leq b^{*k}$ without $z_{j,d_j} \geq 0$ and without $z_{j,r_j+p_j} \leq 1$ if $r_j + p_j = r_k$, is a minimal linear representation of K_{jk}.*

Proof By Corollary 2, the linear system in the statement represents K_{jk}. Now consider e.g., a facet representing inequality $\alpha x_j + \beta z_j \leq \beta_0$ for K_{j*}. First suppose $\alpha x_j + \beta z_j \leq \beta_0$ is not one among (2.7e). There exists a point $(x_j, z_j) \in K_{j*}$ satisfying this inequality with equality while all other inequalities in the minimal linear representation of K_{j*} are strict. In particular, $z_{jt} > z_{j,t+1}$ for $t \in \{r_k, \ldots, d_j - 1\}$ by assumption and w.l.o.g. $z_{j,d_j} > 0$, since this inequality does not represent a facet of K_{j*}. By the same token, we may also assume that $z_{j,r_k} < 1$. We have to show that there exists a point $(x_k, \tilde{z}_j) \in K_{*k}$, where $\tilde{z}_{jt} = z_{jt}$ for each $t \in \{r_k, \ldots, d_j\}$, such that all inequalities in the minimal linear representation of K_{*k} are strict. To this end, we construct a point in the polytope K which can be transformed to

a point in K_{*k} with the desired properties. Let $m = d_j - r_k + 1$, $n = s_k$, $\lambda_1 = 1 - z_{j,r_k}$, $\lambda_\ell = z_{j,\ell+r_k-2} - z_{j,\ell+r_k-1}$, $\ell \in \{2,\ldots,m\}$, and $\lambda_{m+1} = z_{j,d_j}$. We clearly have $\sum_{\ell=1}^{m+1} \lambda_\ell = 1$, and $\lambda_\ell > 0$ for each ℓ. Define the vectors $x^\ell \in \mathbb{R}^n$ as follows:

$$x_t^\ell = \begin{cases} 0 & \text{if } 1 \le t \le \ell - 1 \\ 1/(n-\ell+1) & \text{if } \ell \le t \le n \end{cases} \qquad \ell \in \{1,\ldots,m+1\}.$$

Hence, $(x^\ell, z^\ell) \in K$ for each ℓ. Moreover, for each inequality in the system (2.8b)-(2.8g) there exists at least one vector (x^ℓ, z^ℓ) on which that inequality is strict, as one may verify. Therefore, $(x, z) = \sum_\ell \lambda_\ell (x^\ell, z^\ell)$ is a point in K such that all inequalities on it are strict, since each $\lambda_\ell > 0$.

The cases when $\alpha x_j + \beta z_j \le \beta_0$ is one among (2.7e) (in which case it represents a facet of both K_{j*} and K_{*k}) or $\alpha x_k + \beta \tilde{z}_j \le \beta_0$ represents a facet of K_{*k} can be handled similarly. □

2.1.8 Implementation and computational evaluation

We implemented two branch-and-cut algorithms, B^+ and B^-, for solving the mixed integer-linear program (2.4). We assume familiarity with this technique, for an introduction see e.g. Padberg and Rinaldi [88] and Jünger et al. [55]. The LP relaxation of RCPSVP is obtained from the strong formulation by relaxing the constraint (2.4h) to $0 \le z_{jt} \le 1$, $\forall\, i, t$. The algorithm B^+ uses the following classes of valid inequalities to cut off a solution (x, y, z) of the LP relaxation with fractional z:

(i) To each pair of activities $(j,k) \in \mathcal{E}$ corresponds a polytope K_{jk}. The algorithm checks whether $(x_j, x_k, z_j) \in K_{jk}$ by finding the most violated (S_1, S_2) inequalities for K_{j*} and also for K_{*k}.

(ii) Flow cover inequalities. These inequalities are defined in [89]. Although they are not needed to describe the polytopes K_{j*} and K_{*k}, the constraints (2.4e) along with (2.4bc), (2.4g) and (2.4h) give rise to variable upper bound flow problems.

(iii) Gomory fractional cuts, see e.g., Nemhauser and Wolsey [83].

Algorithm B^- is identical to B^+ except it does not separate class (i) inequalities. Both algorithms start from scratch, i.e., without setting any initial upper bound.

We coded algorithms B^+ and B^- in C++ using the ILOG CPLEX 7.5 branch-and-cut solver [54]. Besides modeling the problem and calling the solver, we coded only the separation algorithm described in Section 2.1.6 for

class (i) and we let the solver find violated inequalities in classes (ii) and (iii). The LPs were solved by the built-in dual-simplex method and we also used the built-in heuristic to find feasible solutions. In B^+ the separation algorithm for class (i) inequalities was called in the root and then in every fifth node during the search. The numerical tables of this section can be found in Section 5.1.

Comparison to Branch-and-Price

In this section we compare and evaluate three algorithms: the branch-and-price algorithm of Hans [47], denoted by H, and the algorithms B^+ and B^-. Hans evaluated his method on the benchmark instances of De Boer [24]. Each instance consists of one project only whose precedence graph was generated by the randomized procedure of Kolisch et al. [71]. The maximal intensity h_j of each activity j is given by $1/p_j$, where p_j was chosen randomly between 1 and 5. Moreover, every activity may require up to 5 distinct resources. In each time period the internal capacity of each resource is finite, and its external capacity is infinite with cost uniformly 1. The instances are subdivided into classes characterized by common parameter values such as the number of activities in the project n, the number of resources r and the average slack $s = \sum_{i=1}^{n}(s_j - p_j)/n$. For each combination of the parameter values $n = 10, 20, 50$, $r = 3, 10, 20$ and $s = 2, 5, 10, 15, 20$, ten random instances were generated yielding a total of 450 instances.

The computing environment of Hans was a PC with a 600 MHz Pentium 3 processor, Windows 2000 operating system, Borland Delphi 5 programming language and CPLEX 7.0. Hans stopped the algorithm after 1800 seconds. We performed the tests on a PC with a 1.6 GHz Pentium 4 processor under Windows 2000, and terminated the search after 675 seconds. The computational results of algorithm H reported here are slightly better than those in [47], since we used the latest data (fall of 2003) from E. Hans [48]. In the following $ub(A)$ and $lb(A)$ denote the best upper and lower bound, respectively, obtained by algorithm A, where A is one of H, B^+ and B^-.

Table 5.1 shows how often the three algorithms found provably optimal solutions. For each class of instances the upper number indicates the number of times algorithm H proved optimality, while the other two numbers show the performance of B^+ and B^-, respectively. Notice that either variant of our branch-and-cut algorithm proved optimality in considerably more cases than algorithm H and we got better results with B^+.

Table 5.2 summarizes the average $ub(B^+)/ub(H)$ ratio over the ten instances in each class, and also, when different, the average $ub(B^-)/ub(H)$ ratio. In almost all classes B^+ gives at least as good results as algorithm H

on average, the only exception being when $n = 50$, $r = 20$ and $s = 15$, but B^- is inferior to H in four classes consisting of hard instances. Moreover, both B^+ and B^- improved on the best upper bounds in several cases. Comparing this table to Table 5.1 reveals that algorithm H found the optimum in more cases than it was able to prove optimality.

The performance of our branch-and-cut algorithms can also be measured by the ratio of the lower bound to the upper bound on every instance. Table 5.3 depicts for each class the average $lb(B^+)/ub(B^+)$ ratio and also, when different, the average $lb(B^-)/ub(B^-)$ ratio. In either case the ratio decreases when both the number of activities and the average slack tend to be high. However, the gap between the lower and upper bounds is significantly bigger when the (S_1, S_2) inequalities are not used.

Finally, some details of the computations are provided in Table 5.4. For technical reasons we give this data for algorithm B^+ only. There are three groups corresponding to the three problem sizes in terms of the number of activities. Averages are taken over the 150 instances constituting a group.

In summary, our algorithms proved optimality in considerably more cases than branch-and-price and we also improved on the previously known upper bounds. Finally, it is advantageous to use (S_1, S_2) inequalities especially when solving hard instances with a large number of activities and large slack.

Results on Tavares' instances

Tavares [99] proposed a non-linear program for the makespan minimization problem (cf. Section 2.1.4). In his book, computational results are reported for two test cases (Case A and Case B) with the following main characteristics. There is only one resource with finite constant internal capacity and with no external capacity. For each activity j, $r_j = 0$, $h_j = 1/p_j$, where p_j is the given integer minimum duration of activity j, and $a_{1,j} = 10p_j$. The precedences between the activities are given by an acyclic directed graph.

In Case A there is a project network with 75 activities. However, the same network gives rise to a series of problem instances by varying the capacity of the single resource. The tested values are 120, 100, 80, 75, 70, 65 giving rise to six distinct problem instances. In Case B the project network consists of 150 activities and there is a problem instance for each of the resource capacity limits 240, 200, 180, 120. For more details of the instance generation we refer to [100].

A lower bound on the minimum makespan is the length of the longest path in the project network, denoted by lb_p, which is computed after fixing the activity durations to the given minimum values. Clearly, this bound does not depend on the resource capacity limit. For all Case A instances

$lb_p = 27$, whereas for all Case B instances $lb_p = 21$. A *resource based lower bound*, lb_r, can be computed by dividing the total resource requirement of the activities by the capacity of the resource, which was communicated to us by J. Coelho [21]. Clearly, $lb_{\max} = \max\{lb_p, lb_r\}$ is a valid lower bound on the makespan.

We solved the makespan minimization problem by a binary search procedure with run-time limits $\tau_1 = 3600$ and $\tau_2 = 600$ seconds. Table 5.5 summarizes our computational results on Case A and Case B, respectively, and also compare them to those of Tavares. Column rcap indicates the resource capacity, ub(Tav) is the upper bound obtained by Tavares [99] (the starred values are optimal) and *Opt* is our upper bound which was always optimal. Some details of computations are given in terms of the CPU time (in seconds), the total number of Gomory fractional cuts added (#frac cut), and the total number of (S_1, S_2) cuts added, $\#(S_1, S_2)$. Since CPLEX never generated any flow cover cuts, we do not indicate their number. In every case algorithm B^+ found a feasible solution for $T = lb_{\max}$ which, therefore, was always optimal. In addition, the optimal solution was always found already in the root node of the branch-and-bound tree. For all but the last instance in Case A, J. Coelho obtained the same bounds as ours by using metaheuristics [21]. However, we improved on the upper bounds of Tavares in five out of ten cases in a short computation time.

Evaluation on PSPLIB

We also evaluated our makespan minimization procedure on the well-known PSPLIB instances that were originally designed for the non-preemptive resource constrained project scheduling problem [70]. In each PSPLIB instance there are 4 resources each having a finite constant internal capacity and no external capacity. Each activity j has a deterministic duration p_j, which we interpret as the inverse of the maximum intensity of activity j, that is, let $h_j = 1/p_j$. In each time period of execution, activity j requires ρ_{ij} units of resource i. Therefore, we set $a_{ij} = \rho_{ij} p_j$, for each resource i.

We tested our algorithm on 30-activity ($j30$) and 60-activity ($j60$) instances, respectively. On the $j30$ instances the run-time limit of the two phases were set as $\tau_1 = \tau_2 = 300$ seconds, whereas on $j60$ instances we set $\tau_1 = 1800$, $\tau_2 = 600$ seconds. We divide the $j30$ as well as $j60$ instances into two subclasses: those on which the total CPU time (through all invocations of B^+) of the algorithm was less than 60 seconds and the rest. Thus we have four classes: $j30^-$, $j60^-$ (less than 60 seconds total CPU time), $j30^+$, $j60^+$ (more than 60 seconds total CPU time). Every $j30$ instace has been solved, i.e., a feasible solution has been found, but fourteen $j60^+$ instances have re-

mained unsolved. Table 5.6 has four rows corresponding to these subclasses. The second column indicates the number of instances in the class, and, when different, the number of instances solved. Moreover, the third and fourth columns indicate the average as well as minimum ratio of the strongest lower bound to the best upper bound over all solved instances in the class. We measured the total CPU time and gathered the total search tree nodes, and the total number of flow, fractional and (S_1, S_2) cuts added through all invocations of B^+ while solving each instance. The averages of these data over all solved instances in a class are given in the next five columns of the table. Notice that the majority of instances were solved to optimality in less than 3 seconds. On hard instances the computation time can be 1 hour or more, but was never more than 1.5 hours.

2.2 RCPSVP with feeding precedence constraints

Feeding precedence constraints extend traditional ones by allowing partial overlap between pairs of activities. Such a constraint can model e.g. the flow of material or information between pairs of activities. It is described by a triple (j, k, ϕ), where j and k are activities, and ϕ is a rational parameter from the set $[0, 1]$. Let x_{jt} and x_{kt} denote the intensity of activity j and k, respectively, in time period t. Then the constraint is satisfied if (i) activity k starts only after an ϕ fraction of activity j is completed, i.e., $\sum_{\tau=r_k}^{t} x_{k\tau} = 0$ unless $\sum_{\tau=r_j}^{t-1} x_{j\tau} \geq \phi$, and (ii) for every time period t, the total fraction of activity j until t is at least the total fraction completed of k until t, i.e., $\sum_{\tau=r_j}^{t} x_{j\tau} \geq \sum_{\tau=r_k}^{t} x_{k\tau}$. This concept is illustrated in Figure 2.4.

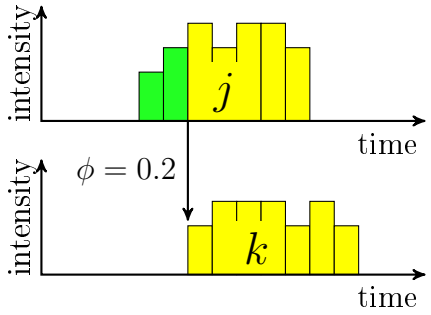

Figure 2.4: Variable intensity activities connected by a feeding precedence constraint.

In order to model feeding precedence constraints we have to modify the MIP formulation (2.4). First of all, the set variables z_{jt}, $t \in \{r_j + p_j, \ldots, d_j\}$,

has to be replaced by new sets of variables z^{ϕ}_{jt}, $t \in \{r_j + p^{\phi}_j, \ldots, d_j\}$, for each distinct values of $\phi \in F_j := \{\phi_{jk} \mid \exists k \text{ such that } (j,k,\phi_{jk}) \in \mathcal{E}\}$, and $p^{\phi}_j := \lceil \phi / h_j \rceil$ is the minimum number of time periods to complete an ϕ fraction of activity j. The meaning of z^{ϕ}_{jt} is the following:

$$z^{\phi}_{jt} = \begin{cases} 1 & \text{if less than an } \phi \text{ fraction of activity } j \text{ is processed} \\ & \text{up to period } t. \\ 0 & \text{otherwise.} \end{cases}$$

After these preliminaries, the MIP model for RCPSVP with feeding precedence constraints is

$$\min \sum_{i \in \mathcal{R}} \sum_{t=1}^{T} c_{it} y_{it} \tag{2.15a}$$

subject to

$$\sum_{t=r_j}^{d_j} x_{jt} = 1, \qquad \forall j \in \mathcal{J}, \tag{2.15b}$$

$$\sum_{t=r_j}^{\ell-1} x_{jt} \geq \phi(1 - z^{\phi}_{j\ell}), \qquad \begin{array}{l} \forall j \in \mathcal{J},\ \phi \in F_j, \\ \ell \in \{r_j + p^{\phi}_j, \ldots, d_j\}, \end{array} \tag{2.15c}$$

$$x_{kt} \leq h_k(1 - z^{\phi}_{jt}), \qquad \forall j \in \mathcal{J},\ (j,k,\phi) \in \mathcal{E}, \tag{2.15d}$$

$$\sum_{t=r_j}^{\ell} x_{jt} \geq \sum_{t=r_k}^{\ell} x_{kt}, \qquad \begin{array}{l} \forall j \in \mathcal{J}, (j,k,\phi) \in \mathcal{E}, \\ \ell \in \{\max\{r_j, r_k\}, \ldots, \min\{d_j, d_k\}\}, \end{array} \tag{2.15e}$$

$$z^{\phi}_{jt} \geq z^{\phi}_{j,t+1}, \qquad \begin{array}{l} \forall j \in \mathcal{J},\ \phi \in F_j, \\ t \in \{r_j + p^{\phi}_j, \ldots, d_j - 1\}, \end{array} \tag{2.15f}$$

$$\sum_{j: r_j \leq t \leq d_j} a_{ij} \cdot x_{jt} \leq b_{it} + y_{it}, \quad \forall i \in \mathcal{R}, t \in \{1, \ldots, T\}, \tag{2.15g}$$

$$0 \leq x_{jt} \leq h_j, \qquad \forall j \in \mathcal{J},\ t \in \{r_j, \ldots, d_j\}, \tag{2.15h}$$

$$0 \leq y_{it} \leq \bar{b}_{it}, \qquad \forall i \in \mathcal{R},\ t \in \{1, \ldots, T\}, \tag{2.15i}$$

$$z^{\phi}_{jt} \in \{0,1\}, \qquad \begin{array}{l} \forall j \in \mathcal{J},\ \phi \in F_j, \\ t \in \{r_j + p^{\phi}_j, \ldots, d_j\}. \end{array} \tag{2.15j}$$

The main difference between the two formulations (2.4) and (2.15) lies in the modeling of precedence constraints. In particular, (2.15c) ensures that $z^{\phi}_{j\ell} = 1$ as long as $\sum_{t=r_j}^{\ell-1} x_{jt} < \phi$.

2.2.1 Preprocessing

First, when $0 \in F_j$, there is no need for the variables z_{jt}^0, and these variables along with all constraints involving any of them can be eliminated. A further reduction is possible by using the earliest and latest start times of the activities, respectively. These parameters can be computed, as usual, by first determining a topological order of the activities with respect to the precedence constraints and then scanning the list forward and backward, respectively. In the forward pass the *earliest start time*, $est(k)$, of activity k is determined by the formula:

$$est(k) = \max\{r_k, \max\{est(j) + p_j^\phi \mid \exists j \in \mathcal{J},\ (j,k,\phi) \in \mathcal{E}\}\}.$$

Let $p_j = \lceil 1/h_j \rceil$. The *latest start time*, $lst(j)$, of activity j is computed in the backward pass as follows:

$$lst(j) = \min\{d_j - p_j + 1, \min\{lst(k) - p_j^\phi \mid \exists k \in \mathcal{J},\ (j,k,\phi) \in \mathcal{E}\}\}.$$

Using the values computed above, the release times and the deadlines of the activities can be tightened. Namely, for any activity j, r_j can be increased to $est(j)$. Moreover, d_j can be decreased to $lst(j) + p_j$, provided that there exists $(j,k,\phi) \in \mathcal{E}$ with $\phi = 1.0$, in which case $p_j^\phi = p_j$ and the activity must complete by $lst(j) + p_j$. In addition, the variables z_{jt}^ϕ has to be defined only for $t \in \{r_j^\phi, \ldots, d_j^\phi\}$, where $r_j^\phi = est(j) + p_j^\phi$ and $d_j^\phi = lst(j) + p_j^\phi - 1$. If $\phi < 1.0$, we add the constraint $\sum_{t=r_j}^{d_j^\phi} x_{jt} \geq \phi$ to the model.

Since $est(j)$ and $lst(j)$ can be computed in linear time, the time complexity of the reduction is linear in the size of the problem.

A necessary condition for the existence of a feasible solution is that $est(j) \leq lst(j)$ for all $j \in \mathcal{J}$. This condition is also sufficient when all y_{it} are unbounded. Therefore, preprocessing may provide important information about the feasibility status of the problem at hand.

2.2.2 Valid inequalities

In order to generalize the results of Section 2.1 to feeding precedence constraints, firstly we adapt the system (2.7), which was the starting point of strengthening the LP relaxation. Let $(j,k,\phi) \in \mathcal{E}$ be a feeding precedence

constraint. The corresponding part of (2.15) is

$$\sum_{t=r_j}^{d_j} x_{jt} = 1, \tag{2.16a}$$

$$0 \le x_{jt} \le h_j, \quad \forall t \in \{r_j, \dots, d_j\} \tag{2.16b}$$

$$\sum_{\tau=r_j}^{t-1} x_{j\tau} \ge \phi \cdot (1 - z^\phi_{jt}), \quad \forall t \in \{r^\phi_j, \dots, d^\phi_j\} \tag{2.16c}$$

$$z^\phi_{jt} \ge z^\phi_{j,t+1}, \quad \forall t \in \{r^\phi_j, \dots, r_k - 1\} \tag{2.16d}$$

$$z^\phi_{jt} \ge z^\phi_{j,t+1}, \quad \forall t \in \{r_k, \dots, d^\phi_j - 1\} \tag{2.16e}$$

$$\sum_{t=r_k}^{d_k} x_{kt} = 1, \tag{2.16f}$$

$$0 \le x_{kt} \le h_k \cdot (1 - z^\phi_{jt}), \quad \forall t \in \{r_k, \dots, d^\phi_j\} \tag{2.16g}$$

$$0 \le x_{kt} \le h_k, \quad \forall t \in \{d^\phi_j + 1, \dots, d_k\} \tag{2.16h}$$

$$\sum_{t=r^i}^{\ell} x^i_t \ge \sum_{t=r^j}^{\ell} x^j_t, \quad \forall \ell \in \{\max\{r^i, r^j\}, \dots, \min\{d^i, d^j\}\}. \tag{2.16i}$$

We define the polytopes K^ϕ_{j*} and K^ϕ_{*k} analogously to K_{j*} and K_{*k} (defined in Section 2.1.6) as follows:

$$K^\phi_{j*} = \operatorname{conv}\left\{(x_j, z_j) \in \mathbb{R}^{s_j} \times \{0,1\}^{d^\phi_j - r^\phi_j + 1} \mid (x_j, z_j) \text{ satisfies } (2.16a) - (2.16e)\right\},$$
$$K^\phi_{*k} = \operatorname{conv}\left\{(x_k, \tilde{z}_j) \in \mathbb{R}^{s_k} \times \{0,1\}^{d^\phi_j - r_k + 1} \mid (x_k, \tilde{z}_j) \text{ satisfies } (2.16e) - (2.16h)\right\}.$$

Analogously to Lemma 2, one can prove the following, more general, result:

Lemma 6. *[65]* $(x_j, x_k, z^\phi_j) \in K^\phi_{jk}$ *if and only if* $(x_j, z^\phi_j) \in K^\phi_{j*}$, $(x_k, \tilde{z}^\phi_j) \in K^\phi_{*k}$, $z^\phi_{jt} = \tilde{z}^\phi_{jt}$ *for* $t \in \{r_k, \dots, d^\phi_j\}$, *and* (x_j, x_k, z^ϕ_j) *satisfies ineq. (2.16i).*

Notice that the definitions of K_{*k} and that of K^ϕ_{*k} are almost identical, the only difference being that in the latter we use the variables z^ϕ_{jt} instead of the z_{jt}. In contrast, the definition of K_{j*} and K^ϕ_{j*} differ significantly. So, we focus on K^ϕ_{j*}. In fact, K^ϕ_{j*} is equivalent to the polyhedron K^ϕ, which is the convex hull of all the vectors $(x, z) \in \mathbb{R}^n \times \{0,1\}^{m-p}$ satisfying the following

conditions

$$\sum_{t=1}^{n} x_t = 1, \tag{2.17a}$$

$$\sum_{\tau=1}^{t-1} x_\tau \geq \phi \cdot (1 - z_t), \quad \forall t \in \{p+1, \ldots, m\} \tag{2.17b}$$

$$\sum_{\tau=1}^{m} x_\tau \geq \phi \tag{2.17c}$$

$$z_t \geq z_{t+1}, \quad \forall t \in \{p+1, \ldots, m-1\} \tag{2.17d}$$

$$0 \leq x_t \leq h, \quad \forall t \in \{1, \ldots, n\}, \tag{2.17e}$$

where $h = h_j$, $p = p_j^\phi$, $m = d_j^\phi - r_j + 1$, and $n = d_j - r_j + 1$. In any vertex $(\hat{x}, \hat{z})$ of K^ϕ, $\hat{z}$ is equal to precisely one of the following vectors in $\{0,1\}^{m-p}$:

$$z_t^\ell = \begin{cases} 1, & t \in \{p+1, \ldots, \ell\}, \\ 0, & t \in \{\ell+1, \ldots, m\}, \end{cases} \qquad \ell \in \{p, \ldots, m\}.$$

A generalization of Lemma 3 can be easily derived:

Lemma 7. *[65]*A vector $(x, z) \in \mathbb{R}^n_+ \times \mathbb{R}^{m-p}_+$ belongs to K^ϕ if and only if the scalars*

$$\lambda_\ell = \begin{cases} 1 - z_{p+1} & \ell = p, \\ z_\ell - z_{\ell+1}, & \ell \in \{p+1, \ldots, m-1\}, \\ z_m, & \ell = m. \end{cases}$$

are all non-negative, and there exist vectors $x^\ell \in [0,h]^n$, $\forall \ell \in \{p, \ldots, m\}$, such that $\sum_{t=1}^{\ell} x_t^\ell \geq \phi$, $\sum_{t=1}^{n} x_t^\ell = 1$, and $\sum_{\ell=p}^{m} \lambda_\ell (x^\ell, z^\ell) = (x, z)$.

In order to find a linear representation of K^ϕ, we define a new network flow model $G^\phi(x, z)$. The set of nodes contains a unique source s, a unique sink q, the pairs of nodes v_t^1, v_t^2 for $t = p, \ldots, m$, and the nodes w_t for $t \in \{1, \ldots, n\}$. There is an arc from source s to each node v_ℓ^1 with capacity $c(s, v_\ell^1) = \lambda_\ell$, $\ell = p, \ldots, m$. There is an arc from v_ℓ^1 to each w_t with $1 \leq t \leq \ell$, having capacity $c(v_\ell^1, w_t) = h \cdot \lambda_\ell$, and an arc (v_ℓ^1, v_ℓ^2) with capacity $c(v_\ell^1, v_\ell^2) = \lambda_\ell(1 - f)$. v_ℓ^2 is connected to each w_t with $\ell + 1 \leq t \leq n$ with an arc of capacity $c(v_\ell^2, w_t) = h \cdot \lambda_\ell$. Finally, each w_t is connected to sink q with an arc of capacity x_t.

Using Lemma 7, we can show the following result:

Lemma 8. *[65]*$(x, z) \in K^\phi$ if and only if (x, z) satisfies (2.17a), (2.17d), (2.17e), and the minimum capacity of an $s - q$ cut in $G^\phi(x, z)$ is 1.*

Consequently, we can apply the same technique as in Section 2.1 to derive a linear representation of K^ϕ. Let $h_r = \phi - (p-1)\cdot h$.

Theorem 3. *[65]* *K^ϕ equals the set of vectors (x,z) in $[0,h]^n \times [0,1]^{m-p}$ that satisfy (2.17a)-(2.17e), and the following linear constraints:*

$$h_r z_{t_1} + h \sum_{t\in S_1\setminus\{t_1\}} z_t + \sum_{t\in\{1,\ldots,t_1\}\setminus(S_1\cup S_2)} x_t \;\geq\; \phi - h|S_2|, \qquad (2.17f)$$

for every $\emptyset \neq S_1 \subseteq \{p+1,\ldots,m\}$, $S_2 \subset \{1,\ldots,p\}$, $|S_1|+|S_2| = p$, and t_1 being the greatest element of S_1;

$$(\phi - 1 + h|U_\ell^2|)z_\ell + h\sum_{t\in U_\ell^1} z_t + \sum_{t\in\{1,\ldots,n\}\setminus(U_\ell^1\cup U_\ell^2)} x_t \geq \phi, \qquad (2.17g)$$

for $\ell \in \{p+1,\ldots,m\}$, $U_\ell^1 \subseteq \{p+1,\ldots,\ell\}$, $U_\ell^2 \subseteq \{\ell+1,\ldots,n\}$, $h|U_\ell^1\cup U_\ell^2| \leq 1$, and $h|U_\ell^2| \leq 1-\phi$;

$$h\sum_{t\in U} z_t + \sum_{t\in\{1,\ldots,m\}\setminus U} x_t \geq \phi, \qquad (2.17h)$$

for $U \subseteq \{p+1,\ldots,m\}$ with $h|U| \leq 1$.

Proof. (sketch) The main idea of the proof is that by Lemma 8, if $(x,z) \notin K^\phi$, but it satisfies (2.17a), (2.17d), (2.17e), then the minimum capacity of an $s-q$ cut of $G^\phi(x,z)$ is smaller than 1. Then one can show that the set of nodes inducing a minimum capacity $s-q$ cut takes one of the following forms:

- $S = \{s\} \cup \{v_i^1 \;:\; i = p,\ldots,m\} \cup \{v_i^2 \;:\; i = t_1,\ldots,m\} \cup \{w_t \;:\; t \in \{1,\ldots,n\}\setminus(S_1\cup S_2)\}$, where S_1 and S_2 satisfy the conditions of (2.17f).
- $S = \{s\} \cup \{v_i^1 \;:\; i = p,\ldots,m\} \cup \{v_i^2 \;:\; i = \ell,\ldots,m\}\} \cup \{w_t \;:\; t \in \{1,\ldots,n\}\setminus(U_\ell^1\cup U_\ell^2)\}$, where U_ℓ^1 and U_ℓ^2 satisfy the conditions of (2.17g).
- $S = \{s\} \cup \{v_i^1 \;:\; i = p,\ldots,m\} \cup \{w_t \;:\; t \in \{1,\ldots,m\}\setminus U\}$, where U satisfies the conditions of (2.17h). □

Notice that if $\phi = 1$, then inequalities (2.17f) are equivalent to (2.8g) when applied to K_{j*}.

Proposition 3. *Inequalities (2.17f) can be separated in $O(n\log n)$ time, (2.17g) in $O(n^2)$ time, and (2.17h) in $O(n)$ time.*

2.2.3 Implementation and computational evaluation

In order to assess the appropriateness of the MIP formulation and the power of the cutting planes developed in the present paper, we implemented a Branch-and-Cut based solver for our problem along the lines as already described in Section 2.1.8.

The Branch-and-Cut algorithm

In our Branch-and-Cut algorithm, we have assessed the merits of inequalities (2.17f)-(2.17h), which are are valid for $K^\phi = K^\phi_{j*}$, and the inequalities (2.8g) valid for $K = K^\phi_{*k}$.

Preliminary computations had shown that the inequalities (2.17g) and (2.17h) had been violated only very rarely, therefore, in the following tests they were not used.

In our tests we compared two algorithms: B^+ and B^-. The algorithm B^+ separates the following cuts:

(i) for each K^ϕ_{jk}, the inequalities (2.17f) for K^ϕ_{j*}, and the inequalities (2.8g) adapted to $K^\phi_{*k} = K^{*j}$.

(ii) Flow Cover inequalities, as defined by [89].

(iii) Gomory's fractional cuts, see e.g., [83].

The cuts (2.17f) and (2.8g) will be called together (S_1, S_2) cuts.

The algorithm B^- separates only the inequalities (ii) and (iii) above.

Both algorithms were implemented in C++ using the ILOG MIP Library ver. 7.5 through the ILOG Concert Technology interface (these are products of ILOG). Each algorithm started with preprocessing that sometimes eliminated hundreds of rows and columns. The inequalities in class (i) were separated by our procedure, while those in classes (ii) and (iii) were separated automatically by the solver. Moreover, we used the built-in heuristic to find feasible solutions during the search.

Test instances

Being not aware of any benchmark instances to the problem at hand, we have modified the instances of [24]. These instances were devised for the problem with finish-to-start precedence constraints (cf. Section 2.1). The most important parameters are the following:

- number of activities, n,

- number of resources, r,
- average slack, s, which is computed as $s = \sum_{j=1}^{n}(d_j - r_j + 1 - p_j)/n$, where p_j is the minimum time to complete activity j.

The processing times and the precedence relations were generated at random. De Boer obtained a total of 450 instances by generating 10 random instances for each combinations of the above parameters, where $n \in \{10, 20, 50\}$, $r \in \{3, 10, 20\}$ and $s \in \{2, 5, 10, 15, 20\}$. We modified each instance by setting $\phi_{jk} = 0.5$ for each pair of activities (j, k) connected by a precedence constraint. That is, we replaced each finish-to-start precedence constraint by a feeding precedence constraint with an overlap of 0.5.

Results

We run each of the algorithms B^+ and B^- on each instance for 420 seconds of CPU time on a PC with a 2.4GHz Pentium 4 processor and Windows 2000 operating system. When the time limit was hit, the search terminated and the algorithms returned the best upper and lower bounds found so far. Let ub^+ and lb^+ denote the best upper and lower bounds, respectively, found by algorithm B^+. Similarly, let ub^- and lb^- denote the best upper and lower bounds determined by B^-. The numerical tables of this section can be found in Section 5.2.

Table 5.7 summarizes the average ub/lb values for both algorithms. Averages are taken over the ten instances in each class defined by the parameters (n, r, s). For each class there are two numbers, the first one gives the performance for algorithm B^+, whereas the second one gives that of algorithm B^-. A value of 1 indicates that all instances in the class were solved to optimality within the given time limit. As can be seen, algorithm B^+ performs slightly better than algorithm B^-, except in the class with $(n = 50, r = 20, s = 20)$. In fact, in this class B^+ did not obtain a feasible solution for 3 out of the 10 instances within the 420 CPU seconds time limit.

Table 5.8 compares the upper and the lower bounds obtained by the two algorithms. Notice that in each class (n, r, s), the first row contains the average ratio ub^+/ub^-, while the second row comprises the average lb^+/lb^- value. When $ub^+/ub^- < 1$, the algorithm B^+ found a better feasible solution than B^- on average. In contrast, when $lb^+/lb^- < 1$, then algorithm B^+ proved a weaker lower bound than algorithm B^- on average. Again, in most cases the two algorithms found the same lower and upper bounds, and in a few classes B^+ outperformed B^-, while the converse essentially happened only in the class $(n = 50, r = 20, s = 20)$.

Finally, Tables 5.9 and 5.10 summarize the most important aspects of the computations with algorithm B^+ and B^-, respectively. For B^+ we provide the average CPU time (in seconds), the average number of search-tree nodes, and the average number of Flow Cover, Gomory's fractional and the (S_1, S_2) cuts generated over the ten instances in each class (n, r, s). For the algorithm B^-, only average CPU times and average number of search-tree nodes are provided.

Observe that adding our problem specific cutting planes is not always advantageous. The difference between the performance of the two algorithms is minor. In contrast, for the special case with finish-to-start precedence constraints discussed in Section 2.1, there was a clear cut between the results obtained with and without problem specific cutting planes: it was evident that by adding problem specific cuts we got better results. A partial explanation can be that in case of feeding precedence constraints, the smaller ϕ, the larger the overlap allowed between the pairs of activities j, k with $(j, k, \phi) \in \mathcal{E}$, and the problem is less constrained.

Note however that we also run the two algorithms B^+ and B^- of this section on the instances with finish-to-start precedence constraints and found that the results were inferior to those obtained in Section 2.1.8 using a different MIP formulation and problem specific cuts. However, we got better results in terms of the ratio of the upper and lower bounds for the problem with feeding precedence constraints than in case of finish-to-start precedence constraints using the best method. This hints that project scheduling with feeding precedence constraints is an easier problem than that with finish-to-start precedence constraints.

2.2.4 Outlook to applications and extensions

The results of this section have been used in applications developed partly by the author. In [80, 67] a planning system is described for supporting project planning in a machining factory, and in [73, 103] a hierarchical planning and scheduling system is sketched, where the long time plans were obtained by the models and methods described above.

Since the publication of these results, feeding precedence constraints, along with our modeling approach, has received a considerable attention by others, see e.g., [2, 6, 82, 92]. In addition, we have also used these ideas for solving a large scale industrial job shop scheduling problem [28], where our task was to devise an automatic scheduling system for the light-source (incandescent lamps) production facility of General Electric in Nagykanizsa, Hungary.

Finally, our polyhedral results have been extended to arbitrary partial orders on the arcs of a flow set polyhedron by Atamtürk and Zhang [3].

2.3 Resource leveling in a machine environment

In this section we consider resource leveling problems in a dedicated parallel machine environment. There are m machines, M_1 through M_m, a finite set $\mathcal{R}$ of renewable resources along with target levels $L_i \in \mathbb{Q}$, $i \in \mathcal{R}$, and a set of n tasks, $\mathcal{J}$, partitioned into m disjoint subsets $\mathcal{J} = \mathcal{J}_1 \cup \mathcal{J}_2 \cdots \cup \mathcal{J}_m$. Those tasks in $\mathcal{J}_i$ have to be processed exclusively on machine $\mathcal{J}_i$. Each task j has a processing time p_j, release time r_j, a deadline d_j (all integral numbers), and resource requirements $a_{ij} \in \mathbb{Q}$, $i \in \mathcal{R}$. Preemption of tasks is not allowed. Let $T := (\max_{j\in\mathcal{J}} d_j)$. No machine can process more than one task at a time, but there is no limitation on the maximum parallel usage of the resources. A *schedule* $\mathcal{S}$ specifies the starting time of every task, i.e, $\mathcal{S} = (S_1, \ldots, S_n)$, where S_j is the starting time of task j. A schedule is *feasible* if $r_j \leq S_j$ and $S_j \leq d_j - p_j$ hold for every task j, and the processing of any pair of tasks j and j' on the same machine is performed in disjoint time periods, i.e., $[S_j, S_j + p_j) \cap [S_{j'}, S_{j'} + p_{j'}) = \emptyset$ for $j, j' \in N_k$ with $j \neq j'$. For each resource $i \in \mathcal{R}$, the *resource profile* of schedule $\mathcal{S}$ is a mapping $A_i^{\mathcal{S}} : [0, T] \to \mathbb{Q}$ with $A_i^{\mathcal{S}}(t) := \sum_{j\in\mathcal{J}:S_j\leq t<S_j+p_j} a_{ij}$. We will consider the following type of objective functions:

$$f(A^{\mathcal{S}}) := \sum_{i\in\mathcal{R}} \int_0^T \hat{f}_i(A_i^{\mathcal{S}}(t), L_i)dt, \tag{2.18}$$

where the functions $\hat{f}_i : \mathbb{Q}_+ \times \mathbb{Q}_+ \to \mathbb{Q}_+$ satisfy $\hat{f}_i(x, y - z) = \hat{f}_i(x + z, y)$. For the functions $\hat{f}_i$ we will consider the functions f_{lin} and f_{quad} defined in (2.2) and (2.3), respectively.

A typical application area is workforce leveling, or smooth energy utilization.

We begin with a brief literature review in Sect. 2.3.1. Then we study the complexity of the one machine resource leveling problem in which the starting times of all tasks on all but one machines are fixed. We will show that this problem is $\mathcal{NP}$-hard, but if the ordering of tasks on the remaining machine is fixed, then the optimal starting times can be computed in polynomial time (Sect. 2.3.2).

2.3.1 Previous work

One of the first heuristics for resource leveling is due to Burgess and Killebrew [15]. This procedure is applicable to CPM/PERT networks consisting

of activities (nodes) and temporal relations between them (arcs). It aims at finding the best starting times of the activities by shifting them to the right step-by-step in several rounds. This method has been extended to the multi-project, multi-resource case by Woodworth and Willie [106]. The above procedures cannot handle resource constraints. To the best of our knowledge, the first constructive heuristic for the resource leveling problem with resource constraints has been proposed by Neumann and Zimmermann [85], where it is shown that the *general resource leveling problem* in which there are precedence constraints between the tasks is $\mathcal{NP}$-hard in the ordinary sense. A local search heuristic and an exact method is evaluated in [86]. The computational results of the exact method are limited to instances with 20 tasks only. In Ballestin et al [5], resource leveling is applied in make-to-order manufacturing (but without resource constraints). For a recent review of resource constrained project scheduling, see [49]. Further information on the topic, including several models and algorithms, can be found in [25].

There is a considerable literature on machine scheduling with additional resources. Blazewicz et al. [9] propose a classification scheme and provide several complexity results for machine scheduling under resource constraints. For a more comprehensive overview and references, see the textbook [8]. Kellerer and Strusevich [62, 63] consider dedicated parallel machines and one or more additional resources under various assumptions. However, in all these works resources occur only in the constraints of the problem, and the objective function is related to the completion times of the jobs. Caramia and Dell'Olmo [17] discuss various resource leveling problems, where tasks have unit length and instead of machines, there is an incompatibility relation between the tasks such that only compatible tasks may be executed in parallel. The authors discuss the computational complexity of several variants and propose a heuristic algorithm for solving those problems. A heuristic algorithm for scheduling workers of different skills is presented in Valls et al. [102], where one of the objectives is to minimize the deviation of resource usage from the average load.

2.3.2 The one machine resource leveling problem

Suppose the starting times of the tasks on all but one machine are fixed, and we want to solve the resource leveling problem by finding the optimal sequence and starting times for the set of tasks J_1 assigned to the remaining unscheduled machine. Let $\min_{j\in\mathcal{J}} r_j = v_1 < v_2 < \cdots < v_K < v_{K+1} = \max_{j\in\mathcal{J}} d_j$ be the set of time points when some job starts or finishes in the fixed part of the schedule augmented with the minimum job release date and the maximum deadline. Clearly, the resource usage for all resources is

constant in each interval $[v_k, v_{k+1})$, and let $F_i^v(k)$ denote the *free capacity* of resource $i \in \mathcal{R}$ in this interval, i.e., $F_i^v(k) := L_i - \sum_{j \in \mathcal{J} \setminus \mathcal{J}_1 . S_j \le v_k < S_j + p_j} a_{ij}$. In the *Single Machine Resource Leveling Problem (SMRLP)*, a feasible schedule $\mathcal{S}_1$ of the tasks in $\mathcal{J}_1$ is sought such that an objective function

$$f(A^{\mathcal{S}_1}) := \sum_{j \in \mathcal{J}_1} \sum_{k=1}^{K} \Delta(S_j, S_j + p_j, v_k, v_{k+1}) \sum_{i \in \mathcal{R}} \hat{f}_i(a_{ij}, F_i^v(k)) \tag{2.19}$$

is minimized, where $\Delta(\ell_1, u_1, \ell_2, u_2) := \max\{0, \min\{u_1, u_2\} - \max\{\ell_1, \ell_2\}\}$ is the size of the intersection of two intervals $[\ell_1, u_1)$ and $[\ell_2, u_2)$. Since $\hat{f}_i(a_{ij}, F_i^v(k)) = \hat{f}_i(a_{ij} + \sum_{j' \in \mathcal{J} \setminus \mathcal{J}_1 : S_{j'} \le v_k < S_{j'} + p_{j'}} a_{ij'}, L_i)$ by the assumption on $\hat{f}_i$, (2.19) is equivalent (2.18) for the problem SMRLP (the S_j are fixed for $j \in \mathcal{J} \setminus \mathcal{J}_1$). We can define f_{lin} and f_{quad} using $\hat{f}_\ell(x, y) = w_i \max\{0, x - y\}$ and $\hat{f}_i(x, y) = w_\ell (x - y)^2$, $w_i \ge 0$, respectively.

Proposition 4. *[29]*SMRLP is NP-complete in the strong sense both for f_{lin} and f_{quad} even if a feasible schedule exists.*

Proof. Throughout this proof we assume that there is only one resource, and thus the subscript of the resource is omitted. In order to prove that SMRLP belongs to NP, a non-deterministic algorithm need only guess a schedule $\mathcal{S}$, and check in polynomial time whether $\mathcal{S}$ is feasible and has value at most U. The feasibility of $\mathcal{S}$ for an instance I can certainly be verified in polynomial time. We give an algorithm to compute the value of f_{lin} or f_{quad} on $\mathcal{S}$. That is, for each task j, determine those intervals $[v_k, v_{k+1})$ with non-empty intersection with $[S_j, S_j + p_j)$ in $O(K)$ time. Then for each of these intervals, determine the value $\max\{0, a_j - F^v(k)\}$ or $(a_j - F^v(k))^2$ and multiply it with the size of the intersection, which is $\min\{S_j + p_j, v_{k+1}\} - \max\{S_j, v_k\}$. Adding up these values gives the objective function value on $\mathcal{S}$.

To show that SMRLP is $\mathcal{NP}$-hard, we give a polynomial time Turing reduction from 3-PARTITION to SMRLP.

3-PARTITION

Instance: A set E consisting of $3q$ elements, a size $s(e) \in \mathbb{Z}^+$ for each element in E, and a bound $B \in \mathbb{Z}^+$ such that $B/4 < s(e) < B/2$ for each $e \in E$.

Question: Can E be partitioned into q disjoint sets $E_1, E_2, \ldots, E_q$ such that each E_i contains three elements of E, and such that, for $1 \le i \le q$, $\sum_{e \in E_i} s(e) = B$?

For an instance of 3-PARTITION, the corresponding instance of SMRLP consists of $4q - 1$ tasks, one machine and one resource. There is a one-to-one correspondence between the first $3q$ tasks and the elements of E. For $e \in E$,

$p_e = s(e)$, $r_e = 0$, $d_e = q(B+1)-1$, $a_e = 1$. All of the remaining $q-1$ tasks have processing time 1, resource requirement 2, earliest start time 0 and deadline $q(B+1)-1$. The free capacity of the resource is 1, except in the time periods $[\ell(B+1)-1, \ell(B+1))$, for $1 \leq \ell < q$, in which it is 2. This can be encoded by $2q-1$ intervals, i.e., $v_{2\ell-1} = \ell(B+1)-1$, $v_{2\ell} = \ell(B+1)$, $F^v(2\ell-1) = 1$, and $F^v(2\ell) = 2$ for $1 \leq \ell < q$, and $F^v(2q-1) = 1$. Since the total processing time of all tasks is $q(B+1)-1$, any sequence of tasks is feasible. Finally, the answer is 'YES' to the instance of 3-PARTITION if and only if the corresponding instance of SMRLP has a feasible solution of value 0. □

A special case of SMRLP is when a total ordering of the tasks on the unscheduled machine is given and the optimum starting times respecting the fixed ordering is sought. That is, without loss of generality, the tasks have to be scheduled in the order 1 through n, where $n = |\mathcal{J}_1|$. Under this assumption, we will prove that an optimal solution can be found in polynomial time. We may assume that $r_j + p_j \leq r_{j+1}$ and that $d_j \leq d_{j+1} - p_{j+1}$ for $1 \leq j \leq n-1$. The cost of starting task j in time point t can be specified by a closed-form expression as $\hat{c}_j(t) = \sum_{k=1}^{K} \Delta(t, t+p_j, v_k, v_{k+1}) \sum_{i \in \mathcal{R}} \hat{f}_i(a_{ij}, F_i^v(k))$. This function is defined for $t \in [r_j, d_j - p_j]$, it is piecewise linear and continuous with at most $2K$ breakpoints, since $\hat{c}_j$ may have a breakpoint when task j starts or completes at some time point v_k.

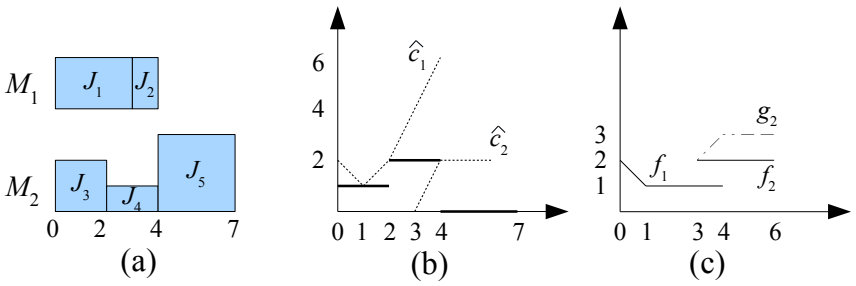

Figure 2.5: Illustration for the example SMRLP problem.

Example. There are 2 machines, a single resource of capacity 3, and 5 jobs, J_1 through J_5, with processing times $p_1 = p_5 = 3$, $p_2 = 1$, $p_3 = p_4 = 2$, and resource requirements $a_1 = a_2 = a_3 = 2$, $a_4 = 1$, $a_5 = 3$. Moreover, $r_j = 0$ and $d_j = 7$ for every job j and $w_1 = 1$. Jobs J_1 and J_2 are assigned to the first machine, while jobs J_3, J_4 and J_5 to M_2. Suppose the schedule of M_2 is fixed as in Fig. 2.5 (a). The height of the rectangles indicate the requirements from the single resource. However, the schedule of M_1 is not fixed, but J_1 and J_2 have to be processed in this order. Fig. 2.5 (b) depicts

the free capacity of the resource over time with fat horizontal line segments, as well as the functions $\hat{c}_1$ and $\hat{c}_2$ for the f_{lin} objective function.

Let $f_j(t)$ be the minimum cost when the problem is restricted to the first j tasks and task j starts not later than time t. We define $f_j(t)$ recursively by using an auxiliary function, $g_j(t)$. That is,

$$\begin{aligned} f_1(t) &= \min_{r_1 \le t' \le \min\{t, d_1 - p_1\}} \hat{c}_1(t'), && \text{for } t \ge r_1 \\ g_j(t) &= f_{j-1}(t - p_{j-1}) + \hat{c}_j(t), && \text{for } j = 2, \ldots, n,\ t \ge r_j, \\ f_j(t) &= \min_{r_j \le t' \le \min\{t, d_j - p_j\}} g_j(t'), && \text{for } j = 2, \ldots, n,\ t \ge r_j. \end{aligned}$$

Clearly, f_j is decreasing, i.e., $f_j(t_1) \ge f_j(t_2)$ for $t_1 \le t_2$ by definition. The overall optimum objective function value is given by $f_n(d_n - p_n)$. We have to prove that the optimum solution can be computed in polynomial time in the input length. It suffices to show that every function f_j is piecewise linear with a polynomial number of line segments in the input length, see Fig. 2.5 (c).

Proposition 5. *[29]* The functions f_j and g_j are piecewise linear and continuous.*

Proof. Apply induction on j and use the fact that the sum of piecewise linear continuous functions is piecewise linear and continuous. □

Proposition 6. *[29]* Let u_j and h_j be the number of line-segments of the graph of $f_j(t)$ and $\hat{c}_j(t)$, respectively. Then $u_j \le u_{j-1} + h_j$.*

Proof. The claim clearly holds for $f_1(t)$. By induction, suppose that $f_{j-1}(t)$ satisfies the claimed property, and verify it for $f_j(t)$. Since the graph of $f_{j-1}(t - p_j)$ and $\hat{c}_j(t)$ consists of u_{j-1} and h_j line-segments, respectively, $g_j(t)$ has at most $u_{j-1} + h_j$ line-segments. Let $s_1 < s_2 < \cdots, < s_{K'}$ be the breakpoints between the line-segments of $g_j(t)$. Starting with s_1 and s_2, we determine $f_j(t)$ between the consecutive pairs of breakpoints. Clearly, $f_j(r_j) = g_j(r_j)$ and suppose $f_j(t)$ is determined until s_k. We distinguish between two cases:

- $f_j(s_k) \le g_j(s_{k+1})$. Since $g_j(t)$ is linear and continuous on $[s_k, s_{k+1}]$ and $f_j(s_k) \le g_j(s_k)$ by definition, we have $f_j(t) = f_j(s_k)$ for $t \in (s_k, s_{k+1}]$.
- $f_j(s_k) > g_j(s_{k+1})$. We know that $f_j(s_k) \le g_j(s_k)$ by definition. If $g_j(s_k) = f_j(s_k)$, then on the interval $[s_k, s_{k+1}]$, $f_j(t) = g_j(t)$. Otherwise, we must have $g_j(s_k) > f_j(s_k)$. Then, since both functions $f_j(t)$ and $g_j(t)$ are piecewise linear and continuous, there exists a unique time

point $t_1 \in (s_k, s_{k+1})$ such that $g_j(t_1) = f_j(t_1)$. Then $f_j(t) = f_j(s_k)$ for $t \in (s_k, t_1]$, and $f_j(t) = g_j(t)$ for $t \in (t_1, s_{k+1}]$. Hence, t_1 is a new breakpoint, but s_k will not be a breakpoint of f_j, since $g_j(s_k) > f_j(s_k)$ and both functions are continuous.

In all cases, the number of breakpoints of the graph of f_j until s_{k+1} is not more than that of g_j, which is at most $u_{j-1} + h_j$. □

Theorem 4. *[29]* $f_n(d_n - p_n)$ *can be computed in* $O(nK)$ *time.*

Proof. We proceed by induction. First notice that in order to determine $\hat{c}_j$ for any $j \in \mathcal{J}_1$, it suffices to evaluate it in its breakpoints, which can be accomplished in $O(K)$ time. Therefore, f_1 can be computed in $O(K)$ time. Now consider f_j. Since both g_j and f_j has at most $\sum_{i=1}^{j} h_i$ breakpoints, and $h_i \leq 2K$, f_j can be computed in $O(jK)$ time, and the statement follows. □

In order to recover the optimum solution, we have to keep all of the f_j, represented by (breakpoint, value) pairs. Then working backward, we can find the starting times of the tasks in the optimal solution.

The single machine resource leveling problem is repeatedly solved in a branch-and-bound procedure which was designed for the m-machine problem with jobs dedicated to machines. It has several other ingredients, like fast heuristics for finding a good ordering of jobs on the machines, special branching rules, etc. The method is very efficient in practice both for the linear and for the quadratic objective function, since it finds better solutions than CPLEX solver (ver 11.2) in a limited computation time, for details see Drótos and Kis [29].

Chapter 3

Machine scheduling with non-renewable resources

In this chapter we will study machine scheduling problems amended with non-renewable resource constraints. We will consider problems, where jobs *consume* non-renewable resources. On the other hand, we will also tackle problems, where the jobs *produce* some products, that we also call resources. One reason for our terminology is that we will show that the two types of problems are equivalent for certain objective functions.

Our primary focus is on computational complexity and approximability. That is, we will identify the borderline between tractable and hard problems, and we will describe approximation algorithms and also inapproximability results.

In our terminology, we will follow Garey and Johnson [36]. Recall that an *optimization problem* Π is given by a tripe (D, S, c), where D is the set of *problem instances*, for each $I \in D$, $S(I)$ is the set of *feasible* solutions (it may be empty), and for each $\sigma \in S(I)$, $c(I, \sigma) \in \mathbb{R}$ is the *value* of solution σ (see [36], Section 6.1). If $S(I)$ is not empty, then let $\sigma^*(I)$ denote an optimal solution, i.e., in case of minimization problems, $c(I, \sigma^*) \leq c(I, \sigma)$ for each $\sigma \in S(I)$, and in case of maximization problems, $c(I, \sigma^*) \geq c(I, \sigma)$ for each $\sigma \in S(I)$. The value of the optimal solution will be denoted by $OPT(I) := c(I, \sigma^*(I))$. An *approximation algorithm* A for an optimization problem Π determines a feasible solution for each instance I of Π with nonempty $S(I)$. We require that A be of polynomial time complexity. For each instance I of Π, let $A(I)$ denote the value of the solution returned by A. The *relative error* of A in case of minimization [maximization] problems is

$$R_A(I) := \frac{A(I)}{OPT(I)} \qquad \left[R_A(I) := \frac{OPT(I)}{A(I)}\right].$$

Notice that by this definition, $R_A(I) \geq 1$ both for minimization and maximization problems. The *approximation ratio* of A is

$$R_A := \min\{r \geq 1 \mid R_A(I) \leq r, \forall I \in D\}.$$

The *approximability* of an optimization problem Π can be measured by the smallest r for which there exists an approximation algorithm A with $R_A \leq r$. Notice that r may not be a constant, and may depend on some parameters of the problem instances. Approximation schemes play a central role in the theory of approximation algorithms. A set of algorithms $\{A_\varepsilon\}_{\varepsilon>0}$ constitutes a *polynomial time approximation scheme* (PTAS) for an optimization problem Π if $R_{A_\varepsilon} \leq 1+\varepsilon$ for each $\varepsilon > 0$. Notice that each A_ε must be of polynomial time complexity on the instances of Π. If, in addition, each A_ε of the scheme has polynomial time complexity in $1/\varepsilon$ as well, then we say that $\{A_\varepsilon\}_{\varepsilon>0}$ is a *fully polynomial time approximation scheme* (FPTAS). A very useful tool for proving the existence/non-existence of approximation algorithms with given properties for an optimization problem is the application of approximation preserving reduction. The main definitions are summarized in Section 5.4 of the Appendix.

In Section 3.1 we will study the relationships between three classes of optimization problems: (i) single-machine scheduling problems with resource consuming jobs, (ii) single-machine scheduling problems with resource producing jobs, and (iii) knapsack problems. We will provide approximation preserving reductions between these problem classes. With the help of these reductions, we obtain approximation algorithms and inapproximability results for various special cases of the scheduling problems.

In Section 3.2 we describe positive and negative results on the approximability of parallel machine scheduling problems with the makespan and the maximum lateness objective, respectively.

The results of this section are based on the papers Drótos and Kis [30], Györgyi and Kis [42], Györgyi and Kis [43], Györgyi and Kis [44], Kis [66], Györgyi and Kis [45].

3.1 Single-machine problems

In this section we consider single-machine problems involving some non-renewable resources. The primary focus is on approximation preserving reductions between scheduling problems and variants of the knapsack problem. Firstly, we define the scheduling problems as well as the variants of the knapsack problems that we will use later.

3.1.1 Resource Scheduling Problems

In this section we define the two resource scheduling problems, the *Delivery tardiness problem* (see [30]) and the *Material consumption problem* (see [19]).

In the *Delivery tardiness problem* (DTP_q^r) we have a single machine, a finite set of n jobs, and a set of r materials produced by the jobs. The machine can perform only one job at a time, and preemption is not allowed. Job J_j, $j \in \{1, \ldots, n\}$, has a processing time $p_j \in \mathbb{Z}_+$, and produces some materials, which is described by an r-dimensional non-negative vector $a_j \in \mathbb{Z}_+^r$. There are due dates along with required shipments, i.e., pairs (u_ℓ, b_ℓ) with $u_\ell \in \mathbb{Z}_+$, and $b_\ell \in \mathbb{Z}_+^r$, $\ell = 1, \ldots, q$, and $0 \leq u_1 < \cdots < u_q$. The solution of the problem is a sequence σ of the jobs. The starting time of the i^{th} job is then $S_{\sigma(i)} = \sum_{k=1}^{i-1} p_{\sigma(k)}$. A shipment (u_ℓ, b_ℓ) is *met* by S, if the total production of those jobs finishing by u_ℓ is at least $\tilde{b}_\ell := \sum_{k=1}^{\ell} b_k$, i.e., $\sum_{(j\,:\,S_j+p_j \leq u_\ell)} a_j \geq \tilde{b}_\ell$ (coordinate wise), otherwise it is *tardy*. Let $C_\ell(S)$ be the earliest time point $t \geq 0$ with $\sum_{(j\,:\,S_j+p_j \leq t)} a_j \geq \tilde{b}_\ell$. The *tardiness* of a shipment is $T_\ell(S) := \max\{0, C_\ell(S) - u_\ell\}$. The *maximum tardiness of a schedule* is $T_{\max}(S) := \max_\ell T_\ell(S)$. The objective is to minimize the maximum tardiness. We denote this problem by $1|dm = r|T_{\max}$, where '$dm = r$' indicates that the number of products is fixed to r (not part of the input). An important special case of this problem is when there are only two time points ($0 \leq u_1 < u_2$) when some product is due (denoted by $1|dm = r, q = 2|T_{\max}$). Since $T_{\max}$ can be 0 in an optimal solution, we will consider the *shifted delivery tardiness objective function* defined as $T_{\max}^s := T_{\max} + const$, where *const* is a positive constant, depending on the problem data. This problem was first introduced in Drótos and Kis [30].

In the *Material consumption problem* (MCP_q^r) there is a single machine, a finite set of n jobs, and a set of r materials consumed by the jobs. The machine can perform only one job at a time, and preemption is not allowed. There are n jobs J_j, $j = 1, \ldots, n$, each characterized by two numbers: processing time p_j and quantities consumed from the resources $a_j \in \mathbb{Z}_+^r$. The resources have initial stocks, and they are replenished at given moments in time, i.e., there are q pairs $(u_1, \tilde{b}_1), \ldots, (u_q, \tilde{b}_q)$, with $0 = u_1 < \cdots < u_q$ being the time points and the $\tilde{b}_\ell \in \mathbb{Z}_+^r$ the quantities supplied. A *schedule* S specifies a starting time for each job such that the jobs do not overlap in time, and the total material supply up to the starting time of every job is at least the total request of those jobs starting not later than S_j, i.e., $\sum_{(\ell\,:\,u_\ell \leq S_j)} \tilde{b}_\ell \geq \sum_{(j'\,:\,S_{j'} \leq S_j)} a_{j'}$ (coordinate wise). The objective is to minimize the *makespan* defined as the maximum job completion time. We denote this problem by $1|nr = r|C_{\max}$, where '$nr = r$' indicates that the number of the raw materials is fixed to r (not part of the input). An important spe-

cial case of this problem is when there are only two time points ($u_1 = 0$ and $u_2 > 0$) when some resource is supplied ($1|nr = r, q = 2|C_{\max}$). This problem was first introduced by Carlier [19]. For shorter notation, let $b_\ell := \sum_{k=1}^{\ell} \tilde{b}_k$.

Assumption 1. *In both problems $\sum_\ell \tilde{b}_\ell = \sum_j a_j$ holds without loss of generality.*

3.1.2 Knapsack Problems

In the (basic) *Knapsack Problem* (KP) there is a set of n items j with profit v_j and weight w_j. One has to select a subset of the items with the largest total profit so that the total weight of the selected items is at most a given constant ('capacity') b'. Formally:

$$OPT_{KP} := \max \sum_{j=1}^{n} v_j x_j \tag{3.1}$$

$$\sum_{j=1}^{n} w_j x_j \leq b' \tag{3.2}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{3.3}$$

We assume that $w_j \leq b'$ for each item j. We will use the notation OPT_{KP} for the optimal value of this problem.

In the *r-dimensional Knapsack Problem (r-DKP)* each item has r weights and there are r constraints:

$$OPT_{r-DKP} := \max \sum_{j=1}^{n} v_j x_j \tag{3.4}$$

$$\sum_{j=1}^{n} w_{ij} x_j \leq b'_i, \quad i = 1, \ldots, r \tag{3.5}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{3.6}$$

The optimum value of this problem is denoted by OPT_{r-DKP}.

3.1.3 Previous work

Scheduling problems with producer jobs only is also known as *scheduling of inventory releasing jobs*, and this model has been recently proposed by Boysen et al. [10]. They studied the problem of minimizing inventory levels while satisfying all the external demands on time (there, the delivery requests

have strict deadlines). They proved the $\mathcal{NP}$-hardness of the problem and proposed polynomial algorithms for several variants.

Scheduling of jobs consuming some non-renewable resources (like raw materials, money, energy, etc.) is an old problem class: the original model was described by Carlier [19] and by Carlier and Rinnooy Kan [18] in the early 80's. Since then several authors studied scheduling problems with jobs consuming non-renewable resources (e.g. [96], [101], [84], [41], [11], [35], [12]. In particular, Carlier and Rinnooy Kan [18] defined the problem with precedence constraints, but without machines, and derived polynomial algorithms for various special cases. Carlier [19] showed algorithmic and complexity results. Slowinski [96] studied problems with preemptive jobs on parallel unrelated machines with renewable and non-renewable resources. Toker et al. [101] proved that the problem $1|nr = 1|C_{\max}$ reduces to the 2-machine flow shop problem provided that the resource has a unit supply at each time period. Grigoriev et al. [41] studied problems with one machine and presented some basic complexity results and simple approximation algorithms. Gafarov et al. [35] complemented the findings of Grigoriev et al. by additional complexity results. Neumann and Schwindt [84] studied general project scheduling problems with inventory constraints in a more general setting, where jobs (activities) may consume as well as produce non-renewable resources. In case of a single machine, the problem was proved $\mathcal{NP}$-hard in the strong sense by Kellerer et al. [59], and for minimizing the maximum stock level, the authors proposed three different approximation algorithms with relative error 2, 8/5, 3/2, respectively. Briskorn et al. [11] provided complexity results for several variants, while Briskorn et al. [12] described an exact algorithm for minimizing the weighted sum of the job completion times on a single machine.

Knapsack problems are among the most-studied problems in combinatorial optimization. There are many variants and methods of all kinds have been devised over the years to get some solutions, see e.g. the book of Kellerer et al. [61] for an excellent overview. These problems have played an important role in the design of algorithms for scheduling problems, see e.g., [76], [72], [98], [32] to mention but a few examples.

3.1.4 Summary of main results

The reductions between the different problems proved subsequently are summarized in Figure 3.1 and Table 3.1. In the figure, a directed arc from problem Π_1 to problem Π_2 labeled by some reduction indicates that Π_1 is reducible to Π_2 by that kind of reduction. In the table we summarize the implications in terms of algorithms of the reductions among the problems.

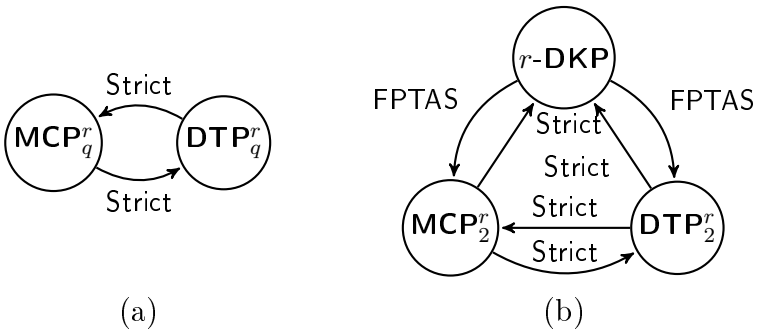

Figure 3.1: Summary of approximation preserving reductions between scheduling and knapsack problems.

The most important results are: (i) There is a Strict reduction from the problem of minimizing the makespan with consumer jobs, and the scheduling problem with producer jobs and the shifted delivery tardiness objective, and vice versa. This finding allows us to convert approximation algorithms for one type of scheduling problems to the other (part (a) of the figure). (ii) If there are only two supply periods, and a single raw material, then scheduling of consumer jobs to minimize the makespan admits a Strict reduction to the basic knapsack problem (part (b) of the figure), which yields a PTAS as well as an FPTAS for the former problem, i.e., we can use any approximation algorithm devised for the knapsack problem to solve the scheduling problem. (iii) There is no FPTAS for the scheduling problem with consumer jobs and at least two non-renewable resources unless $\mathcal{P} = \mathcal{NP}$, because there is an FPTAS reduction from the multi-dimensional knapsack problem to the scheduling problem (part (b) of the figure), and the multi-dimensional knapsack problem does not admit an FPTAS unless $\mathcal{P} = \mathcal{NP}$ if the number of dimensions is at least two. (iv) The problems MCP_{const}^{const} and DTP_{const}^{const} both admit a PTAS. (v) The problem $1|nr = 1, q = 2|\sum w_j C_j$ admits an FPTAS, but if q is part of the input, then no FPTAS exists, unless $\mathcal{P} = \mathcal{NP}$.

3.1.5 Strict reductions between MCP_q^r and DTP_q^r

In this section we prove that there is a Strict-reduction between DTP_q^r and MCP_q^r in both directions. To illustrate the main idea, we present an example in Figure 3.2. In the top, there is a schedule for an instance of the DTP_4^1 problem, and in the bottom, a schedule for the MCP_4^1 problem. The rectangles are the jobs, where the horizontal width indicates the processing time, and the vertical height the amount of resource produced (DTP problem), or the material required (MCP problem). The two schedules consist

Problem		Result	Source
MCP_2^1	$1\|nr = 1, q = 2\|C_{\max}$	FPTAS	Section 3.1.6
MCP_2^{const}	$1\|nr = const, q = 2\|C_{\max}$	no FPTAS[a]	Section 3.1.7
MCP_2^*	$1\|nr, q = 2\|C_{\max}$	no PTAS	Section 3.1.8
MCP_{const}^{const}	$1\|nr = const, q = const, r_j\|C_{\max}$	PTAS[b]	Section 3.1.8
	$1\|nr = 1\|\sum w_jC_j$	no FPTAS	Section 3.1.9
	$1\|nr = 1, q = 2\|\sum w_jC_j$	FPTAS	Section 3.1.9
DTP_2^1	$1\|dm = 1, q = 2\|T_{\max}$	FPTAS	Section 3.1.6
DTP_2^{const}	$1\|dm = const, q = 2\|T_{\max}$	no FPTAS[c]	Section 3.1.7
DTP_2^*	$1\|dm, q = 2\|T_{\max}$	no PTAS	Section 3.1.8
DTP_{const}^{const}	$1\|dm = const, q = const\|T_{\max}$	PTAS	Section 3.1.8

[a] if nr is a constant of at least 2
[b] Jobs may have release dates
[c] If dm is a constant of at least 2

Table 3.1: Approximation schemes and unapproximability results for single-machine problems.

of the same jobs, and the sequence in the bottom is just the reverse of that in the top. The delay in the top indicates the late delivery by job J_{j^*} with respect to due date u_3, whereas in the bottom, the same delay occurs before job J_{j^*} due to waiting for resource supply.

Lemma 9. *[43]* Given an instance $I_D = \{n, q, (p_j, a_j)_{j=1}^n, (u_\ell, \tilde{b}_\ell)_{\ell=1}^q\}$ of the Delivery tardiness problem. Define an instance $I_M = \{n, q, (p_j, a_j)_{j=1}^n, (u'_\ell, b'_\ell)_{\ell=1}^q\}$ of the Material consumption problem:*

$$\begin{aligned} u'_\ell &= u_q - u_{q+1-\ell} \\ b'_\ell &= \tilde{b}_{q+1-\ell} \end{aligned} \qquad \ell = 1, \ldots, q.$$

Then, if σ is a sequence of jobs giving a maximum delivery tardiness of $T^\sigma_{\max}$ for I_D, then scheduling the jobs in reverse σ order gives a schedule of makespan $u_q + T^\sigma_{\max}$ for instance I_M.

Proof. Without loss of generality, $\sigma = (J_1, \ldots, J_n)$, and then the reverse order of jobs is $\sigma^{-1} = (J_n, J_{n-1}, \ldots, J_1)$. For the problem instance I', let $S(\sigma^{-1})$ be the schedule obtained by scheduling the jobs in the order of σ^{-1}, and scheduling each job to start as early as possible while respecting the resource constraints. By contradiction, suppose the makespan $C^{S(\sigma^{-1})}_{\max}$ of schedule $S(\sigma^{-1})$ is larger than $u_q + T^\sigma_{\max}$ (notice that u_q is the last due-date of problem instance I of the Delivery tardiness problem). Then by the

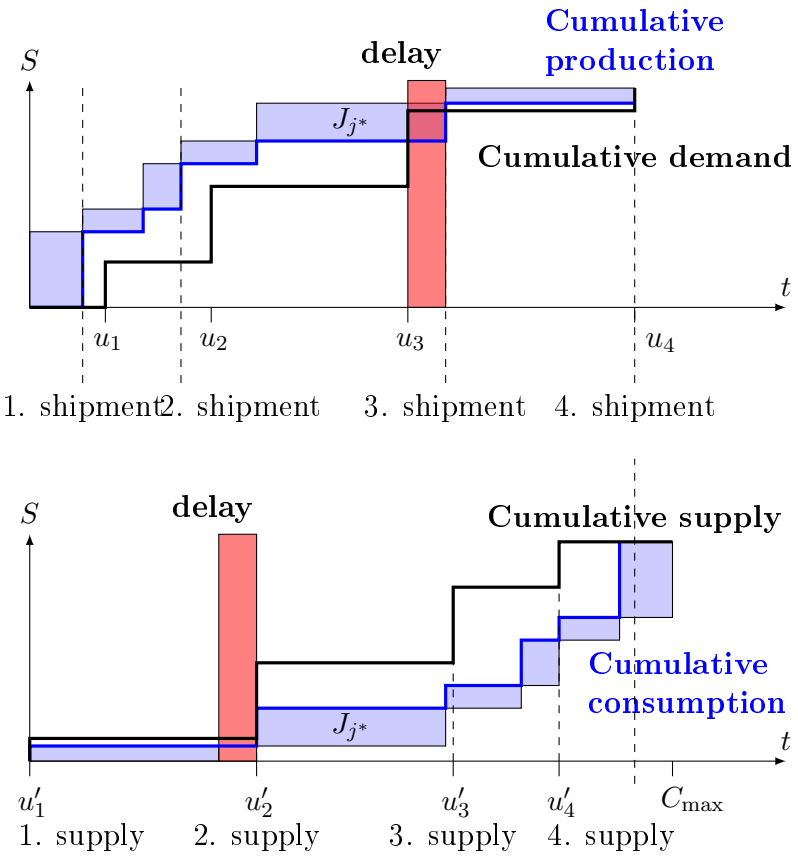

Figure 3.2: Corresponding schedules for the DTP (top) and MCP (bottom) problems.

definition of the makespan, there exist a resource supply date u'_{ℓ^*} and a job index j^* such that

$$C_{\max}^{S(\sigma^{-1})} = u'_{\ell^*} + \sum_{j=1}^{j^*} p_j \tag{3.7}$$

Take the earliest such ℓ^* and the corresponding index j^*. Since job J_{j^*} is scheduled at the earliest possible time, we also have

$$\sum_{j=j^*+1}^{n} a_j \leq \sum_{\ell=1}^{\ell^*-1} b'_\ell \tag{3.8}$$

$$\sum_{j=j^*}^{n} a_j > \sum_{\ell=1}^{\ell^*-1} b'_\ell \tag{3.9}$$

Notice that if $\ell^* = 1$, then since $u'_1 = 0$ by definition, it follows that $j^* = n$ (the makespan is the sum of processing times of all the jobs, since no job may start before time 0), and the right-hand-sides in (3.8), and (3.9) are 0. Since $\sum_\ell \tilde{b}_\ell = \sum_j a_j$, (3.8) and (3.9) are equivalent to

$$\sum_{j=1}^{j^*} a_j \geq \sum_{\ell=\ell^*}^{q} b'_\ell = \sum_{\ell=\ell^*}^{q} \tilde{b}_{q+1-\ell} = \sum_{\ell=1}^{q-\ell^*+1} \tilde{b}_\ell \tag{3.10}$$

$$\sum_{j=1}^{j^*-1} a_j < \sum_{\ell=\ell^*}^{q} b'_\ell = \sum_{\ell=1}^{q-\ell^*+1} \tilde{b}_\ell \tag{3.11}$$

This means that in the instance I of the Delivery tadiness problem, the first $j^* - 1$ jobs are not enough to satisfy the demand of the first $q - \ell^* + 1$ time periods. Since $u'_{\ell^*} = u_q - u_{q-\ell^*+1}$, we have

$$u_q + T^\sigma_{\max} < C^{S(\sigma^{-1})}_{\max} = u'_{\ell^*} + \sum_{j=1}^{j^*} p_j = u_q - u_{q-\ell^*+1} + \sum_{j=1}^{j^*} p_j,$$

where the first inequality follows from our indirect assumption, and the second and third equations from the definition. However, this implies

$$T^\sigma_{\max} < \sum_{j=1}^{j^*} p_j - u_{q-\ell^*+1}.$$

Therefore, schedule σ for instance I_D of the Delivery tardiness problem cannot have maximum tardiness $T^\sigma_{\max}$, a contradiction. □

Lemma 10. *[43]*Given an instance $I_M = \{n, q, (p_j, a_j)_{j=1}^n, (u_\ell, \tilde{b}_\ell)_{\ell=1}^q\}$ of the Material consumption problem. Define an instance $I_D = \{n, q, (p_j, a_j)_{j=1}^n, (u'_\ell, b'_\ell)_{\ell=1}^q\}$ of the Delivery tardiness problem:*

$$\begin{aligned} u'_\ell &= u_q - u_{q+1-\ell} \\ b'_\ell &= b_{q+1-\ell} \end{aligned} \qquad \ell = 1, \ldots, q.$$

Then, if S is a schedule with a makespan of $C^S_{\max}$ for I_M, then scheduling the jobs in reverse order (without any delays among them) gives a schedule of maximum tardiness at most $C^S_{\max} - u_q$ for instance I_D.

Proof. Suppose S completes the jobs in the order $\sigma = (J_1, \ldots, J_n)$. The reverse order is $\sigma^{-1} = (J_n, \ldots, J_1)$. Let $S(\sigma^{-1})$ be the schedule corresponding to the reverse order σ^{-1}, i.e., $S_j(\sigma^{-1}) := \sum_{j'=j+1}^n p_{j'}$. By contradiction,

suppose $T_{\max}(S(\sigma^{-1})) > C^S_{\max} - u_q$. By the definition of $T_{\max}(S(\sigma^{-1}))$, there exist $\ell^* \in \{1, \ldots, q\}$, and some job j^* such that

$$T_{\max}(S(\sigma^{-1})) = \sum_{j=j^*}^{n} p_j - u'_{\ell^*}.$$

Moreover,

$$\sum_{j=j^*}^{n} a_j \geq \sum_{\ell=1}^{\ell^*} b'_\ell \tag{3.12}$$

$$\sum_{j=j^*+1}^{n} a_j < \sum_{\ell=1}^{\ell^*} b'_\ell \tag{3.13}$$

Observe that

$$C^S_{\max} - u_q < T_{\max}(S(\sigma^{-1})) = \sum_{j=j^*}^{n} p_j - u'_{\ell^*} = \sum_{j=j^*}^{n} p_j - (u_q - u_{q+1-\ell^*}),$$

which implies

$$C^S_{\max} < u_{q+1-\ell^*} + \sum_{j=j^*}^{n} p_j. \tag{3.14}$$

In addition (3.12) and (3.13) and the assumption $\sum_\ell \tilde{b}_\ell = \sum_j a_j$ imply

$$\sum_{j=1}^{j^*-1} a_j \leq \sum_{\ell=\ell^*+1}^{q} b'_\ell = \sum_{\ell=\ell^*+1}^{q} \tilde{b}_{q-\ell+1} = \sum_{\ell=1}^{q-\ell^*} \tilde{b}_\ell \tag{3.15}$$

$$\sum_{j=1}^{j^*} a_j > \sum_{\ell=\ell^*+1}^{q} b'_\ell = \sum_{\ell=1}^{q-\ell^*} \tilde{b}_\ell \tag{3.16}$$

However, (3.15) and (3.16) mean that the first j^* jobs in instance I of the Material consumption problem require more resource than that supplied in the first $q - \ell^*$ supply periods. Therefore, the makespan of the schedule is at least $u_{q-\ell^*+1} + \sum_{j=j^*}^{n} p_j$, which is more than the makespan of schedule S by (3.14), a contradiction. □

Corollary 3. *[43]*Let (I_D, I_M) be corresponding instances of the Delivery tardiness and the Material consumption problems. Then the optimum value $T^*_{\max}(I_D)$ of the Delivery tardiness problem equals $C^*_{\max}(I_M) - u_q$, the optimum value of the Material consumption problem minus u_q, where u_q is the last material shipment date in I_M.*

Now we turn to reductions. Since $T_{\max}$ may be 0 in an optimal solution to DTP_q^r, we shift the objective function by a positive constant depending on the problem data: $T^s_{\max} := \max_\ell T_\ell + u_q - u_1$, where u_1 and u_q are the first, and the the last due-date in the DTP_q^r problem instance, respectively. Now we prove the following:

Theorem 5. *[43]* There is a Strict-reduction from the Material consumption problem to the Delivery tardiness problem, and vice versa, there is a Strict-reduction from the Delivery tardiness problem to the Material consumption problem.*

Proof. Firstly, we show that there is a Strict-reduction from MCP_q^r to DTP_q^r. We use the transformation of Lemma 10 to construct function f which maps instances of MCP_q^r to that of DTP_q^r. Clearly, the transformation can be computed in linear time in the size of any instance I_M of MCP_q^r. Let I_M be any instance of MCP_q^r, and let $0 = u_1 < u_2 < \cdots < u_q$ be the dates when some resource is supplied. Then in the corresponding instance $I_D := f(I_M)$ of DTP_q^r, the due-dates are $u'_1 = u_q - u_q = 0$, $u'_2 = u_q - u_{q-1}$, $\ldots$, $u'_q = u_q - u_1 = u_q$. Let σ_D be the order of jobs any solution of instance I_D. The inverse transformation g consists of reversing σ_D. Then, by Lemma 9 we have

$$C_{\max}(S(\sigma_D^{-1})) - u_q + u_q \le T_{\max}(S(\sigma_D)) + u_q = T^s_{\max}(\sigma_D) = (1+\varepsilon)(T^s_{\max}(I_D))^*$$
$$= (1+\varepsilon)(T_{\max}(I_D)^* + u_q) = (1+\varepsilon)((C^*_{\max}(I_M) - u_q) + u_q),$$

where $\varepsilon \ge 0$ is chosen such that $T^s_{\max}(\sigma_D) = (1+\varepsilon)(T^s_{\max}(I_D))^*$, and the second equation follows from $u'_q = u_q$ and $u'_1 = 0$.

Now we prove that there is a Strict-reduction from DTP_q^r to MCP_q^r. We use the transformation of Lemma 9 to construct the function f which maps instances of DTP_q^r to that of MCP_q^r. Let I_D be any instance of DTP_q^r with due-dates $0 \le u_1 < \cdots < u_q$. Then in the corresponding instance $I_M := f(I_D)$ of MCP_q^r, $u'_1 = u_q - u_q = 0,\ldots$, $u'_q = u_q - u_1$. Let σ_M be the order of jobs in any solution to I_M. The inverse transformation g reverses the order of jobs in σ_M. We use Lemma 10 to derive

$$T^s_{max}(S(\sigma_M^{-1})) = T_{max}(S(\sigma_M^{-1})) + u_q - u_1 \le C_{\max}(S(\sigma_D)) - u'_q + (u_q - u_1)$$
$$= (1+\varepsilon)C^*_{\max}(I_M) = (1+\varepsilon)(T^*_{\max}(I_D) + u'_q) = (1+\varepsilon)(T^*_{\max}(I_D) + u_q - u_1),$$

where $\varepsilon \ge 0$ is chosen such that $C_{\max}(S(\sigma_D)) = (1+\varepsilon)C^*_{\max}(I_M)$. □

As a consequence, if we manage to get some kind of approximation algorithm from MCP_q^r, then this yields immediately essentially the same algorithm for DTP_q^r with the shifted delivery tardiness objective, and vice versa. Therefore, from now on, we deal with variants of MCP_q^r only.

3.1.6 Reductions between KP and MCP_2^1

In this section we prove that there is a Strict-reduction from the problem MCP_2^1 to the knapsack problem, and there is an FPTAS-reduction in the opposite direction. Since every Strict-reduction is an FPTAS-reduction as well, we can use the Strict-reduction to the knapsack problem to obtain an FPTAS for MCP_2^1 by using any known FTPAS for the knapsack problem.

First, consider the Material Consumption Problem. Let S be a schedule for $1|nr = 1, q = 2|C_{\max}$ (MCP_2^1). We say that *a job j is assigned to the time point u_1* if and only if the total requirement of the jobs that start not later than j in S is at most b_1. Let $P_1(S)$ denote the sum of processing times of these jobs and $P_2(S)$ denote the total processing time of the remaining jobs. Clearly, $P_1(S) + P_2(S) = \overline{P}$, where $\overline{P} := \sum_{j=1}^{n} p_j$.

Observation 1. *[43] Let S^* be an optimal schedule for MCP_2^r. We have*

i) $C^*_{\max} = \max\{P_1(S^*) + P_2(S^*), u_2 + P_2(S^*)\}$.

ii) $C^*_{\max} \geq \overline{P}$ *and* $C^*_{\max} > u_2$.

Proof. Notice that

i) $P_1(S^*) \geq u_2$ implies $C^*_{\max} = P_1(S^*) + P_2(S^*)$ and $P_1(S^*) < u_2$ implies $C^*_{\max} = u_2 + P_1(S^*)$.

ii) $C^*_{\max} \geq \overline{P}$ is obvious from the previous point, and $C^*_{\max} > u_2$ holds because of Assumption 1.

□

Lemma 11. *[43] Consider the following two problems :*

Knapsack Problem (KP): *There are n items with profits v_j, item weights w_j $(j = 1, \ldots, n)$, and the knapsack has a capacity of b'.*

Material consumption problem: $1|nr = 1, q = 2|C_{\max}$ *(MCP_2^1) with processing times p_j, resource requirements a_j $(j = 1, \ldots, n)$, and supply dates $0 = u_1 < u_2$, and amount of resource supplied b_1 and b_2 at u_1 and u_2, respectively.*

Suppose $p_j = v_j$, $a_j = w_j$ $(\forall j \in \mathcal{J})$, $\tilde{b}_1 = b'$ and $\tilde{b}_2 = \sum_j a_j - \tilde{b}_1$. Let OPT_{KP} denote the optimum value of KP, and $C^*_{\max}$ that of the Material consumption problem.

i) If $P_1(S^*) < u_2$ for some optimal schedule S^* of the scheduling problem, then $P_1(S') < u_2$, $C^*_{\max} = u_2 + P_2(S')$ and $OPT_{KP} = P_1(S')$ for every optimal schedule S'.

ii) If $P_1(S^*) \geq u_2$ for an optimal schedule S^*, then $C^*_{\max} = P_1(S') + P_2(S') = \overline{P}$ for every optimal schedule S', and $OPT_{KP} \geq u_2$.

Proof. i) Firstly, notice that $P_1(S') = P(S^*)$ for every optimal schedule S', because if there were an optimal schedules S' such that $P_1(S^*) < P_1(S')$, then $P_2(S^*) > P_2(S')$ would follow, and thus $C^*_{\max} = C_{\max}(S^*) = u_2 + P_2(S^*) > \max\{u_2 + P_2(S'), P_1(S') + P_2(S')\} = C_{\max}(S')$, which contradicts the optimality of S^*. Since $P_2(S') = \overline{P} - P_1(S')$, $C^*_{\max} = u_2 + P_2(S')$ follows.

Consider an optimal schedule S^*. Pack the items to the knapsack that correspond to the jobs assigned to u_1 in schedule S^*. Since $b' = \tilde{b}_1$, this is a feasible packing and the total profit is $P_1(S^*)$, therefore $OPT_{KP} \geq P_1(S^*)$.

It remains to prove $OPT_{KP} \leq P_1(S^*)$. Let $\mathcal{K}$ denote the set of the packed items in an optimal solution of KP. Now we build a new schedule S' by scheduling the jobs that correspond to the items in $\mathcal{K}$ in arbitrary order from $t = 0$ without any gaps, and schedule the remaining jobs in arbitrary order from $t = \max\{u_2, p(\mathcal{K})\}$ without any gaps. Since $\tilde{b}_1 = b'$, S' is feasible, hence, $C_{\max}(S') = \max\{u_2 + P_2(S'), P_1(S') + P_2(S')\} \geq u_2 + P_2(S^*) = C_{\max}(S^*)$. Since $P_1(S') + P_2(S') = \overline{P} < u_2 + P_2(S^*)$, as $P_1(S^*) < u_2$ by assumption, we must have $C_{\max}(S') = u_2 + P_2(S')$, and therefore, $OPT_{KP} = p(\mathcal{K}) = P_1(S') \leq P_1(S^*)$.

ii) The first part of the statement is trivial. For the second part consider the schedule S^*. Pack those items into the knapsack that correspond to the jobs assigned to u_1 in schedule S^*. Since S^* is a feasible schedule and $b' = b_1$, this yields a feasible packing for KP of profit $P_1(S^*)$, and thus $OPT_{KP} \geq P_1(S^*)$. Since $P_1(S^*) \geq u_2$ by assumption, we deduce $OPT_{KP} \geq P_1(S^*) \geq u_2$.

□

The first main result of this section is a strict reduction from MCP^1_2 to KP. That is, we show that any instance I of MCP^1_2 can be mapped to an instance $f(I)$ of KP in such a way that any solution y of $f(I)$ can be mapped back to a solution $g(I, y)$ of MCP^1_2 with the property that the ratio of the value of the solution $g(I, y)$ and the value of an optimal solution to I is not

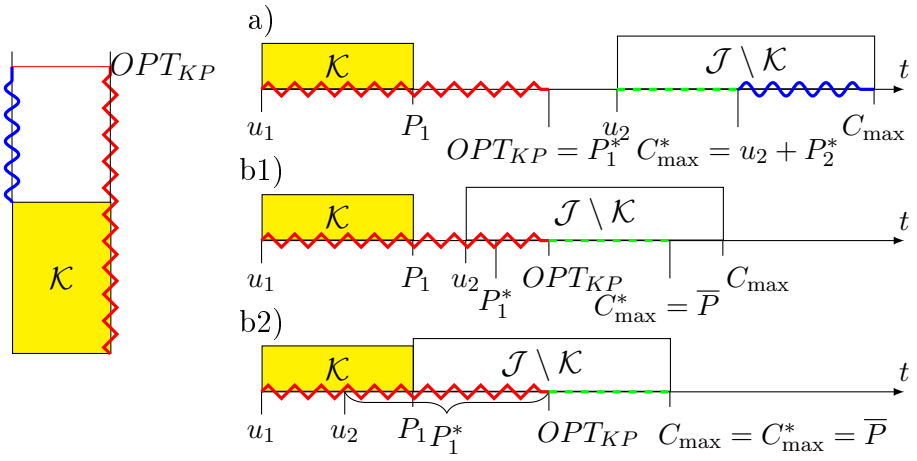

Figure 3.3: The corresponding solutions of KP and MCP_2^1. On the left: the approximate and optimal solutions of KP (the height indicates the value of a solution). On the right: the approximate and optimal solution of MCP_2^1 in case of a) $P_1^* < u_2$, b1) $P_1^* \geq u_2 > P_1$ and b2) $P_1, P_1^* \geq u_2$. The length of the red zigzag line equals OPT_{KP}, that of the blue wavy line equals $OPT_{KP} - \sum_{j \in \mathcal{K}} p_j$, and the length of the green dashed line is $\overline{P} - OPT_{KP}$.

greater than the ratio of the optimum value of $f(I)$ and the value of the solution y. The idea of the transformation is shown in Figure 3.3. There is a one-to-one correspondence between the jobs of the scheduling problem, and the items of the corresponding instance of the knapsack problem. Moreover, if $\mathcal{K}$ is the set of items packed into the knapsack in a feasible solution of the KP problem instance, then the corresponding jobs are scheduled consecutively from time 0 on, and the remaining jobs from time $\max\{u_2, \sum_{j \in \mathcal{K}} p_j\}$ on. Let $P_k := P_k(g(I, y))$ and $P_k^* := P_k(S^*)$ for $k = 1, 2$ where S^* is an optimal schedule to I. The three schedules on the right of Figure 3.3 depend on the relations between P_1, P_1^*, and u_2, and will be elaborated in the proof of the next statement.

Theorem 6. *[43]* $MCP_2^1 \leq_{Strict} KP$.

Proof. Firstly, we define functions f and g. For a given instance $I = \{n, (p_j, a_j)_{j=1}^n, (u_\ell, b_\ell)_{\ell=1}^2\}$ of MCP_2^1, let $f(I) := \{n, (v_j, w_j)_{j=1}^n, b'\}$ be an instance of KP with $v_j = p_j$, $w_j = a_j$, $j = 1, \ldots, n$, and $b' = b_1$. For a given feasible solution y of instance $f(I)$ of KP, let $\mathcal{K}$ be the set of items that are packed into the knapsack. Define a solution $g(I, y)$ of the Material consumption problem as follows: schedule the jobs that correspond to the items in $\mathcal{K}$ in arbitrary order from time $t = 0$ without any gaps. Define $p(\mathcal{K}) := \sum_{j \in \mathcal{K}} v_j$

which equals $\sum_{j\in\mathcal{K}} p_j$ by the definition of the v_j. Schedule the remaining jobs in arbitrary order after $\max\{u_2, p(\mathcal{K})\}$ without any gaps. Since $b' = b_1$, $g(I, y)$ is a feasible solution of the scheduling problem, and let $C_{\max}$ denote its makespan.

Let y be an approximate solution to $f(I)$. It suffices to prove that for any solution y to the instance $f(I)$ of KP, $R_{MCP_2^1}(I, g(I, y)) \leq R_{KP}(f(I), y)$. Let $\varepsilon \geq 0$ be such that $R_{KP}(f(I), y) = 1/(1-\varepsilon)$. Since $R_{KP}(f(I), y) \geq 1$, ε is well defined, and $\varepsilon < 1$. It is enough to show that $R_{MCP_2^1}(I, g(I, y)) \leq 1+\varepsilon$, since $1+\varepsilon < 1/(1-\varepsilon)$ for any $0 \leq \varepsilon < 1$. Let S^* be an optimal schedule, $P_k^* := P_k(S^*)$ for $k = 1, 2$. Let $P_1 := p(\mathcal{K})$, and $P_2 := \overline{P} - P_1$. Using Lemma 11, we distinguish between two cases:

a) $P_1^* < u_2$: in this case $C_{\max}^* = u_2 + P_2^*$, and $OPT_{KP} = P_1^*$ (see Figure 3.3a) for illustration). By the definition of ε, $P_1 = (1-\varepsilon)OPT_{KP}$. Therefore, $P_2 = \overline{P} - (1-\varepsilon)OPT_{KP}$. Since $P_1^* < u_2$ by assumption, we have $C_{\max} = u_2 + P_2 = u_2 + \overline{P} - (1-\varepsilon)OPT_{KP}$. Since $P_2^* = \overline{P} - P_1^* = \overline{P} - OPT_{KP}$, we have $C_{\max}^* = u_2 + \overline{P} - OPT_{KP}$, hence $C_{\max} = C_{\max}^* + (1-(1-\varepsilon))OPT_{KP} \leq (1+\varepsilon)C_{\max}^*$.

b) $P_1^* \geq u_2$: in this case $C_{\max}^* = P_1^* + P_2^* = \overline{P}$, and $OPT_{KP} \geq u_2$, thus $p(\mathcal{K}) \geq (1-\varepsilon)u_2$ (see Figure 3.3 b1) and b2) for illustration). Then $P_2 \leq \overline{P} - (1-\varepsilon)u_2$. Notice that $C_{\max} = \max\{P_1 + P_2; u_2 + P_2\}$ by Observation 1. Since $P_1 + P_2 = \overline{P} = C_{\max}^*$, we only have to prove that $u_2 + P_2 \leq (1+\varepsilon)C_{\max}^*$: $u_2 + P_2 \leq u_2 + \overline{P} - (1-\varepsilon)u_2 = \overline{P} + (1-(1-\varepsilon))u_2 \leq (1+\varepsilon)C_{\max}^*$.

Finally, notice that both of the transformations f and g take linear time and space in the size of I. □

Corollary 4. *[43] There is an FPTAS for MCP_2^1 in $O(n\cdot\min\{\log n, \log(1/\varepsilon)\} + (1/\varepsilon^2)\log(1/\varepsilon)\cdot\min\{n, (1/\varepsilon)\log(1/\varepsilon)\})$ time and in $O(n + 1/\varepsilon^2)$ space.*

Proof. Since every Strict-reduction is an FPTAS-reduction and there is an FPTAS for KP (see e.g. [53]) we can use Lemma 26 to obtain an FPTAS for MCP_2^1. Since the currently best FPTAS for KP requires $O(n\cdot\min\{\log n, \log(1/\varepsilon)\} + (1/\varepsilon^2)\log(1/\varepsilon)\cdot\min\{n, (1/\varepsilon)\log(1/\varepsilon)\})$ time and $O(n + 1/\varepsilon^2)$ space (see [60]), and the transformations f and g take linear time and space, we have proved the complexity results. □

Corollary 5. *[43] There is an 3/2-approximation algorithm for MCP_2^1 of time complexity $O(n\log n)$.*

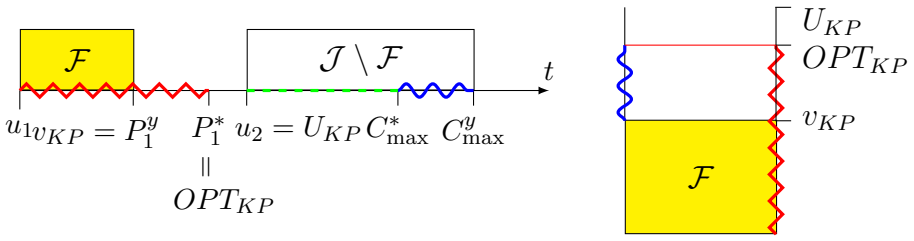

Figure 3.4: The corresponding solutions of MCP_2^1 (on the left) and KP (on the right, the height of a solution indicates its value). The length of the red zigzag line equals $P_1^* = OPT_{KP}$, that of the blue wavy line equals $OPT_{KP} - v_{KP}$, and the length of the green dashed line is $\overline{P} - OPT_{KP}$.

Proof. We have shown in the proof of Theorem 6 that if KP admits a $(1/(1-\varepsilon))$-approximation algorithm ($\mathcal{A}$) then MCP_2^1 admits an $(1+\varepsilon)$-approximation algorithm. The complexity of this algorithm is that of $\mathcal{A}$ plus the linear time transformation. Let $\varepsilon := 1/2$ and use the 2-approximation algorithm for KP (see e.g. [61]). □

Notice that one can create other approximation algorithms for MCP_2^1 if we transform other algorithms originally devised for KP (for an overview of these algorithms see [61]).

Before we prove a converse reduction, observe that there is an easily computable upper bound U_{KP} on the optimum value of KP with $OPT_{KP} \leq U_{KP} \leq 2 \cdot OPT_{KP}$. Let $e_j := v_j/w_j$ denote the efficiency of item j. Sort the items by their efficiency in decreasing order (assume that $e_1 \geq \ldots \geq e_n$). Let k be the smallest index such that $w_1 + \cdots + w_k \geq b'$, unless $\sum_{j=1}^n w_j < b'$ in which case $k := n$, and let $U_{KP} := v_1 + \cdots + v_k$. Further on, $n \cdot OPT_{KP} \geq \sum_{j=1}^n v_j$.

Theorem 7. *[43] $KP \leq_{FPTAS} MCP_2^1$.*

Proof. Let us define functions f and g as follows. For a given instance $I = \{n, (p'_j, w'_j)_{j=1}^n, b'\}$ of KP, let $f(I,\varepsilon) := \{n, (p_j, a_j)_{j=1}^n, (u_\ell, \tilde{b}_\ell)_{\ell=1}^2\}$ be an instance of MCP_2^1 with $p_j = p'_j$, $a_j = w'_j$, $j = 1, \ldots, n$, $\tilde{b}_1 = b', \tilde{b}_2 = \sum_{j=1}^n w'_j - b', u_1 = 0, u_2 = U_{KP}$ (where U_{KP} is an upper bound for OPT_{KP} with $OPT_{KP} \leq U_{KP} \leq 2 \cdot OPT_{KP}$, see section 3.1.2). For a given feasible solution y of instance $f(I,\varepsilon)$ of MCP_2^1, let $\mathcal{F}$ be the set of jobs that are assigned to u_1 in y. Define a solution $g(I,y,\varepsilon)$ of the Knapsack Problem as follows: put the items into the knapsack that correspond to the jobs in $\mathcal{F}$. Let v_{KP} denote the total profit of the items in $\mathcal{F}$. See Figure 3.4 for illustration.

Since $\tilde{b}_1 = b'$, $g(I, y, \varepsilon)$ is a feasible solution for KP. Notice that the transformation of instance x to $f(I)$ and that of the solution of $f(I, \varepsilon)$ back to a solution of x all take linear time and space in the size of I.

Let $\alpha(I, \varepsilon) := \varepsilon/((1+\varepsilon)(n+1))$, and suppose that y is an $\alpha(I, \varepsilon)$-approximate solution (schedule) to $f(I, \varepsilon)$. We have to show that $g(I, y, \varepsilon)$ is an $(1+\varepsilon)$-approximate solution for KP. Notice that $1/\alpha(I, \varepsilon) = (n+1)(1+\varepsilon)/\varepsilon$ is bounded by a polynomial in $|I|$ and $1/\varepsilon$ for any constant bound on ε (cf. Remark 4 after the definition of the FPTAS-reduction in Section 5.4) of the Appendix. Let $C^y_{\max}$ denote the makespan of the approximate solution y, $P^y_k := P_k(y)$ for $k = 1, 2$, and $\delta := \varepsilon/((1+\varepsilon)(n+1))$. Let S^* be an optimal solution to the scheduling problem of makespan $C^*_{\max}$, and let $P^*_k := P_k(S^*)$ for $k = 1, 2$.

We know that $OPT_{KP} \leq U_{KP}$, thus $P^*_1 \leq u_2$, $C^*_{\max} = u_2 + P^*_2$ (see Lemma 11) and $C^y_{\max} = u_2 + P^y_2 \leq (1+\delta)C^*_{\max}$. We have $v_{KP} = P^y_1 = \overline{P} - P^y_2 = \overline{P} + u_2 - C^y_{\max}$. Since $OPT_{KP} = P^*_1$ from Lemma 11, thus $C^*_{\max} = u_2 + \overline{P} - OPT_{KP}$, therefore $v_{KP} \geq \overline{P} + u_2 - (1+\delta)C^*_{\max} = \overline{P} + u_2 - (1+\delta)u_2 - (1+\delta)\overline{P} + (1+\delta)OPT_{KP} = -\delta\overline{P} - \delta u_2 + (1+\delta)OPT_{KP}$. Since $u_2 = U_{KP} \leq 2OPT_{KP}$, $\overline{P} \leq n \cdot OPT_{KP}$ and $\delta > 0$, we deduce that $v_{KP} \geq -\delta n \cdot OPT_{KP} - 2\delta OPT_{KP} + (1+\delta)OPT_{KP} = (1-(n+1)\delta)OPT_{KP} = (1-\varepsilon/(1+\varepsilon))OPT_{KP} = OPT_{KP}/(1+\varepsilon)$. □

Remark 1. *Since the best FPTAS for MCP^1_2 is built on the best FPTAS for KP, this theorem does not have any practical use. However, we can draw an important conclusion from a generalized version of this result for MCP^r_2 (see Corollary 9).*

Corollary 6. *[43] $DTP^1_2 \leq_{Strict} KP$ and $KP \leq_{FPTAS} DTP^1_2$.*

Proof. It is a trivial corollary from Lemmas 27, 28 and from Theorems 5, 6 and 7. □

From this we get the following, like we have got Corollaries 4 and 5 from Theorem 6:

Corollary 7. *[43] There is an FPTAS for DTP^1_2 in $O(n \cdot \min\{\log n, \log(1/\varepsilon)\} + (1/\varepsilon^2)\log(1/\varepsilon) \cdot \min\{n, (1/\varepsilon)\log(1/\varepsilon)\})$ time and in $O(n + 1/\varepsilon^2)$ space, it requires $O(n^7 \cdot 1/\varepsilon^4))$. There is an 3/2-approximation algorithm for DTP^1_2 of time complexity $O(n \log n)$.*

3.1.7 Reductions between r-DKP and MCP_2^r

It is easy to generalize the results of the previous sections: there are very similar connections between the problems r-DKP and MCP_2^r. With these results we can prove that there is no FPTAS for the problem MCP_2^r if $r \geq 2$ unless $\mathcal{P} = \mathcal{NP}$.

We generalize Lemma 11 to r-DKP and MCP_2^r:

Lemma 12. *[43] Consider the following two problems :*

r-Dimensional Knapsack Problem (r-DKP): *There are n items with profits v_j, item weights w_{ij} ($i = 1, \ldots, r$; $j = 1, \ldots, n$), and there are capacities of b'_i ($i = 1, \ldots, r$).*

Material Consumption Problem: $1|nr = r, q = 2|C_{\max}$ *(MCP_2^r) with processing times p_j, resource requirements a_{ij} ($i = 1, \ldots, r$; $j = 1, \ldots, n$), and supply dates $0 = u_1 < u_2$, and amount of resource i supplied $b_{1,i}, b_{2,i}$ at u_1 and u_2, respectively.*

Suppose $p_j = v_j$, $a_{ij} = w_{ij}$ ($\forall i \in \mathcal{R}$ and $\forall j \in \mathcal{J}$), $\tilde{b}_{1,i} = b'_i$ and $\tilde{b}_{2i} = \sum_j a_{ij} - \tilde{b}_{1,i}$ ($\forall i \in \mathcal{R}$). Let OPT_{r-DKP} denote the optimum value of r-DKP, and $C^*_{\max}$ that of the Material consumption problem.

i) If $P_1(S^*) < u_2$ for some optimal schedule S^* of the scheduling problem, then $P_1(S') < u_2$, $C^*_{\max} = u_2 + P_2(S')$ and $OPT_{r-DKP} = P_1(S')$ for every optimal schedule S'.

ii) If $P_1(S^*) \geq u_2$ for an optimal schedule, then $C^*_{\max} = P_1(S') + P_2(S') = \overline{P}$ for every optimal schedule S', and $OPT_{r-DKP} \geq u_2$.

Theorem 8. *[43]* $MCP_2^r \leq_{Strict} r - DKP$

The proof is identical to that of Theorem 6.

Corollary 8. *[43] For any fixed r, there is a PTAS for MCP_2^r.*

The corollary follows from a result of [16], which provides a PTAS for r-DKP for any fixed r.

Theorem 9. *[43]* $r\text{-}DKP \leq_{FPTAS} MCP_2^r$.

The proof is very similar to that of Theorem 7, the crucial difference being that we use Lemma 12 instead of Lemma 11. That is, let $U_{r-DKP} := \sum_{j=1}^{n} v_j$ be a trivial upper bound on the optimum value OPT_{r-DKP} of $r-DKP$. Now, let $u_2 = U_{r-DKP}$ in the transformation of an instance of $r - DKP$ to that of MCP_2^r, and we use the bound $U_{r-DKP} \leq n \cdot OPT_{r-DKP}$ in the proof. Remark 2 shows what we can prove exactly:

Remark 2. *For any $\varepsilon > 0$, if MCP_2^r admits an $\left(1 + \frac{\varepsilon}{(2n-1)(1+\varepsilon)}\right)$-approximation algorithm, then there is an $(1+\varepsilon)$-approximation algorithm for r-DKP.*

Corollary 9. *[43] If $r \geq 2$ then there is no FPTAS for MCP_2^r unless $\mathcal{P} = \mathcal{NP}$.*

Proof. If there were an FPTAS for MCP_2^r, then there would exist an FPTAS for $r - DKP$ by Lemma 26 and Theorem 9. However, there is no FPTAS for $2 - DKP$ unless $\mathcal{P} = \mathcal{NP}$ (see e.g., [38]), a contradiction. □

Corollary 10. *[43] $DTP_2^r \leq_{Strict} r - DKP$ and $r - DKP \leq_{FPTAS} DTP_2^r$.*

Proof. Follows from Lemmas 27, 28 and from Theorems 5, 8 and 9. □

Corollary 11. *[43] For any fixed r, there is a PTAS for DTP_2^r. If $r \geq 2$ then there is no FPTAS for DTP_2^r unless $\mathcal{P} = \mathcal{NP}$.*

3.1.8 Approximability of $1|nr|C_{\max}$

In this section we present approximation schemes, and an inapproximability result for single machine problems with resource consuming jobs with the makespan objective. Notice that these algorithms immediately give rise to approximation algorithms for the delivery tardiness problem with the same guarantee and complexity, by Theorem 5. We emphasize that prior to our work, the only constant factor approximation algorithm for $1|nr = 1|C_{\max}$ was that of Grigoriev et al [41] with a worst-case guarantee of 2.

For the problem $1|nr = const, q = const|C_{\max}$ we present a PTAS based on LP-rounding. We will also prove that if the number of resources is not a constant, then $1|nr, q = 2|C_{\max}$ is APX-complete. Before all these approximability results, we discuss the computational complexity of $1|nr|C_{\max}$. The first complexity results have been established by Carlier [19], we mention only two of them:

Proposition 7 (Carlier [19])**.** *Problem $1|nr = 1, q = 2|C_{\max}$ is $\mathcal{NP}$-hard in the ordinary sense.*

Proposition 8 (Carlier [19])**.** *Problem $1|nr = 1|C_{\max}$ is $\mathcal{NP}$-hard in the strong sense.*

In Györgyi and Kis [42] a pseudo-polynomial time algorithm is proposed when both the number of resources and that of the supply dates are constants.

Proposition 9. *[42]* $1|nr = const, q = const|C_{\max}$ can be solved in pseudo-polynomial time.*

The latter algorithm directly leads to an FPTAS for $1|nr = 1, q = 2|C_{\max}$ as described in the same paper. However, an FPTAS exists for the same problem with a much better time complexity by reducing it to the knapsack problem, see Corollary 4. Unfortunately, it is unknown if an FPTAS exists for $1|nr = 1, q = const|C_{\max}$, i.e., if we have a constant number of supply periods, where the constant is greater than 2.

Inapproximability of $1|nr, q = 2|C_{\max}$

In this section we will prove that if the number of resources is part of the input (not a constant), then there is a small constant $\varepsilon_0 > 0$ such that it is $\mathcal{NP}$-hard to approximate $1|nr, q = 2|C_{\max}$ better that $1 + \varepsilon_0$. To this end, we will reduce the APX-complete Vertex Cover Problem in Bounded-degree graphs (*VERTEX-COVER-B*) to $1|nr, q = 2|C_{\max}$. In VERTEX-COVER-B, given a connected graph and a constant B, such that the degree of any vertex is bounded by B. One has to find a subset U of vertices of minimum cardinality such that each edge of the graph is adjacent to some vertex in U. The following result was shown by [90]:

Theorem 10. *For any constant $B \geq 4$, there is a constant $\delta > 0$, such that it is $\mathcal{NP}$-hard to approximate VERTEX-COVER-B better than $1 + \delta$.*

We can state the following well-known observation.

Proposition 10. *In a connected graph on n vertices in which the maximum degree of any vertex is B, the size of the minimum vertex cover is at least $\lceil (n-1)/B \rceil$.*

Now we are ready to prove the main result of this section.

Theorem 11. *[44]* There is some constant $\varepsilon > 0$ such that it is $\mathcal{NP}$-hard to approximate the problem $1|nr, q = 2|C_{\max}$ better than $1 + \varepsilon$ if the number of resources is part of the input.*

Proof. We will transform instances of VERTEX-COVER-B to instances of $1|nr, q = 2|C_{\max}$ and show that if the latter problem admits a polynomial time $(1 + \varepsilon)$-approximation algorithm for any $\varepsilon > 0$, then VERTEX-COVER-B has a $(1+(B+1)\varepsilon)$-approximation algorithm for any $\varepsilon > 0$, which contradicts Theorem 10.

In the course of the transformation, we map a connected graph $G = (V, E)$ of maximum degree B with $n = |V|$ vertices and $m = |E|$ edges to a scheduling problem instance with n jobs and $r = m$ non-renewable resources,

and $q = 2$ supply dates. For each vertex $v \in V$ of the graph, we define one job, denoted by J_v, of processing time 1, and for each edge $e \in E$ we define one resource R_e. The resource requirements of job J_v are determined by the edges adjacent to the corresponding vertex v, that is, if edge e is adjacent to v, then job J_v requires one unit of the resource R_e. Let $u_1 = 0$, $u_2 = n - 1$, $\tilde{b}_{1,e} = 1$, and $\tilde{b}_{2,e} = 1$ for each resource R_e, i.e., from each resource one unit is supplied at time 0, and one more unit at time u_2.

What does a schedule of minimum length represent in the graph? Observe that in any feasible schedule, in the time interval u_1 and u_2 no two jobs requiring the same resource may be scheduled, since the initial supply from each resource is 1 unit. Let I be the set of vertices indexing those jobs scheduled between u_1 and u_2. Since no two jobs J_v and J_w with $v \neq w \in I$ may require the same resource (since $\tilde{b}_{1,e} = 1$ for each resource R_e), nodes v and w are not adjacent in the graph. Hence, I is an independent set in G. But then the vertices of G indexing those jobs scheduled after u_2 constitute a vertex cover of G (since the complement of an independent set is a vertex cover in any graph). Since $u_2 = (n - 1)$, and the graph is connected, in a schedule of minimum length, the vertices of G indexing those jobs scheduled after u_2 constitute an optimal vertex cover of G. Let vc^* denote the size of a minimum vertex cover of G. Then, the optimum makespan is $C^*_{\max} = u_2 + vc^*$.

Now we claim that if there is an $(1 + \varepsilon)$-approximation algorithm for $1|nr, q = 2|C_{\max}$, then there is a $(1 + (B + 1)\varepsilon)$-approximation algorithm for VERTEX-COVER-B. So suppose we have an $(1+\varepsilon)$-approximation algorithm for $1|nr, q = 2|C_{\max}$. The schedule supplied by this algorithm on the set of instances constructed above satisfies the following:

$$C_{\max} = (n - 1) + k \leq ((n - 1) + vc^*)(1 + \varepsilon),$$

where k is the number of jobs scheduled after $u_2 = (n - 1)$. Notice that k is the size of the vertex cover in G determined by the jobs scheduled after u_2. After rearranging terms we get

$$k \leq (n - 1)\varepsilon + vc^*(1 + \varepsilon).$$

Finally, using Proposition 10 we obtain

$$k \leq \frac{(n - 1)}{B}(\varepsilon B) + vc^*(1 + \varepsilon) \leq vc^*(1 + (B + 1)\varepsilon).$$

That is, the schedule determines a vertex cover of size at most $vc^*(1 + (B + 1)\varepsilon)$. □

Notice that in the above proof we have provided an L-reduction from VERTEX-COVER-B to $1|nr|C_{\max}$ with parameters $\alpha = (B + 1)$, and $\beta = 1$.

Since the problem $1|nr|C_{\max}$ admits a 2-approximation algorithm [41], and $1|nr, q = 2|C_{\max}$ is just a special case, we can deduce the following:

Corollary 12. *[44]* $1|nr, q = 2|C_{\max}$ *is APX-complete.*

PTAS for $1|nr = const, q = const, r_j|C_{\max}$ based on LP-rounding

In this section we describe a polynomial time approximation scheme for scheduling resource consuming jobs on a single machine and with a constant number of non-renewable resources and supply dates. Firstly, we will assume that the number of distinct job release dates *before* u_q is bounded by a constant, and then we will waive this extra constraint. The objective is to minimize the makespan. The PTAS presented is based on LP-rounding, however, for the problem $1|nr = 1, q = const, r_j|C_{\max}$ there is a PTAS based purely on combinatorial enumeration, for details see Györgyi and Kis [42].

We can model our scheduling problem by means of a mathematical program. To this end, firstly we construct a set of time points $\mathcal{T}$ consisting of all the *distinct values* from the set of time moments u_ℓ, $\ell = 1, \ldots, q$, (when some non-renewable resource is supplied), and the set of release dates of the jobs r_j, $j \in \mathcal{J}$. Suppose $\mathcal{T}$ has τ elements, denoted by v_1 through v_τ, with $v_1 = 0$. We define the values $b_{\ell i} := \sum_{k\,:\,u_k \le v_\ell} \tilde{b}_{ki}$ for $i \in \mathcal{R}$, that is, $b_{\ell i}$ equals the total amount supplied from resource i up to time point v_ℓ.

We introduce $\tau \cdot |\mathcal{J}|$ binary decision variables $x_{j\ell}$, $(j \in \mathcal{J}, \ell = 1, \ldots, \tau)$ such that $x_{j\ell} = 1$ if and only if job j is assigned to the time point v_ℓ, which means that the requirements of job j must be satisfied by the resource supplies up to time point v_ℓ. The mathematical program is

$$C^*_{\max} = \min \max_{v_\ell \in \mathcal{T}} \left(v_\ell + \sum_{j \in \mathcal{J}} \sum_{\nu=\ell}^{\tau} p_j x_{j\nu} \right) \tag{3.17}$$

s.t.

$$\sum_{j \in \mathcal{J}} \sum_{\nu=1}^{\ell} a_{ij} x_{j\nu} \le b_{\ell i}, \quad v_\ell \in \mathcal{T},\ i \in \mathcal{R} \tag{3.18}$$

$$\sum_{\ell=1}^{\tau} x_{j\ell} = 1, \quad j \in \mathcal{J} \tag{3.19}$$

$$x_{j\ell} = 0, \quad j \in \mathcal{J},\ v_\ell \in \mathcal{T} \text{ such that } r_j > v_\ell \tag{3.20}$$

$$x_{j\ell} \in \{0, 1\}, \quad j \in \mathcal{J},\ v_\ell \in \mathcal{T}. \tag{3.21}$$

The objective function expresses the completion time of the job finished last using the observation that there is a time point, either a release date of some

job, or when some resource is supplied, from which the machine processes the jobs without idle times. Constraints (3.18) ensure that the jobs assigned to time points v_1 through v_ℓ use only the resources supplied up to time v_ℓ. Equations (3.19) ensure that all jobs are assigned to some time point. Finally, no job may be assigned to a time point before its release date by (3.20). Any feasible job assignment $\bar{x}$ gives rise to a set of schedules which differ only in the ordering of jobs assigned to the same time point v_ℓ.

Notice that in a feasible solution $\hat{x}$ of (3.17)-(3.21) there can be more than one jobs assigned to the same time point v_ℓ. We obtain a schedule of the jobs by putting them on the machine in the order of their assignment to the time points in $\mathcal{T}$. That is, first we schedule in any order without idle times the jobs with $\hat{x}_{j1} = 1$ from time v_1 on. Let C_1 be the completion time of these jobs. In a general step $\ell \geq 2$, we schedule the jobs with $\hat{x}_{j\ell} = 1$ in any order after $\max\{C_{\ell-1}, v_\ell\}$, and we denote by C_ℓ the completion time of the job finished last in this group. The schedule obtained in this way is feasible, and its makespan is the completion time of the job finished last, which is necessarily equal to the objective function value of solution $\hat{x}$. Let $C_{\max}(\hat{x})$ denote this value.

Let $p_{\text{sum}} := \sum_{j \in \mathcal{J}} p_j$ denote the sum of the processing times of all the jobs. For a fixed $\varepsilon > 0$, let $\mathcal{B} := \{j \in \mathcal{J} \mid p_j \geq \varepsilon p_{\text{sum}}\}$ be the set of big jobs, and $\mathcal{S} := \mathcal{J} \setminus \mathcal{B}$ the set of small jobs. We divide further the set of small jobs according to their release dates, that is, we define the sets $\mathcal{S}^b := \{j \in \mathcal{S} \mid r_j < u_q\}$, and $\mathcal{S}^a := \mathcal{S} \setminus \mathcal{S}^b$. Let $\mathcal{T}^b := \{v_\ell \in \mathcal{T} \mid v_\ell < u_q\}$ be the set of time points v_ℓ before u_q. The following observation reduces the number of solutions of (3.17)-(3.21) to be examined.

Proposition 11. *[44]*From any feasible solution $\hat{x}$ of (3.17)-(3.21), we can obtain a solution $\tilde{x}$ with $C_{\max}(\tilde{x}) \leq C_{\max}(\hat{x})$ such that each job J_j is assigned to some time point v_ℓ ($\tilde{x}_{j\ell} = 1$), satisfying either $v_\ell < u_q$, or $v_\ell = \max\{u_q, r_j\}$.*

Proof. Let $\mathcal{J}^a(\hat{x})$ be the subset of jobs with $\hat{x}_{j\ell} = 1$ for some $v_\ell > u_q$. We define a new solution $\tilde{x}$ in which those jobs in $\mathcal{J}^a(\hat{x})$ are reassigned to new time points and show that $C_{\max}(\tilde{x}) \leq C_{\max}(\hat{x})$. Let $\tilde{x} \in \{0,1\}^{\mathcal{J} \times \mathcal{T}}$ be a binary vector which agrees with $\hat{x}$ for those jobs in $\mathcal{J} \setminus \mathcal{J}^a(\hat{x})$. For each $j \in \mathcal{J}^a(\hat{x})$, let $\tilde{x}_{j\ell} = 1$ for $v_\ell = \max\{u_q, r_j\}$, and 0 otherwise. We claim that $\tilde{x}$ is a feasible solution of (3.17)-(3.21), and that $C_{\max}(\tilde{x}) \leq C_{\max}(\hat{x})$. Feasibility of $\tilde{x}$ follows from the fact that u_q is the last time point when some resource is supplied, and that no job is assigned to some time point before its release date. As for the second claim, consider the objective function (3.17).

We will verify that for each $\ell = 1, \ldots, \tau$,

$$v_\ell + \sum_{j \in \mathcal{J}} \sum_{\nu=\ell}^{\tau} p_j \tilde{x}_{j\nu} \leq v_\ell + \sum_{j \in \mathcal{J}} \sum_{\nu=\ell}^{\tau} p_j \hat{x}_{j\nu}, \tag{3.22}$$

from which the claim follows. If $v_\ell \leq u_q$, the left and the right-hand sides in (3.22) are equal. Now consider any ℓ with $v_\ell > u_q$. Since no job in $\mathcal{J}^a(\hat{x})$ is assigned to a later time point in $\tilde{x}$ than in $\hat{x}$, the inequality (3.22) is verified again. □

Notice that Proposition 11 also follows form a result of Lawler [75], which implies that if we have a single machine and a set of jobs with release dates, and the objective is the minimum makespan, then it is optimal to sequence the jobs in non-decreasing release date order.

An *assignment of big jobs to the time points* v_1 *through* v_τ is given by a partial solution $x^{big} \in \{0,1\}^{\mathcal{B} \times \mathcal{T}}$ which assigns each big job to some time point v_ℓ. An assignment x^{big} of big jobs is *feasible* if the vector $x = (x^{big}, 0) \in \{0,1\}^{\mathcal{J} \times \mathcal{T}}$ satisfies (3.18), (3.20) and also (3.19) for the big jobs. Consider any feasible assignment x^{big} of big jobs. If we fix the assignment of the big jobs in (3.18)-(3.20) to x^{big}, then the supply from any resource i up to time point v_ℓ is decreased by the requirements of those big jobs assigned to time points v_1 through v_ℓ. Hence, we define the *residual resource supply* up to time point v_ℓ as $\bar{b}_{\ell i} := b_{\ell i} - \sum_{j \in \mathcal{B}} a_{ij} \left(\sum_{\mu=1}^{\ell} x_{j\mu}^{big} \right)$. Further on, let $\bar{C}_\ell^{\mathcal{B}} := \max_{\mu=1,\ldots,\ell} (v_\mu + \sum_{\kappa=\mu}^{\ell} \sum_{j \in \mathcal{B}} p_j x_{j\kappa}^{big})$ denote the earliest time point when the big jobs assigned to v_1 through v_ℓ may finish.

In order to assign approximately the small jobs, we will solve a linear program and round its solution. Our linear programming formulation relies on the following result.

Proposition 12. *[44]* There exists an optimal solution* $(\hat{x}^{big}, \hat{x}^{small})$ *of (3.17)-(3.21) such that for each* $v_\ell \in \mathcal{T}^b$*:*

$$\sum_{j \in \mathcal{S}^b} p_j \hat{x}_{j\ell}^{small} \leq \max\{0, v_{\ell+1} - \bar{C}_\ell^{\mathcal{B}}\} + \varepsilon p_{\text{sum}}. \tag{3.23}$$

Proof. Suppose $(\hat{x}^{big}, \hat{x}^{small})$ is an optimal solution which does not meet the property claimed. Without loss of generality, we may assume that in the optimal schedule corresponding to $(\hat{x}^{big}, \hat{x}^{small})$, for each $v_k \in \mathcal{T}$, small jobs assigned to v_k follow the big ones assigned to v_k. Let $v_\ell \in \mathcal{T}^b$ be the smallest time point for which (3.23) is violated. Then some small jobs assigned to v_ℓ necessarily start after $v_{\ell+1}$ in any schedule corresponding to $(\hat{x}^{big}, \hat{x}^{small})$. Since all small jobs are of processing time less than $\varepsilon p_{\text{sum}}$, we can reassign

some of the small jobs from time point v_ℓ to $v_{\ell+1}$ until (3.23) is satisfied for v_ℓ. Clearly, such a reassignment of small jobs does not increase the length of the schedule. Then we proceed with the next time point in $\mathcal{T}$ until we get a schedule meeting (3.23). □

Now, the linear program is defined with respect to any feasible assignment x^{big} of the big jobs:

$$\max \sum_{v_\ell \in \mathcal{T}^b} \sum_{j \in \mathcal{S}^b} p_j x_{j\ell}^{small} \tag{3.24}$$

s.t.

$$\sum_{j \in \mathcal{S}^b} \sum_{\nu=1}^{\ell} a_{ij} x_{j\nu}^{small} \leq \bar{b}_{\ell i}, \quad v_\ell \in \mathcal{T}^b,\ i \in \mathcal{R} \tag{3.25}$$

$$\sum_{j \in \mathcal{S}^b} p_j x_{j\ell}^{small} \leq \max\{0, v_{\ell+1} - \bar{C}_\ell^{\mathcal{B}}\} + \varepsilon p_{\text{sum}}, \quad v_\ell \in \mathcal{T}^b \tag{3.26}$$

$$\sum_{v_\ell \in \mathcal{T}^b \cup \{u_q\}} x_{j\ell}^{small} = 1, \quad j \in \mathcal{S}^b \tag{3.27}$$

$$x_{j\ell}^{small} = 0, \quad j \in \mathcal{S}^b,\ v_\ell \in \mathcal{T} \text{ such that } v_\ell < r_j, \text{ or } v_\ell > u_q \tag{3.28}$$

$$x_{j\ell}^{small} \geq 0, \quad j \in \mathcal{S}^b,\ v_\ell \in \mathcal{T}. \tag{3.29}$$

The objective function (3.24) maximizes the total processing time of those small jobs assigned to some time point v_ℓ *before* u_q. Constraints (3.25) make sure that no resource is overused taking into account the fixed assignment of big jobs as well. Inequalities (3.26) ensure that the small jobs assigned to v_ℓ fit into the interval $[\bar{C}_\ell^{\mathcal{B}}, v_{\ell+1} + \varepsilon p_{\text{sum}})$. Due to (3.27), small jobs are assigned to some time point in $\mathcal{T}^b \cup \{u_q\}$. The release dates of those jobs in $\mathcal{S}^b$, and Proposition 11 are taken care of by (3.28). Finally, we require that the values $x_{j\ell}^{small}$ be non-negative.

Notice that this linear program always has a finite optimum provided that x^{big} is a feasible assignment of the big jobs. Let $\bar{x}^{small}$ be any feasible solution of the linear program. Job $j \in \mathcal{S}^b$ is *integral* in $\bar{x}^{small}$ if there exists $v_\ell \in \mathcal{T}$ with $\bar{x}_{j\ell}^{small} = 1$, otherwise it is *fractional*. After all these preliminaries, the PTAS is as follows.

Algorithm A [44]*

1. Assign the big jobs to time points v_1 through v_τ in all possible ways which satisfies Proposition 11, and for each feasible assignment x^{big} do steps 2-5:

2. Define and solve linear program (3.24)-(3.29), and let $\bar{x}^{small}$ be an optimal basic solution.

3. Round each fractional value in $\bar{x}^{small}$ down to 0, and let $x^{small} := \lfloor \bar{x}^{small} \rfloor$ be the resulting partial assignment of small jobs, and $U \subset \mathcal{S}^b$ the set of fractional jobs in $\bar{x}^{small}$.

4. Finally, each $j \in U$ is assigned to time point $u_q \in \mathcal{T}$ by letting $x^{small}_{j\ell} := 1$ for all $j \in U$ where $v_\ell = u_q$, and all jobs in $\mathcal{S}^a$ are assigned to their release dates by setting $x^{small}_{j\ell} := 1$ for all $j \in \mathcal{S}^a$, where $v_\ell = r_j$ (recall that the release dates of jobs belong to set $\mathcal{T}$).

5. If the value of the complete assignment (x^{big}, x^{small}) of all the jobs is better than the best solution found so far, then update the best solution to (x^{big}, x^{small}).

6. After examining each feasible assignment of big jobs, output the best complete solution found.

We have to verify that the solution found by the above algorithm is feasible for (3.17)-(3.21), its value is not too far from the optimum, and that the algorithm runs in polynomial time in the size of the input. We introduce some terminology to facilitate the following discussion. A 0/1-vector (x^{big}, x^{small}) satisfying constraints (3.18) and (3.20) constitutes a *complete solution* if every job is assigned to some time point v_ℓ, i.e., it satisfies also the equations (3.19), otherwise it is a *partial solution.*

Lemma 13. *[44]*Every complete solution (x^{big}, x^{small}) constructed by the algorithm is feasible for (3.17)-(3.21).*

Proof. Since the algorithm examines only feasible assignments of big jobs, $(x^{big}, 0)$ satisfies (3.18), (3.20), (3.21), and also (3.19) for all jobs in $\mathcal{B}$ by definition. The binary vector $(x^{big}, \lfloor \bar{x}^{small} \rfloor, 0)$ consists of the assignment of big jobs, and of those small jobs in $\mathcal{S}^b$ which are integral in the optimal solution $\bar{x}^{small}$ of the linear program. Notice that the fractional jobs in $\bar{x}^{small}$, and all jobs in $\mathcal{S}^a$ are unassigned. The partial solution $(x^{big}, \lfloor \bar{x}^{small} \rfloor, 0)$ satisfies (3.18), (3.20), (3.21), and also (3.19) for all jobs in $\mathcal{B} \cup \{j \in \mathcal{S}^b \mid j \text{ is integral in } \bar{x}^{small}\}$, since $\bar{x}^{small}$ is a feasible solution of (3.24)-(3.29). Finally, since u_q is the last time point when some resource is supplied and all job in $\mathcal{S}^a \cup U$ are assigned to some time points not before u_q, the 0/1-vector (x^{big}, x^{small}) is feasible for (3.17)-(3.21). □

We will need the following result:

Proposition 13. *[44]*In any basic solution of the linear program (3.24)-(3.29), there are at most* $(|\mathcal{R}|+1)\cdot|\mathcal{T}^b|$ *fractional jobs.*

Proof. Let $\bar{x}^{small}$ be a basic solution of the linear program in which f jobs of $\mathcal{S}^b$ are assigned fractionally, and $e=|\mathcal{S}^b|-f$ jobs integrally. Clearly, each integral job gives rise to precisely one positive value, and each fractionally assigned job to at least two. This program has $|\mathcal{S}^b|\cdot|\mathcal{T}^b|$ decision variables, and $m=|\mathcal{S}^b|+(|\mathcal{R}|+1)\cdot|\mathcal{T}^b|$ constraints. Therefore, in $\bar{x}^{small}$ there are at most m positive values, as no variable may be nonbasic with a positive value. Hence,

$$e+2f\leq|\mathcal{S}^b|+(|\mathcal{R}|+1)\cdot|\mathcal{T}^b|=e+f+(|\mathcal{R}|+1)\cdot|\mathcal{T}^b|.$$

This implies

$$f\leq(|\mathcal{R}|+1)\cdot|\mathcal{T}^b|$$

as claimed. □

Lemma 14. *[44]*The algorithm constructs at least one complete assignment* (x^{big},x^{small}) *whose value is at most* $(1+O(\varepsilon))$ *times the optimum makespan* $C^*_{\max}$.

Proof. Consider an optimal solution $(\hat{x}^{big},\hat{x}^{small})$ of (3.17)-(3.21), consisting of the assignment of big jobs and that of the small jobs. We can suppose that this solution satisfies the condition of Proposition 11. The algorithm will examine $\hat{x}^{big}$, being a feasible assignment of the big jobs. Let $x^a\in\{0,1\}^{\mathcal{S}^a\times\mathcal{T}}$ be an assignment of the small jobs in $\mathcal{S}^a$ with $x^a_{j\ell}=1$ if and only if $r_j=v_\ell$. Let $C_{\max}((\hat{x}^{big},0,x^a))$ be the value of the partial assignment $(\hat{x}^{big},0,x^a)$ in which no job in $\mathcal{S}^b$ is assigned to any time point. Clearly, $C_{\max}((\hat{x}^{big},0,x^a))\leq C_{\max}((\hat{x}^{big},\hat{x}^{small}))=C^*_{\max}$. Now, let us consider the assignment of the small jobs in $\mathcal{S}^b$ constructed by the algorithm. Let $\lfloor\bar{x}^{small}\rfloor\in\{0,1\}^{\mathcal{S}^b\times(\mathcal{T}^b\cup\{u_q\})}$ be the assignment of the small jobs in $\mathcal{S}^b$ assigned by Algorithm A integrally. Let $C_{\max}((\hat{x}^{big},\lfloor\bar{x}^{small}\rfloor,x^a))$ be the value of the partial assignment $(\hat{x}^{big},\lfloor\bar{x}^{small}\rfloor,x^a)$.

Since $\bar{x}^{small}$ is a feasible solution of (3.24)-(3.29), the partial solution $(\hat{x}^{big},\lfloor\bar{x}^{small}\rfloor,x^a)$ may assign small jobs of total processing time at most $\max\{0,v_{\ell+1}-\bar{C}^{\mathcal{B}}_\ell\}+\varepsilon p_{\rm sum}$ to each $v_\ell\in\mathcal{T}^b$ (in addition to the big jobs assigned by x^{big}). Hence, the jobs assigned to time points $v_k\geq u_q$ may be pushed to the right by at most $|\mathcal{T}^b|\varepsilon p_{\rm sum}$ by the small jobs assigned to time points in $\mathcal{T}^b$. Since the linear program maximizes $\sum_{v_\ell\in\mathcal{T}^b}\sum_{j\in\mathcal{S}^b}p_jx_{j\ell}$, thus $\sum_{j\in\mathcal{S}^b,v_\ell=u_q}p_j\bar{x}_{j\ell}\leq\sum_{j\in\mathcal{S}^b,v_\ell=u_q}p_j\hat{x}_{j\ell}$, therefore $C_{\max}((\hat{x}^{big},\lfloor\bar{x}^{small}\rfloor,x^a))\leq C_{\max}((\hat{x}^{big},\hat{x}^{small}))+|\mathcal{T}^b|\varepsilon p_{\rm sum}=C^*_{\max}+|\mathcal{T}^b|\varepsilon p_{\rm sum}$.

Finally, we bound the total processing time of those jobs in set U. By Proposition 13, the size of U is at most $(|\mathcal{R}|+1)\cdot|\mathcal{T}^b|$. But then, the total processing time of those small jobs from $\mathcal{S}^b$ that are moved to the end of the schedule is at most $(|\mathcal{R}|+1)\cdot|\mathcal{T}^b|\cdot\varepsilon p_{\text{sum}}$. To summarize, we have

$$\begin{aligned} C_{\max}((\hat{x}^{big}, x^{small})) &\leq C_{\max}((\hat{x}^{big}, \lfloor \bar{x}^{small} \rfloor, x^a)) + (|\mathcal{R}|+1)\cdot|\mathcal{T}^b|\cdot\varepsilon p_{\text{sum}} \\ &\leq C^*_{\max} + (|\mathcal{R}|+2)\cdot|\mathcal{T}^b|\cdot\varepsilon p_{\text{sum}} \leq (1+O(\varepsilon))C^*_{\max}, \end{aligned}$$

where the first and the second inequality follows from the above discussion, and the last from the fact that both $|\mathcal{R}|$ and $|\mathcal{T}^b|$ are bounded by constants by assumption and from the observation that $p_{\text{sum}} \leq C^*_{\max}$. □

Lemma 15. *[44]* For any fixed $\varepsilon > 0$, the running time of the algorithm is polynomial in the size of the input.*

Proof. Since the processing time of each big job is at least $\varepsilon p_{\text{sum}}$, the number of big jobs is at most $\lfloor 1/\varepsilon \rfloor$, a constant, since ε is constant by assumption. Since the number of time points in $\mathcal{T}$ is also constant by assumption, the total number of assignments of big jobs to time point in $\mathcal{T}$ is also constant. For each feasible assignment, a linear program of polynomial size in the input must be solved. This can be accomplished by the Ellipsoid method in polynomial time [34]. Rounding the solution takes linear time in the number of small jobs. Hence, the entire running time is polynomial in the size of the input, as claimed. □

Theorem 12. *[44]* There is a PTAS for the problem $1|nr = const., q = const., \#\{r_j \ : \ r_j < u_q\} = const.|C_{\max}$.*

Proof. We show that Algorithm A is a PTAS for $1|nr = const, q = const, \#\{r_j \ : \ r_j < u_q\} = const.|C_{\max}$. The polynomial time complexity of the algorithm in the size of the input was shown in Lemma 15. In Lemma 14 it was shown that the performance ratio is $(1+O(\varepsilon))$, where the constant factor c in $O(\cdot)$ does not depend on the input. Hence, to reach a desired performance ratio δ, we let $\varepsilon := \delta/c$, and perform the computations with this choice of ε. □

Now we show how to waive the restriction $\#\{r_j \ : \ r_j < u_q\} = const$ in the previous statement. To this end, for a fixed $\delta > 0$, we modify some of the job release dates to define a new problem instance in which there are at most $1/\delta$ different job release dates before u_q. That is,

$$r'_j := \begin{cases} r_j & \text{if } r_j \geq u_q \\ g(r_j)\cdot\delta u_q & \text{if } r_j < u_q \end{cases}$$

where $g : \mathbb{Z}_{\geq 1} \to \mathbb{Z}_{\geq 1}$, and $g(r_j)$ equals the smallest non-negative integer z such that $z \cdot \delta u_q \geq r_j$. Let $C^r_{\max}(x)$ denote the value of some solution x of the modified problem instance, and $C^{r^*}_{\max}$ the optimum makespan.

Lemma 16. *[44]* $C^{r^*}_{\max} \leq C^*_{\max} + \delta u_q$.

Proof. Consider an optimal schedule of the original problem, and let S^*_j denote the starting time of job $j \in \mathcal{J}$. Let $S'_j = S^*_j + \delta u_q$ for each $j \in \mathcal{J}$. We claim that S' is a feasible schedule for the modified problem with makespan $C^*_{\max} + \delta u_q$. Both claims follow from the following easy observation:

$$r'_j \leq r_j + \delta u_q \leq S^*_j + \delta u_q = S'_j, \ \forall j \in \mathcal{J}.$$

□

Apply the PTAS of the previous section with error parameter δ to the modified problem instance to obtain a δ-approximate solution $\hat{x}^r$. We show that $\hat{x}^r$ is a 3δ-approximate solution for the original problem instance.

Theorem 13. *[44] There is a PTAS for the problem* $1|nr = const, q = const, r_j|C_{\max}$.

Proof. We will show that for any $0 < \delta \leq 1$, $C_{\max}(\hat{x}^r) \leq (1 + 3\delta)C^*_{\max}$, and thus the above algorithm for $1|r_j, rm = const., q = const.|C_{\max}$ is a PTAS.

We have

$$C_{\max}(\hat{x}^r) \leq C^r_{\max}(\hat{x}^r) \leq (1 + \delta)C^{r^*}_{\max} \leq (1 + \delta)(C^*_{\max} + \delta u_q) \leq (1 + 3\delta)C^*_{\max},$$

where the first inequality follows from $r_j \leq r'_j$, the second from the fact that we have applied a PTAS with error parameter δ to the modified problem instance, the third form Lemma 16, and the last one from $u_q < C^*_{\max}$ (by Assumption 1), and from elementary calculations. □

3.1.9 Approximability of $1|nr|\sum w_jC_j$

In this section first we discuss the computational complexity of $1|nr = 1|\sum w_jC_j$ and then we describe an FPTAS for $1|nr = 1, q = 2|\sum w_jC_j$.

To our best knowledge, the only paper dealing with machine scheduling with a non-renewable resource and a min-sum type criterion (the average completion time) is by Gafarov et al. [35], who proved that minimizing the average completion time of the jobs ($1|nr = 1|\sum_j C_j$) is $\mathcal{NP}$-hard in the ordinary sense when the number of supply dates is not a constant. In this section, we will sharpen the result of Gafarov et al. [35] by showing that $1|nr = 1|\sum_j C_j$ is $\mathcal{NP}$-hard in the strong sense. we will also prove that if

there are only $q = 2$ supplies, then the problem $1|nr = 1, q = 2|\sum_j C_j$ is $\mathcal{NP}$-hard in the ordinary sense. Further on, we will describe an FPTAS for this special case.

The scheduling problem studied in this section has the following characteristics. There is a single machine, a set of n jobs $\mathcal{J}$, and a non-renewable resource consumed by the jobs. The machine can process only one job at a time, and preemption of processing is not allowed. Each job J_j, $j \in \mathcal{J}$, has a processing time $p_j \in \mathbb{Z}_{\geq 1}$, a weight $w_j \in \mathbb{Z}_{\geq 0}$, and a resource requirement $a_j \in \mathbb{Z}_{\geq 0}$. The resource is supplied in q different moments in time, $0 = u_1 < u_2 < \ldots < u_q$; the scalar $\tilde{b}_\ell \in \mathbb{Z}_{\geq 1}$, for $\ell = 1, \ldots, q$, represents the quantity supplied at u_ℓ. A *schedule* σ specifies a starting time $S_j \in \mathbb{Z}_{\geq 0}$ for each job $j \in \mathcal{J}$, and it is *feasible* if (i) the jobs do not overlap in time and if (ii) at any time point t the total supply from the resource is at least the total request of those jobs starting not later than t, i.e., $\sum(\tilde{b}_\ell \mid \ell \in \{1, \ldots, q\} : u_\ell \leq t) \geq \sum(a_j \mid j \in \mathcal{J} : S_j \leq t)$. The objective is to minimize the total weighted completion time $\sum_j w_j C_j$, where $C_j = S_j + p_j$, S_j being the starting time of job J_j, $j \in \mathcal{J}$, in some feasible schedule.

We have assumed that $p_j \geq 1$ for all $j \in \mathcal{J}$, which is a slight restriction, but this helps to keep the presentation simple. In contrast, job j having zero resource requirement, i.e. $a_j = 0$, is possible.

By Assumption 1, there must exist a feasible solution (if every job starts not before u_q, the last supply date) and at least one job must start not before u_q (since all the supplies are positive).

The difficulty of this problem stems from the fact that distinct jobs may have different processing times and resource requirements, and sometimes jobs have to be delayed until a sufficient amount of the resource is supplied in order to start them.

The above problem is a generalization of the single machine scheduling problem with the total weighted job completion time objective, $1||\sum w_j C_j$, the latter being a very well understood and widely studied problem of scheduling theory. In fact, an optimal schedule can here be obtained by Smith's ratio rule [97], i.e., by scheduling the jobs in non-increasing w_j/p_j order. The structure of the polyhedron of feasible job completion times is also known [91]. More on the use of $1||\sum w_j C_j$ in scheduling theory can be found in [7, 13]. As even a single non-renewable resource constraint renders the problem intractable, see below, gaining more insight is a challenging research direction. As a practical application, consider a production line which processes a set of jobs, and the jobs require some raw materials that arrive over time. Finding a good ordering of the jobs with different material requirements, or processing times such that the weighted completion time of the jobs is minimized can be modeled by the scheduling problem described

above.

The $\mathcal{NP}$-hardness proofs

We reduce the single machine scheduling problem with release dates and the average completion time objective ($1|r_j|\sum_j C_j$) to our scheduling problem. In this problem, jobs have release dates specifying their earliest possible start times. We will consider the subset of instances in which exactly one job has a positive release date, and all other jobs have a release date equal to 0. Rinnooy Kan [56] has shown that minimizing the total completion time in this subclass is already $\mathcal{NP}$-hard. We use his result to establish the following theorem:

Proposition 14. *[66]*Minimizing the average completion time on a single machine with resource consuming jobs and one resource with $q = 2$ supply periods is $\mathcal{NP}$-hard ($1|nr = 1, q = 2|\sum_j C_j$).*

Proof. We provide a transformation from the subclass of $1|r_j|\sum_j C_j$ in which there is precisely one job with a positive release date, while all other jobs have release date equal to 0, to our scheduling problem. Take any instance I in this subclass, the corresponding instance I' of our scheduling problem has one machine, one non-renewable resource, and the same set of jobs as I. Suppose J_k is the job with a positive release date in I. Then in I', the second supply date is $u_2 = r_k$, when an amount of one unit is supplied from the single resource, the first supply date is $u_1 = 0$ with an initial stock level 0. In I', all jobs have release date 0, but job J_k has a requirement of 1 unit from the non-renewable resource. Clearly, since the initial supply from the non-renewable resource is 0, J_k cannot start earlier than $u_2 = r_k$ in I'. Hence, the instances I and I' are equivalent in the sense that all jobs except J_k can start at time 0, and job J_k can start at r_k or later. Consequently, I has a solution with objective function value not greater than a given constant D if and only if I' has a solution with objective function value at most D. □

Rinnooy Kan also claims that the 3-PARTITION problem can be reduced to $1|r_j|\sum_j C_j$ by adapting the strong $\mathcal{NP}$-hardness proof of Garey et al. [37] of the two-machine flow-shop scheduling problem with the average completion time objective ($F2||\sum_j C_j$). Recall that an instance of 3-PARTITION consists of $3q$ items each having a non-negative integer size, and a bound B such that each item size is between $B/4$ and $B/2$, and the question is whether the items can be grouped into q 3-element groups of total item size B each. By inspecting the latter proof, one can observe that in the claimed

strong $\mathcal{NP}$-hardness proof of $1|r_j|\sum_j C_j$ there are q jobs with q distinct release dates, and all other jobs have release dates 0. We can easily adapt the proof of Proposition 14 to this case. By this, we obtain the following result.

Theorem 14. *[66]* Minimizing the average completion time on a single machine with resource consuming jobs and one resource is $\mathcal{NP}$-hard in the strong sense ($1|nr = 1|\sum_j C_j$).*

An FPTAS for $1|nr = 1, q = 2|\sum_j w_j C_j$

In this section we describe a fully polynomial time approximation scheme for the special case in which there is a single resource having an initial stock, and one additional supply at date $u_2 > 0$. Recall that an FPTAS for an optimization problem is a class of algorithms, which, for each $0 < \varepsilon < 1$, has an algorithm A_ε which runs in polynomial time in the size of the input and in $1/\varepsilon$, and produces a feasible solution of cost at most $(1+\varepsilon)$ times the cost of an optimal solution in case of minimization problems, and at least $(1-\varepsilon)$ times the cost of an optimal solution in case of maximization problems [95]. Our algorithm has a very simple structure and uses standard techniques, see [95]. The only interesting part is the transformation of the scheduling problem $1|nr = 1, q = 2|\sum_j w_j C_j$ to finding a shortest path in an appropriately defined graph, but the twist is that the shortest path computation is needed for finding a resource feasible solution, instead of computing the cost of the solution.

In order to get more insight into our scheduling problem, consider any instance in the subclass studied in this section, and any feasible solution to that instance. Since there are only two supply dates, u_1 and u_2, with $u_1 = 0$ and corresponding supply $\tilde{b}_1$, and $u_2 > 0$ with corresponding supply $\tilde{b}_2$, we know that the set of jobs is partitioned into $\mathcal{J}^1$ and $\mathcal{J}^2$, where the jobs in $\mathcal{J}^1$ use only the initial supply available at u_1, while the remaining jobs use that arriving at u_2 and the supply from $\tilde{b}_1$ left by the jobs in $\mathcal{J}^1$. Moreover, the jobs in $\mathcal{J}^2$ are scheduled after all the jobs in $\mathcal{J}^1$. Now we prove that there is an optimal solution with a special structure, and this will serve as a basis for our FPTAS.

Proposition 15. *[66]* The problem $1|nr = 1, q = 2|\sum_j w_j C_j$ always admits an optimal solution of the following structure:*

i) The set of jobs is partitioned into subsets $\mathcal{J}^1$ and $\mathcal{J}^2$, the jobs in $\mathcal{J}^1$ use only the initial supply and are scheduled consecutively from time u_1 on, while the jobs in $\mathcal{J}^2$ use the supply left from $\tilde{b}_1$ and also the supply $\tilde{b}_2$, and are scheduled consecutively from the time point $\max\{u_2, \sum_{j\in\mathcal{J}^1} p_j\}$.

ii) The jobs in both of $\mathcal{J}^1$ and $\mathcal{J}^2$ are scheduled in non-increasing w_j/p_j order, and there is at most one job in $\mathcal{J}^1$ which finishes after u_2, and if such a job exists, it starts before u_2.

Proof. Take any optimal solution, and let $\mathcal{J}^1$ consist of those jobs starting before u_2, and $\mathcal{J}^2$ contain the remaining jobs. Then, part i) holds, as one may easily verify. Moreover, there is at most one job in $\mathcal{J}^1$ which finishes after u_2, and this job must start before u_2 by definition. Further on, the jobs in $\mathcal{J}^2$ must be scheduled in non-increasing w_j/p_j order, otherwise the schedule is not optimal as a simple job-exchange argument shows (notice that after u_2, the resource is completely supplied, and there is nothing which could prevent such an ordering of the jobs in $\mathcal{J}^2$). Finally, suppose that the jobs in $\mathcal{J}^1$ are not scheduled in non-increasing w_j/p_j order, then again, a simple exchange argument shows that the schedule is not optimal, a contradiction, which proves part ii). □

In light of Proposition 15, we can express the objective function value of a schedule with a given partitioning $\mathcal{J}^1$ and $\mathcal{J}^2$ of the jobs (assuming that the jobs are indexed such that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$), as follows:

$$\sum_{j\in\mathcal{J}^1} w_j \left(\sum_{k\in\mathcal{J}^1, k\leq j} p_k \right) + \sum_{j\in\mathcal{J}^2} w_j \left(\max\left\{ u_2, \sum_{k\in\mathcal{J}^1} p_k \right\} + \sum_{k\in\mathcal{J}^2, k\leq j} p_k \right). \quad (3.30)$$

Notice that the first half of this expression calculates the weighted completion time of the jobs in $\mathcal{J}^1$ when scheduled in non-increasing w_j/p_j order, and the second half does the same for the jobs in $\mathcal{J}^2$.

In the following, denote, for any subset H of jobs, $p(H) := \sum_{j\in H} p_j$, and $w(H) := \sum_{j\in H} w_j$.

In the approximation algorithm for some $0 < \varepsilon < 1$, we will *round* some partial sums of the problem data, and *guess* the value of $\max\{u_2, p(\mathcal{J}^1)\}$. We define a *rounding function* $f(s)$ which assigns to any nonnegative number $s \geq 0$ a number from a discrete set as follows. Let $\Delta := 1 + \varepsilon/4n$. Then, we define

$$f(s) := \begin{cases} 0, & s = 0 \\ \Delta^{\lceil \log_\Delta s \rceil}, & s > 0 \end{cases},$$

where $\lceil x \rceil$ is the smallest integer $z \geq x$ for any $x \in \mathbb{R}$. Notice that $s \leq f(s) \leq s \cdot \Delta$ holds for any $s \geq 0$, since $\Delta > 1$. We will round iteratively sums of numbers, i.e., the iterative rounding of a sum of the form $x_1 + x_2 + \cdots + x_k$ is defined with the formula $g_t := f(x_t + g_{t-1})$ for $t = 1, \ldots, k$, where $g_0 := 0$.

Proposition 16. *[66]* If $x_t \geq 0$ for all $t = 1, \ldots, k$, then*

i) $x_1 + \cdots + x_t \leq g_t$, *for* $t = 1, \ldots, k$.

ii) $g_t \leq \Delta^t(x_1 + \cdots + x_t)$, *for* $t = 1, \ldots, k$.

Proof. For showing i), it is enough to note that the rounding function $f(\cdot)$ rounds up any value $s \geq 1$ to the least value $\Delta^k \geq s$, with $k \in \mathbb{Z}_{\geq 0}$.

The set of inequalities ii) is verified by induction on t. For $t = 1$, we have $g_1 = f(x_1) \leq \Delta x_1$, by using the definition of $f(\cdot)$. So assume that the inequality is proved for $t-1$, and we verify it for t: $g_t = f(x_t + g_{t-1}) \leq (x_t + g_{t-1})\Delta \leq (x_t + \Delta^{t-1}(\sum_{i=1}^{t-1} x_i))\Delta \leq \Delta^t(x_1 + \cdots + x_t)$, where the first inequality follows from the definition of $f(\cdot)$, the second from the induction hypothesis, and the third from $\Delta > 1$. □

This rounding scheme has been used e.g., in [95] (Section 0.5.1: Makespan on two identical machines).

We will guess the value of $\max\{u_2, p(\mathcal{J}^1)\}$ by picking a member from the set $G := \{u_2\Delta^z \ : \ z = 0, \ldots, \lceil \log_\Delta((u_2 + p(\mathcal{J}))/u_2)\rceil\}$. Notice that the largest value in G is bounded by $(u_2 + p(\mathcal{J}))\Delta$.

With this data, we build a directed graph D_δ for each $\delta \in G$. Firstly, we re-index the jobs such that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$. The nodes of D_δ represent all the distinct partitioning of the first t jobs, for all $t \in \{1, \ldots, n\}$, noting that the same node for a given t may represent several distinct partitionings of the first t jobs. We associate a vector with each node, that is, a node with the first t jobs has a vector $(t, P_1', PW_1', P_2', PW_2')$, where P_1' is the iterative rounding of the total processing times of those jobs assigned to the time period $[u_1, \delta]$ out of the first t jobs, and PW_1' is the iterative rounding of the total weighted completion times of these jobs. Likewise, P_2', and PW_2' are the iteratively rounded total processing times, and the iteratively rounded total weighted completion times, respectively, of those jobs assigned to the time period $[\delta, \infty)$, i.e., these jobs start at time δ. The unique source node (with zero in-degree) represents the empty schedule, when no job is chosen ($t = 0$). Then we gradually assign the jobs to time periods, and add more nodes to the graph, until all the jobs are assigned. Consider a node with vector $(t, P_1', PW_1', P_2', PW_2')$. It has two successor nodes, giving rise to two directed arcs. One of the successors corresponds to assigning job J_{t+1} to the time period $[u_1, \delta]$, and the other successor to assigning J_{t+1} to the time period $[\delta, \infty)$. The vectors associated with these two nodes can be easily computed from $(t, P_1', PW_1', P_2', PW_2')$: if J_{t+1} is assigned to the time period $[u_1, \delta]$, then the associated vector is $(t+1, f(P_1' + p_{t+1}), f(PW_1' + w_{t+1}(P_1' + p_{t+1})), P_2', PW_2')$, and the vector of the other node can be computed as $(t+1, P_1', PW_1', f(P_2' + p_{t+1}), f(PW_2' +$

$w_{t+1}(\delta + P'_2 + p_{t+1})))$. Clearly, the same node can be the successor of several other nodes. The arcs representing the assignment of job J_{t+1} to the time period $[u_1, \delta]$ have weight a_{t+1}, while all other arcs have weight 0, for all $t = 0, \ldots, n-1$. The nodes that contain a partitioning of all the n jobs are called *terminal nodes*. A terminal node with vector $(n, P'_1, PW'_1, P'_2, PW'_2)$ is *feasible* if it satisfies both of the following conditions: (i) $P'_1 \leq \delta$, and (ii) the shortest path from the source node to this node has total arc length at most $\tilde{b}_1$. We say that D_δ *gives rise to a feasible solution* if and only if D_δ admits a feasible terminal node. The *value* of a feasible terminal node with vector $(n, P'_1, PW'_1, P'_2, PW'_2)$ is $PW'_1 + PW'_2$. After these preliminaries, the complete FPTAS is as follows.

1. Determine the set G.

2. For each $\delta \in G$, compute the graph D_δ, and determine if D_δ gives rise to a feasible solution.

3. Choose D_{δ^*} which gives rise to a feasible solution with the smallest value overall.

4. Recover a schedule by following a shortest path in D_{δ^*} from the unique source node to a feasible terminal node of smallest rounded objective function value, and forming the sets $\mathcal{J}^1_*$ and $\mathcal{J}^2_*$ such that a job J_j is put in $\mathcal{J}^1_*$ if and only if J_j is assigned to time period $[u_1, \delta^*]$, and put into $\mathcal{J}^2_*$ otherwise.

Theorem 15. *[66]* The algorithm is an FPTAS for* $1|nr = 1, q = 2|\sum_j w_j C_j$.

Proof. Take any instance I of $1|nr = 1, q = 2|\sum_j w_j C_j$, and consider an optimal solution with the structure specified in Proposition 15. In particular, let $\mathcal{J}^1$ be the set of jobs starting before u_2. Let δ be the largest member of G with $\delta < \Delta^{n+1} \max\{u_2, p(\mathcal{J}^1)\}$ (since $u_2 \in G$, such a member exists). Since $\Delta^n \leq 1 + \varepsilon/2$ (using the fact that $(1 + x/n)^n \leq 1 + 2x$ for $0 \leq x \leq 1$, see [95]), we derive $\delta < \Delta^{n+1} \max\{u_2, p(\mathcal{J}^1)\} \leq (1 + \varepsilon) \max\{u_2, p(\mathcal{J}^1)\}$. Notice that since $\delta = u_2 \Delta^k$ for some $k \in \mathbb{Z}_{\geq 0}$, $\delta \geq \Delta^n p(\mathcal{J}^1)$ follows.

In D_δ consider a terminal node with associated vector $(n, P'_1, PW'_1, P'_2, PW'_2)$ which corresponds to the partitioning of jobs $\mathcal{J}^1, \mathcal{J}^2$ in the optimal solution. Then this must be a feasible terminal node of D_δ, since we started out from a feasible schedule, and $\delta \geq \Delta^n p(\mathcal{J}^1) \geq P'_1 \geq p(\mathcal{J}^1)$, where the first inequality is from the preceding paragraph, while the second and third follow from Proposition 16. The value of this node is $PW'_1 + PW'_2$. How much bigger is

$PW'_1 + PW'_2$ than the objective function value (3.30)? We compute

$$PW'_1 + PW'_2 < \Delta^n \left(\sum_{j \in \mathcal{J}^1} w_j \left(\sum_{k \in \mathcal{J}^1, k \leq j} p_k \right) + \sum_{j \in \mathcal{J}^2} w_j \left(\delta + \sum_{k \in \mathcal{J}^2, k \leq j} p_k \right) \right) <$$

$$\Delta^n \left(\sum_{j \in \mathcal{J}^1} w_j \left(\sum_{k \in \mathcal{J}^1, k \leq j} p_k \right) + \sum_{j \in \mathcal{J}^2} w_j \left((1+\varepsilon) \max\{u_2, p(\mathcal{J}^1)\} + \sum_{k \in \mathcal{J}^2, k \leq j} p_k \right) \right)$$

$$\leq \Delta^n (1+\varepsilon) \left(\sum_{j \in \mathcal{J}^1} w_j \left(\sum_{k \in \mathcal{J}^1, k \leq j} p_k \right) + \sum_{j \in \mathcal{J}^2} w_j \left(\max\left\{u_2, p(\mathcal{J}^1)\right\} + \sum_{k \in \mathcal{J}^2, k \leq j} p_k \right) \right)$$

$$\leq (1+2\varepsilon) \left(\sum_{j \in \mathcal{J}^1} w_j \left(\sum_{k \in \mathcal{J}^1, k \leq j} p_k \right) + \sum_{j \in \mathcal{J}^2} w_j \left(\max\left\{u_2, p(\mathcal{J}^1)\right\} + \sum_{k \in \mathcal{J}^2, k \leq j} p_k \right) \right),$$

where the first inequality follows from the properties of iterative rounding (Proposition 16), the second from the inequality $\delta < (1+\varepsilon)\max\{u_2, p(\mathcal{J}^1)\}$ shown in the first paragraph of this proof, and the rest from the inequalities $\Delta^n \leq 1 + \varepsilon/2$ and $(1+\varepsilon)(1+\varepsilon/2) \leq 1 + 2\varepsilon$ for $0 \leq \varepsilon \leq 1$. Since the algorithm chooses a feasible terminal node with smallest rounded value, we know that the value of the best solution found is also at most $(1+2\varepsilon)$ times the optimum.

The time complexity of the algorithm is determined by the cardinality of G, and the size of the graphs D_δ. The former set has $\lceil \log_\Delta((p(\mathcal{J}) + u_2)/u_2) \rceil = \lceil \ln((p(\mathcal{J}) + u_2)/u_2) / \ln \Delta \rceil$ elements, which is bounded by $CG := \lceil (\ln((p(\mathcal{J}) + u_2)/u_2))(1+4n)/\varepsilon \rceil$, a polynomial in the size of the input and in $1/\varepsilon$, since $\ln(1+\varepsilon/4n) \geq \varepsilon/(\varepsilon+4n) \geq \varepsilon/(1+4n)$, and $\ln z \geq (z-1)/z$, see [95]. On the other hand, the number of nodes of D_δ is bounded by the number of distinct $(t, P'_1, PW'_1, P'_2, PW'_2)$ vectors. Since both of P'_1 and P'_2 take values from the set $\{0\} \cup \{\Delta^z : z \in \{0, \ldots, n + \lceil \log_\Delta p(\mathcal{J}) \rceil\}\}$, and both of PW'_1 and PW'_2 take values from the set $\{0\} \cup \{\Delta^z : z \in \{0, \ldots, n + \lceil \log_\Delta w(\mathcal{J})(u_2 + p(\mathcal{J})) \rceil\}\}$, we deduce that both of P'_1 and P'_2 may take at most $CP := n + \lceil \ln p(\mathcal{J}) \rceil (1+4n)/\varepsilon$ distinct values, and that both of PW'_1 and PW'_2 may take at most $CPW := n + \lceil \ln w(\mathcal{J})(u_2 + p(\mathcal{J})) \rceil (1+4n)/\varepsilon$ distinct values. Therefore, the number of nodes is bounded by a polynomial in the size of the input, and in $1/\varepsilon$. Since the out-degree of each non-terminal node is 2, we know that the size of D_δ is also polynomial in the size of the input and in $1/\varepsilon$, and thus the shortest path to all the terminal nodes can be computed in polynomial time in the size of the input (in linear time in the size of the graph, since D_δ is acyclic), and in $1/\varepsilon$. Therefore, we have an FPTAS with running time $O(CG \cdot n \cdot CP^2 \cdot CPW^2)$. □

3.2 Parallel machine problems

In this section we discuss scheduling problems with parallel machines and jobs consuming non-renewable resources. Again, our focus is on the approximability of various special cases.

We have m parallel machines, $\mathcal{M} = \{M_1, \ldots, M_m\}$, a finite set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, and a finite set of non-renewable resources $\mathcal{R}$ consumed by the jobs. Each job J_j has a processing time $p_j \in \mathbb{Z}_{\geq 1}$, a release date r_j, and resource requirements $a_{ij} \in \mathbb{Z}_{\geq 1}$ from the resources $i \in \mathcal{R}$. Preemption of jobs is not allowed and each machine can process at most one job at a time. The resources are supplied in q different time moments, $0 = u_1 < u_2 < \ldots < u_q$; the vector $\tilde{b}_\ell \in \mathbb{Z}_{\geq 1}^{|\mathcal{R}|}$ represents the quantities supplied at u_ℓ. A *schedule* σ specifies a machine and the starting time S_j of each job and it is *feasible* if (i) on every machine the jobs do not overlap in time, (ii) $S_j \geq r_j$ for each $j \in \mathcal{J}$, and if (iii) at any time point t the total supply from each resource is at least the total request of those jobs starting not later than t, i.e., $\sum_{(\ell \,:\, u_\ell \leq t)} \tilde{b}_{\ell i} \geq \sum_{(j \,:\, S_j \leq t)} a_{ij}, \ \forall i \in \mathcal{R}$. We will consider two types of objective functions: the minimization of the maximum job completion time (makespan) defined by $C_{\max} = \max_{j \in J} C_j$; and the minimization of the maximum lateness, i.e., each job has a due-date d_j, $j \in \mathcal{J}$, and $L_{\max} := \max_{j \in \mathcal{J}}(C_j - d_j)$. Clearly, $L_{\max}$ is a generalization of $C_{\max}$.

Parallel machine scheduling with the $C_{\max}$ and $L_{\max}$ objectives, respectively, (without non-renewable resources), is a well-studied problems in the literature. A straightforward transformation from the 3-PARTITION problem shows that minimizing the makespan (and thus the maximum lateness) is NP-hard in the strong sense, and a similarly straightforward reduction from the PARTITION problem proves that these problems remain $\mathcal{NP}$-hard in the ordinary sense even if the number of machines is 2, see e.g., Garey and Johnson [36]. The most important result, from our point of view, is that the parallel machine scheduling problem with the maximum lateness objective and with job release dates, $P|r_j|L_{\max}$ (and thus also the special case $P||C_{\max}$), admits a PTAS [46]. In fact, we will reuse this PTAS in our polynomial time approximation scheme for $Pm|nr = const, q = const|C_{\max}$ and, when there is a single resource and the processing times are proportional to the resource requirements (or vice versa) $Pm|nr = 1, q = const, p_j = a_j|L_{\max}$.

In Section 3.2.1 we describe a negative result showing that the makespan minimization problem with arbitrary number of machines and with 2 non-renewable resources only does not admit a PTAS unless $\mathcal{P} = \mathcal{NP}$. This is the motivation for considering a fixed number of machines subsequently. In Section 3.2.2 we describe a PTAS for $Pm|nr = 1, p_j = a_j|L_{\max}$, i.e., the

number of machines is a constant, there is only one non-renewable resource, the number of the supply dates is a constant, and the processing time of each job equals its resource requirement.

3.2.1 Inapproximability results

In this section we prove that if the number of machines is part of the input, then $P|nr = 2, q = 2, p_j = 1|C_{\max}$ cannot be approximated better than $3/2 - \varepsilon$ for any $\varepsilon > 0$ unless $\mathcal{P} = \mathcal{NP}$.

Theorem 16. *[45]* Deciding whether there is a schedule of makespan 2 with two non-renewable resources, two supply dates and unit-time jobs on an arbitrary number of machines ($P|nr = 2, q = 2, p_j = 1|C_{\max} \leq 2$) is $\mathcal{NP}$-hard.*

Proof. We reduce the EVEN-PARTITION problem to the problem $P|nr = 2, q = 2, p_j = 1|C_{\max}$, and argue that deciding whether a schedule of makespan two exists is as hard as finding a solution for EVEN-PARTITION. Recall that an instance of the EVEN-PARTITION problem consists of $2t$ items, for some integer t, of sizes $a_1, \ldots, a_{2t} \in \mathbb{Z}_{\geq 1}$. The decision problem asks whether the set of items can be partitioned into two subsets S and $\bar{S}$ of cardinality t each, such that $\sum_{i \in S} a_i = \sum_{i \in \bar{S}} a_i$? This problem is $\mathcal{NP}$-hard in the ordinary sense, see [36]. Clearly, a necessary condition for the existence of set S is that the total size of all items is an even integer, i.e., $\sum_{i=1}^{2t} a_i = 2A$, for some $A \in \mathbb{Z}_{\geq 1}$.

We map an instance I of EVEN-PARTITION to the following instance of $P|nr = 2, q = 2, p_j = 1|C_{\max}$. There are $n := 2t$ jobs, and $m := t$ machines. All the jobs have unit processing time, i.e., $p_j = 1$ for all j. The job corresponding to the jth item in I has resource requirements $a_{1,j} := a_j$ and $a_{2,j} := A - a_j$. The initial supply at $u_1 = 0$ from the two resources is $\tilde{b}_{1,1} := A$ and $\tilde{b}_{1,2} := (t-1)A$, and the second supply at time $u_2 = 1$ has $\tilde{b}_{2,1} := A$, and $\tilde{b}_{2,2} := (t-1)A$. We have to decide whether a feasible schedule of makespan two exists.

First, suppose that I has a solution S. Then we schedule all the jobs corresponding to the items in S at time 0, each on a separate machine. Since S contains t items, and the number of machines is t as well, this is feasible. Moreover, the total resource requirement from the first resource is precisely A, whereas that from the second one is $\sum_{j \in S} a_{2,j} = \sum_{j \in S}(A - a_j) = (t-1)A$. The rest of the jobs are scheduled at time 1. Since their number is t, and since $u_2 = 1$ is the second and last supply date, all the resources are supplied and the jobs can start promptly at time 1.

Conversely, suppose there is a feasible schedule of makespan two. Then, there are t jobs scheduled at time 0, and the remaining t jobs at time 1. Let S denote the set of the jobs scheduled at time 0. The resource requirements of those jobs in S equal the supply at time $u_1 = 0$, because $\sum_{j\in S} a_j = A$ follows from the resource constraints: on the one hand $\sum_{j\in S} a_j = \sum_{j\in S} a_{1,j} \leq A$, and on the other hand $\sum_{j\in S} a_{2,j} = \sum_{j\in S}(A - a_j) = tA - \sum_{j\in S} a_j \leq (t-1)A$, thus $A \leq \sum_{j\in S} a_j$. Hence S is a feasible solution of the EVEN-PARTITION problem instance. □ □

Corollary 13. *[45]* It is $\mathcal{NP}$-hard to approximate problem $P|nr = 2, q = 2, p_j = 1|C_{\max} \leq 2$ better than $3/2 - \varepsilon$ for any $\varepsilon > 0$.*

By Assumption 1, the optimum makespan is at least u_q, therefore, a straightforward three-approximation algorithm would apply list scheduling (list scheduling has an approximation guarantee of $2 - 1/m$ for $P||C_{\max}$) to schedule all the jobs after u_q. Therefore, we have the following result.

Corollary 14. *[45]* $P|nr = 2, q = 2, p_j = 1|C_{\max}$ is APX-complete.*

3.2.2 PTAS for $Pm|nr = 1, p_j = a_j|L_{\max}$

The lateness objective is special in the sense that it can take 0 on negative values in an optimal solution. Therefore, as usual, throughout this section we consider a shifted lateness objective $L'_{\max} := D + \max_{j\in J} L_j$, where $D = u_q + \max_{j\in\mathcal{J}} d_j$ is the offset. By definition, $L'_{\max} \geq u_q$ for any feasible schedule.

Throughout this section we assume that $\varepsilon > 0$ is a small constant with $1/\varepsilon \in \mathbb{Z}$. At the expense of increasing the optimum value by at most εu_q, we may assume that every supply date is an integer multiple of εu_q. This can be achieved by increasing each supply date by at most εu_q. More precisely, we define the supply dates $u'_\ell = \ell\varepsilon u_q$ for $\ell = 1, \ldots, 1/\varepsilon$. The total supply of the resource at u'_ℓ is $\sum_{k:u'_{\ell-1}<u_k\leq u'_\ell} b_\ell$. Then, any schedule S feasible with respect to the original supplies can be turned into a schedule feasible with respect to the shifted supply dates and aggregated supply quantities, all we have to do is to increase each S_j by εu_q. Clearly, this increases the lateness of the schedule by εu_q.

Let $\mathcal{S}' := \{j \in \mathcal{J} | p_j \leq \varepsilon^2 u_q\}$ be the set of *tiny* jobs, and $\mathcal{B}' := \mathcal{J} \setminus \mathcal{S}'$ be the set of *huge* jobs. Note that between two consecutive supply dates at most $1/\varepsilon$ huge jobs can start, thus we can assume $\sum_{j\in\mathcal{B}'} x_{j\ell k} \leq 1/\varepsilon$, if $\ell < q$ and $k \in \mathcal{M}$, therefore there are at most $(n+1)^{(1/\varepsilon)qm}$ different assignments of huge jobs to the supply dates u_1 through u_{q-1}. We can examine all of them, since m and ε are constants. The remaining huge jobs are assigned to u_q, but we assign them to machines later. For each huge job assignment we will

guess approximately the total processing time of those tiny jobs that start in the interval $[u_\ell, u_{\ell+1})$ on machine M_k, $\ell = 1, \ldots, q-1$, and $k = 1, \ldots, m$. A guess is a number of the form $g_{k,\ell} \cdot (\varepsilon^2 u_q)$, where $0 \leq g_{k,\ell} \leq 1/\varepsilon + 1$ is an integer. A guess for all the $q-1$ supply dates and all the m machines can be represented by a $m \times (q-1)$-tuple $g = (g_{k,\ell})$, and let G denote the set of all possible guesses. The algorithm is as follows:

Algorithm B [45]
Initialization: S^{best} is a schedule where each job is scheduled on M_1 after u_q.

1. For each feasible partial assignment $\hat{x}^{huge,b}$ of huge jobs to machines and supply dates u_1 through u_{q-1}, perform the following steps.

2. For each tuple $g \in G$, do steps 3 - 6:

3. We create a feasible partial assignment $\hat{x}^b$ by assigning also the tiny jobs to machines and supply dates u_1 through u_{q-1}. Initially $\hat{x}^b$ is the same as $\hat{x}^{huge,b}$. Let L be the list of tiny jobs sorted in non-decreasing d'_j order. Jobs from L are assigned to machines and to supply dates u_1 through u_{q-1} until all jobs from L get assigned or all the supply dates from u_1 through u_{q-1} are processed. When processing supply date u_ℓ, $\ell \in \{1, \ldots, q-1\}$, we first assign jobs to M_1, then to M_2, etc. Let M_k be the next machine to receive some jobs. Let $h_{k,\ell}$ be the smallest number of tiny jobs from the beginning of L with a total processing time of at least $g_{k,\ell}(\varepsilon^2 u_q)$, and let $z_{k,\ell}$ be the maximum number of tiny jobs from the beginning of L that can be assigned to u_ℓ without violating the resource constraint. Assign $\min\{h_{k,\ell}, z_{k,\ell}\}$ jobs from the beginning of L to supply date u_ℓ on M_k, and remove them from L. Then proceed with the next machine until all machines are processed or L becomes empty.

4. Create a partial schedule S^{part} from $\hat{x}^b$ with Subroutine Sch.

5. Let $C^{part}_{\max}(k)$ be the time when M_k finishes S^{part}. Invoke the PTAS of [45] for the problem $P|preassign, r_j|L_{\max}$ with $\max\{C^{part}_{\max}(k), u_q\}$ amount of pre-assigned work on M_k $(k = 1, 2, \ldots, m)$ to schedule the remaining jobs. Let S^{act} be the resulting schedule.

6. If $L'_{\max}(S^{act}) < L'_{\max}(S^{best})$, then let $S^{best} := S^{act}$.

7. After examining each feasible assignment of huge jobs before u_q, output S^{best}.

Notice that in Step 5 we use the PTAS of Györgyi and Kis [45] (which is an easy extension of the PTAS of Hall and Shmoys for $P|r_j|L_{\max}$) for solving the problem $P|preassign, r_j|L_{\max}$ which is like $P|r_j|L_{\max}$, but on each machine there can be some pre-assigned work which must precede any newly scheduled jobs.

Subroutine Sch Input: $\bar{\mathcal{J}} \subseteq \mathcal{J}$ and $\bar{x}$ such that for each $j \in \bar{\mathcal{J}}$ there exists a unique (ℓ, k) with $\bar{x}_{j\ell k} = 1$.

Output: partial schedule S^{part} of the jobs in $\bar{\mathcal{J}}$.

1. S^{part} is initially empty, then we schedule the jobs on each machine in increasing u_ℓ order (first we schedule those jobs assigned to u_1, and then those assigned to u_2, etc.):

2. For each machine M_k, and supply date u_ℓ, always schedule first the tiny jobs and then the huge jobs. When scheduling the next job with $\bar{x}_{j\ell k} = 1$, then it is scheduled at time $\max\{u_\ell, C_{last}(k)\}$, where $C_{last}(k)$ is the completion time of the last job scheduled on machine M_k, or 0 if no job has been scheduled yet on M_k.

The final schedule S^{best} of Algorithm B is obviously feasible and the running time is polynomial in the size of the input, since the number of possible huge job assignments before u_q can be bounded by $O((n+1)^{(1/\varepsilon)qm})$, the number of the tuples is $(1/\varepsilon+2)^{m(q-1)}$, steps 3 and 4 require $O(n \log n)$ time, while step 5 also requires polynomial time, see [45].

For the sake of proving that Algorithm B is a PTAS, we construct an intermediate schedule $\tilde{S}$ which, on the one hand, has a similar structure to that of an optimal schedule, and on the other hand, not far from the schedule computed by Algorithm B. $\tilde{S}$ is derived from an optimal schedule S^* as follows. Let $g^*_{k,\ell}$ ($k \in \{1,\dots,m\}$ and $\ell \in \{1,\dots,q-1\}$) be the smallest integer such that $(g^*_{k,\ell}-1)\cdot(\varepsilon^2 u_q)$ is at least the total processing time of the tiny jobs starting in $[u_\ell, u_{\ell+1})$ on M_k in S^* unless there is no such tiny job, in which case $g^*_{k,\ell} = 0$. First perform Steps 3 and 4 of Algorithm B with the partial huge job assignment $(x^{huge,b})^*$ that corresponds to S^*, and the tuple g^* just defined. After that, schedule the remaining huge jobs at $\tilde{S}_j := S^*_j + 5\varepsilon u_q$ on the same machine as in S^* and finally schedule the remaining tiny jobs in earliest-due-date (EDD) order after $\max\{C^{part}_{\max}, u_q\}$ at the earliest idle time on any machine.

In order to compare $\tilde{S}$ with S^{best} (Proposition 17), and with S^* (Proposition 18), first we make two observations. Let $\tilde{\mathcal{J}}_{\ell,k}$ denote the set of tiny jobs that are assigned to u_ℓ and M_k in $\tilde{S}$ and $\mathcal{J}^*_{\ell,k}$ denote the set of tiny jobs with $u_\ell \le S^*_j < u_{\ell+1}$ on machine k. $\tilde{\mathcal{J}}_\ell := \cup_k \tilde{\mathcal{J}}_{\ell,k}$ and $\mathcal{J}^*_\ell := \cup_k \mathcal{J}^*_{\ell,k}$. Let $\mathcal{M}^*_\ell$ denote the set of those machines with at least one tiny job that starts in $[u_\ell, u_{\ell+1})$ in S^*.

Observation 2. *[45] For each $\ell < q$ and $M_k \in \mathcal{M}$, $p(\tilde{\mathcal{J}}_{\ell,k}) < p(\mathcal{J}^*_{\ell,k}) + 3\varepsilon^2 u_q$ and $p(\cup_{\nu\le\ell}\tilde{\mathcal{J}}_\nu) \ge p(\cup_{\nu\le\ell}\mathcal{J}^*_\nu) - \varepsilon^2 u_q$.*

Proof. The first part follows from $p(\mathcal{J}^*_{\ell,k}) + 3\varepsilon^2 u_q > (g^*_{k,\ell}-2)(\varepsilon^2 u_q) + 3\varepsilon^2 u_q = (g^*_{k,\ell}+1)(\varepsilon^2 u_q) > p(\tilde{\mathcal{J}}_{\ell,k})$ (the first inequality follows from the choice of g^*,

while the second from the construction of $\tilde{S}$). For the second part, let $\ell' \leq \ell$ denote the last period where the algorithm had to proceed with the next period, because there was not enough resource to schedule the next tiny job, but $\mathcal{M}^*_{\ell'} \neq \emptyset$. The huge jobs that are assigned to a time period until $u_{\ell'}$ in $\tilde{S}$ are scheduled before $u_{\ell'+1}$ in S^*, thus, since $p_j = a_j$ and S^* is feasible, $p(\cup_{\nu \leq \ell'} \tilde{\mathcal{J}}_\nu) \geq p(\cup_{\nu \leq \ell'} \mathcal{J}^*_\nu) - \varepsilon^2 u_q$ follows, because otherwise there would be enough resource to assign at least one more tiny job to $u_{\ell'}$ in $\tilde{S}$. According to the definition of ℓ' and the rules of Algorithm B, we have $p(\tilde{\mathcal{J}}_\nu) \geq p(\mathcal{J}^*_\nu)$ for each $\nu = \ell' + 1, \ldots, \ell$, thus the observation follows. □

Observation 3. *[45] After processing supply date u_ℓ in Step 3 of Algorithm B, then at least one of the following conditions holds: (i) there is not enough resource to assign the next tiny job, (ii) $p(\cup_{\nu \leq \ell} \tilde{\mathcal{J}}_\nu) \geq p(\cup_{\nu \leq \ell} \mathcal{J}^*_\nu)$ or (iii) $\mathcal{M}^*_\ell = \emptyset$.*

Proof. If (i) and (iii) are not true, then we have $p(\mathcal{J}^*_\ell) \leq \sum_{k \in \mathcal{M}^*_\ell} (g^*_{k,\ell} - 1) \cdot (\varepsilon^2 u_q) \leq p(\tilde{\mathcal{J}}_\ell) - \varepsilon^2 u_q$, where the first inequality follows from the definition of g^*, the second from the rule of Algorithm B (step 3). Consequently, the observation follows from the second part of Observation 2 (using it for $\ell - 1$). □

Proposition 17. *[45] $\tilde{S}$ is feasible, and $L'_{max}(S^{best}) \leq (1 + \varepsilon) L'_{\max}(\tilde{S})$.*

Proof. $\tilde{S}$ cannot violate the resource constraints by the rules of Algorithm B, and due to Observation 2, the jobs scheduled on an arbitrary machine M_k must end before a huge job scheduled in the last stage of the construction of $\tilde{S}$ would start, since for all those huge jobs, $\tilde{S}_j = S^*_j + 5\varepsilon u_q$ by definition. In some iteration, Algorithm B will consider the huge job assignment and the tuple that we used to define $\tilde{S}$. Hence, after step 4, $\tilde{S}$ and S^{part} coincide. Therefore, the Proposition follows from [46]. □

Proposition 18. *[45] $L'_{\max}(\tilde{S}) \leq L'_{\max}(S^*) + 6\varepsilon u_q$.*

Proof. Let j be such that $L'_j(\tilde{S}) = L'_{\max}(\tilde{S})$. First suppose that j is huge. If j is scheduled at step 4 (since it is assigned to a supply date u_ℓ and a machine M_k), then the jobs assigned to M_k and to a $u_{\ell'}$ with $\ell' < \ell$, are completed at most $3(\ell - 1)\varepsilon^2 u_q$ later in $\tilde{S}$ than the jobs with $S^*_{j'} < u_\ell$ on M_k in S^* (Observation 2). The total processing time of the jobs that are assigned to u_ℓ and M_k and scheduled before j in $\tilde{S}$ is at most $\varepsilon u_q + 3\varepsilon^2 u_q$, thus $\tilde{C}_j \leq C^*_j + 5\varepsilon u_q$ follows. If it is scheduled at step 5, then originally we have $\tilde{S}_j = S^*_j + 5\varepsilon u_q$ and we may push j to the right by at most $\varepsilon^2 u_q$, thus $\tilde{C}_j \leq C^*_j + 6\varepsilon u_q$.

Now suppose that j is tiny.

Claim 3. $\min\{d_{j'} : j' \in \cup_{\nu\geq\ell}\tilde{\mathcal{J}}_\nu\} \geq \min\{d_{j'} : j' \in \cup_{\nu\geq\ell}\mathcal{J}^*_\nu\}$, *for each* $\ell \leq q$.

Proof. Assume for a contradiction that there exists an $\ell \leq q$ and $j_1 \in \tilde{\mathcal{J}}_\ell$ such that

$$d_{j_1} = \min\{d_{j'} : j' \in \tilde{\mathcal{J}}_\ell\} = \min\{d_{j'} : j' \in \cup_{\nu\geq\ell}\tilde{\mathcal{J}}_\nu\} < \min\{d_{j'} : j' \in \cup_{\nu\geq\ell}\mathcal{J}^*_\nu\}, \tag{3.31}$$

where the second equation follows from the EDD scheduling of tiny jobs in $\tilde{S}$. Let $H := \{j' \in \mathcal{S}' : d_{j'} \leq d_{j_1}\}$. Let $\ell' < \ell$ be the largest index such that $\mathcal{M}^*_{\ell'} \neq \emptyset$. If there is no such ℓ', then the claim follows, since we have $\bigcup_{\nu<\ell}\tilde{\mathcal{J}}_\nu = \bigcup_{\nu<\ell}\mathcal{J}^*_\nu = \emptyset$ from the definition of $\tilde{S}$. Otherwise, for each $\nu = \ell'+1, \ldots, \ell-1$, since $\mathcal{M}^*_\nu = \emptyset$, we have $\mathcal{J}^*_\nu = \tilde{\mathcal{J}}_\nu = \emptyset$. Furthermore, from (3.31), it follows that all the jobs in H start before $u_{\ell'+1}$ in S^* by our indirect assumption. Therefore,

$$p(\cup_{\nu\leq\ell'}\tilde{\mathcal{J}}_\nu) < p(H) \leq p(\cup_{\nu\leq\ell'}\mathcal{J}^*_\nu),$$

where the first inequality follows from the fact that H comprises all the tiny jobs assigned to any time period $u_\nu < u_\ell$ in $\tilde{S}$, and j_1 as well, which is assigned to u_ℓ by definition. Hence, case (i) of the Observation 3 must hold for ℓ'. Thus, there was not enough resource to schedule all the tiny jobs in H before $u_{\ell'+1}$ in $\tilde{S}$. On the other hand, all the jobs in H are scheduled before $u_{\ell'+1}$ in S^*, thus the resource consumption of the tiny jobs starting before $u_{\ell'+1}$ in S^* is not smaller than that in $\tilde{S}$. Moreover, the huge job assignment of the two schedules before u_q is the same. Since S^* is feasible, this is a contradiction. □

If j is assigned to an u_ℓ with $\ell < q$, then according to Claim 3, there exists a job j^* with $d_{j^*} \leq d_j$ and $S^*_{j^*} \geq u_\ell$. Let M_k be the machine which processes j in $\tilde{S}$. We have $\tilde{S}_j \leq u_\ell + (\varepsilon u_q + 3\varepsilon^2 u_q) + 3(q-2)\varepsilon^2 u_q = u_\ell + 4\varepsilon u_q$, since, on the one hand, the total processing time of the tiny jobs assigned to u_ℓ on M_k in $\tilde{S}$ is at most $\varepsilon u_q + 3\varepsilon^2 u_q$, and, on the other hand, for each $\nu < \ell$ the total processing time of the tiny jobs assigned to u_ν and M_k in $\tilde{S}$ is greater by at most $3\varepsilon^2 u_q$ than the same amount in S^* (Observation 2) and the huge job assignment is the same in $\tilde{S}$ and S^*. Therefore $L'_j(\tilde{S}) = \tilde{C}_j - d_j + D \leq u_\ell + 5\varepsilon u_q - d_j + D \leq u_\ell + 5\varepsilon u_q - d_{j^*} + D \leq L'_{j^*}(S^*) + 5\varepsilon u_q \leq L'_{\max}(S^*) + 5\varepsilon u_q$ follows.

Now suppose that j is scheduled at step 5. We will show that there exists a tiny job j^* such that $S^*_{j^*} \geq \tilde{S}_j - 5\varepsilon u_q$ with $d_{j^*} \leq d_j$. From this the proposition follows, since $0 < p_j, p_{j^*} \leq \varepsilon^2 u_q$ by definition. Let $\tilde{A}(t)$ denote the set of tiny jobs j' that are scheduled at step 5 such that $\tilde{S}_{j'} \geq t$, and $\tilde{B}(t) := \mathcal{S}' \setminus \tilde{A}(t)$. Likewise, let $A^*(t)$ denote the set of tiny jobs j' with $S^*_{j'} \geq t$, and $B^*(t) := \mathcal{S}' \setminus A^*(t)$.

Claim 4. *If $t \geq u_q$, then $p(\tilde{A}(t + 5\varepsilon u_q)) \leq p(A^*(t))$.*

Proof of Claim 4. Note that, if $t \geq u_q$ then the total processing time of the huge jobs in $[\max\{C^{part}_{\max}(k), u_q\}, t]$ on any M_k in S^* is at least the total processing time of the huge jobs in $[\max\{C^{part}_{\max}(k), u_q\}, t + 5\varepsilon u_q]$ on M_k in $\tilde{S}$, because $\tilde{S}_{j'} \geq S^*_{j'} + 5\varepsilon u_q$ if j' is huge and $S^*_{j'} \geq u_q$. Since $p(\tilde{A}(u_q)) \leq p(A^*(u_q)) + \varepsilon^2 u_q$ (apply Observation 2 to $\ell = q - 1$), and there is no gap before any tiny job on any machine M_k in $\tilde{S}$ after $\max\{C^{part}_{\max}(k), u_q\}$, the claim follows, because there is more time to schedule tiny jobs until $t + 5\varepsilon u_q$ in $\tilde{S}$ on any machine for any $t \geq u_q$ than until t in S^*. □

From the claim we deduce $p(\tilde{B}(\tilde{S}_j)) \geq p(B^*(\tilde{S}_j - 5\varepsilon u_q))$. It follows that there exists $j^* \in \{j\} \cup \tilde{B}(\tilde{S}_j)$ such that $j^* \in A^*(\tilde{S}_j - 5\varepsilon u_q)$. Since the tiny jobs are scheduled in EDD order in $\tilde{S}$, we have $d_{j^*} \leq d_j$, and we are done. □

Theorem 17. *[45] $Pm|nr = 1, p_j = a_j|L'_{\max}$ admits a PTAS.*

Proof. If we put together the above results we get that Algorithm B constructs a feasible schedule in polynomial time and the (shifted) lateness of this schedule is at most $L'_{\max}(S^{best}) \leq (1+\varepsilon)L'_{\max}(\tilde{S}) \leq (1+\varepsilon)(L'_{\max}(S^*)+6\varepsilon u_q) \leq (1 + 8\varepsilon)L'_{\max}(S^*)$ by Propositions 17 and 18. □

3.2.3 Outlook to further results

The makespan minimization problem has also been investigated in Györgyi and Kis [45]:

Theorem 18. *The problem $Pm|nr = const|C_{\max}$ admits a PTAS.*

We do not provide a detailed proof of this result, as it is similar to that of Theorem 17. Moreover, it is a a generalization of that of Theorem 13, but it is more involved, since we have to deal with m parallel machines instead of a single machine. Notice that this result is the strongest possible for the makespan objective, sine the problem is $\mathcal{NP}$-hard in the strong sense (see Proposition 8), and if the number of machines, or the number of resources are not constants, then the problem becomes APX-hard (see Corollary 13 and Theorem 11).

Chapter 4

Bilevel scheduling problems

In this chapter we are concerned with scheduling problems formulated as bilevel optimisation problems, which has its origins in market economy theory, and in particular in Stackelberg games [27]. Bilevel optimization is concerned with two-level optimization problems, in which there is a top level decision maker or *leader*, and one (or more) bottom level decision maker(s) or *follower(s)*. The leader decides first, and fixes those decision variables that are under her control. While making her decisions, she takes into account the possible responses of the follower in order to optimize her own objective function. The leader's decisions affect the constraints and/or the objective function of the follower. The follower decides second, and makes its decisions in view of those of the leader. However, its decisions affect the objective function of the leader or even the feasibility of the leader's solution. The follower also wants to optimize its own objective function. When solving bilevel optimization problems, we want to support the leader in making optimal decisions. In the *optimistic case*, the leader assumes that the follower chooses an optimal solution which is the most favorable for her, while in the *pessimistic case* the leader assumes that the follower chooses an optimal solution with the worst outcome for her. For an overview and references on bilevel optimizaiton, see Dempe [26, 27]. However, there are only sporadic results on bilevel machine scheduling problems, see e.g., [57], [79]. The connection between bilevel optimization and multi-criteria optimiozation in the context of linear programming has been explored by Fülöp [33].

We will consider two bilevel scheduling problems. In the first one, there is a parallel machine environment, and the leader has to assign jobs to machines, whereas the follower orders the jobs on each machine (Section 4.1). At both levels the objective function is the weighted sum of job completion times, but with different job weights at the two levels. We will study the complexity of the problem, its relation to multi-criteria optimization, and then we will

analyze various special cases. An important by-product is that we present a new polynomially solvable special case of the MAX-CUT problem in graphs. In the second problem class (Section 4.2) there is only a single machine, and each job has a deadline. The leader has to accept or reject jobs while maximizing the total weight of accepted jobs. On the other hand, the follower aims at minimizing the weighted sum of the completion time of the accepted jobs. Again, we will analyze the complexity, and consider various special cases.

We will frequently refer to the *weighted shortest processing time order* of the jobs with respect to some job weights w_j (WSPT order for short), in which job j precedes job k, if $w_j/p_j > w_k/p_k$, and there may be additional tie-breaking rules. Finally, we call a schedule *non-delayed*, if no job may be started earlier without violating some of the constraints of the scheduling problem.

The results of this chapter are based on Kis and Kovács [68].

4.1 The bilevel weighted completion time problem

In the *bilevel weighted completion time problem*, there are n jobs and m identical parallel machines. Each job has a processing time p_j and two non-negative weights, w_j^1 and w_j^2. The leader assigns jobs to machines, and the follower orders the jobs assigned to each machine. Let C_j denote the completion time of job j in a solution. The follower's objective is to minimize $\sum_{i=1}^{m} \sum_{j \in J_i} w_j^2 C_j$, where J_i is the set of those jobs assigned to machine i. In the optimistic case, the leader's objective is to minimize the total completion time $\sum_{j=1}^{n} w_j^1 C_j$, where the minimum is taken over all job assignment $J_1, \ldots, J_m$. In contrast, in the pessimistic case the leader wants to find an assignment of jobs to machines such that the maximum total weighted completion time is minimal by minimizing (over all job assignments) $\max \sum_{j=1}^{n} w_j^1 C_j$, where the maximum is taken over all optimal solutions of the follower with respect to a job assignment. The leader aims to find the sets J_i, $i = 1, \ldots, m$, such that her decision is optimal in the optimistic or in the pessimistic sense. Notice that such a solution cannot be modeled by imposing precedence constraints among the jobs, because the assignment of jobs to machines is not known in advance.

4.1.1 Preliminaries

The bilevel weighted completion time problem is a generalization of the parallel machine weighted completion time problem, which is as follows. There are m identical parallel machines, and n jobs each with a processing time p_j and weight w_j. Each job has to be assigned to a machine, and the jobs assigned to the same machine have to be sequenced such that the objective function $\sum_j w_j C_j$ is minimized, C_j being the completion time of job j. In the $\alpha|\beta|\gamma$ notation this problem is denoted as $P||\sum_j w_j C_j$. This problem is $\mathcal{NP}$-hard [93], strong $\mathcal{NP}$-hardness is claimed by Lenstra (cf. Brucker [13]). The special case $P||\sum_j C_j$ is solvable in polynomial time by network flow techniques. An important property of the optimal solutions is that on each machine the jobs are processed in non-decreasing processing time order, see e.g., [13].

4.1.2 Bilevel weighted completion time and Pareto optimality

Multi-criteria scheduling problems are thoroughly discussed in the literature, see e.g., the review of Hoogeveen [52]. In those problems, there are two or more criteria to evaluate solutions, and we associate with each solution a vector of objective function values. A central notion of multi-criteria optimization is that of *Pareto optimality*. Suppose there are k criteria, and S^1, S^2 are two solutions with values $(f_1^1, \ldots, f_k^1)$, and $(f_1^2, \ldots, f_k^2)$, respectively. We say that S^1 *Pareto-dominates* S^2 if $f_i^1 \leq f_i^2$ for each $i = 1, \ldots, k$. The Pareto-dominance is *strict* if $f_i^1 < f_i^2$ for at least one i. Then, a solution is *Pareto optimal* if it is not strictly Pareto-dominated by some other solution.

Now we demonstrate that the optimal solution of the bilevel weighted completion time problem is not Pareto optimal in general. In fact, we describe an instance where every optimal solution of the bilevel problem is strictly dominated by a feasible solution. This shows that the two notions of optimality are different.

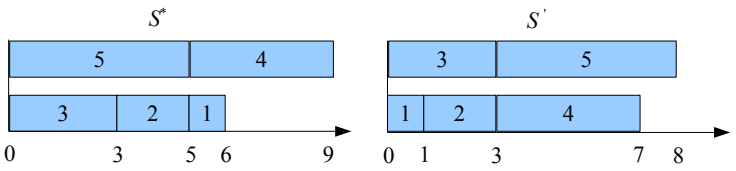

Figure 4.1: Two schedules for the bilevel weighted completion time problem.

Suppose there are 5 jobs, with processing times $p_j = j$ for $j = 1, \ldots, 5$.

The weights of the jobs are as follows: $w_j^1 = 1$, and $w_j^2 = p_j(c + p_j)$ for every job j, where c is a constant to be chosen later. Notice that w_j^2 induces a decreasing processing time order, since $w_j^2/p_j \geq w_k^2/p_k$ is equivalent to $c + p_j \geq c + p_k$. Therefore, in any optimal solution of the bilevel scheduling problem, the machines process the assigned jobs in decreasing processing time order, otherwise the solution is not optimal for the follower.

It can be shown that this problem admits the unique optimal solution S^* (up to permutation of the machines) in which $J_1^* = \{3, 2, 1\}$ and $J_2^* = \{5, 4\}$, and the machines process the assigned jobs in the given order. In this solution the job completion times for $j \in J_1^*$ are $C_3^* = 3$, $C_2^* = 5$, $C_1^* = 6$, and those for $j \in J_2^*$ are $C_5^* = 5$, $C_4^* = 9$, see Figure 4.1. Therefore, the optimal objective function value of the leader is $\sum_{j=1}^{5} C_j^* = 28$. On the other hand, the followers' objective function value is $\sum_{j=1}^{5} w_j^2 C_j^* = 86c + 322$. Therefore, the vector of objective function values is $(f_1^*, f_2^*) = (28, 86c + 322)$.

We construct another solution S': let $J_1' = \{1, 2, 4\}$ and $J_2' = \{3, 5\}$, and suppose the machines process the assigned jobs in increasing processing time order. Then $C_1' = 1$, $C_2' = 3$, $C_4' = 7$, and $C_3' = 3$, $C_5' = 8$. The leader's objective function value is $\sum_{j=1}^{5} C_j' = 22$. On the other hand, the follower's objective function value on this solution is $\sum_{j=1}^{5} w_j^2 C_j' = 84c + 352$. Therefore, $(f_1', f_2') = (22, 84c + 352)$. Clearly, this solution is not optimal for the follower, since by reversing the processing order on the two machines the objective function would decrease.

Now we compare the solutions S^* and S'. S' strictly Pareto-dominates S^*, i.e.,

$$(f_1^*, f_2^*) = (28, 86c + 322) > (22, 84c + 352) = (f_1', f_2')$$

if $c > 15$. Therefore, S^* is strictly Pareto-dominated.

4.1.3 Global ordering of jobs

Given the leader's decision about the job assignments J_i, $i = 1, \ldots, m$, the follower faces m independent $1||\sum_{j \in J_i} w_j^2 C_j$ problems, one for each machine. Hence, the follower is going to minimize the objective function $\sum_{i=1}^{m} \sum_{j \in J_i} w_j^2 C_j$ by sequencing the jobs according to the WSPT rule using the weights w^2, with ties broken according to w^1 (the direction of tie-braking differs in the optimistic and in the pessimistic case). Since the processing times and weights of the jobs do not depend on the machine assignment, the sequence on each machine is a sub-sequence of one global partial ordering. The global partial ordering in the optimistic case is:

job j precedes job k if $w_j^2/p_j > w_k^2/p_k$ or $(w_j^2/p_j = w_k^2/p_k$ and $w_j^1/p_j > w_k^1/p_k)$ (4.1)

while in the pessimistic case it is:

job j precedes job k if $w_j^2/p_j > w_k^2/p_k$ or $(w_j^2/p_j = w_k^2/p_k$ and $w_j^1/p_j < w_k^1/p_k)$ (4.2)

The above ordering is partial because it does not decide which of jobs j and k should be scheduled first if $w_j^2/p_j = w_k^2/p_k$ and $w_j^1/p_j = w_k^1/p_k$.

Proposition 19. *[68]*In the optimistic case there always exists an optimal non-delayed schedule for any non-negative job-weights. In the pessimistic case, if there exists a job j with $w_j^2 = 0$ and $w_j^1 > 0$, then the pessimistic bilevel scheduling problem admits no finite optimum. Otherwise, it always has a finite optimum.*

Proof. In the optimistic case, the follower always chooses the most favourable schedule for the leader among its optimal solutions with respect to a specific assignment of jobs to machines. Therefore, it always chooses a non-delay schedule, since $w_j^2 \geq 0$ for all jobs, and therefore, there is no gain in delaying a job. Since there exist only a finite number of non-delayed schedules, the problem always admits a finite optimum.

In contrast, in the pessimistic case, the leader assumes that the follower plays against her. Hence, if $w_j^2 = 0$ and $w_j^1 > 0$ for some job j, then the optimum value of the leader is infinite, since the follower has an optimal schedule with an arbitrarily large C_j value for any assignment of jobs to machines. If $w_j^2 = w_j^1 = 0$, then $w_j^1 C_j = 0$, and therefore such jobs can be scheduled arbitrarily by the follower after those jobs with $w_k^2 > 0$ without affecting the objective function of the leader. Finally, if $w_j^2 > 0$ for all the jobs, then any optimal solution of the follower is a non-delayed schedule, and therefore, the problem has a finite optimum. □

The above technical difficulties can be avoided by assuming that the follower always chooses a non-delayed optimal solution, which is quite reasonable in practice.

Lemma 17. *[68]*The optimistic weighted completion time problem always admits an optimum solution of finite value such that on each machine the jobs are ordered according to (4.1).*

Proof. The existence of an optimal schedule is guaranteed by Proposition 19. As for the structure of optimal schedules, the key observation is that once

the assignment of jobs to machines is fixed, the follower always chooses an optimal ordering of the set of jobs assigned to each machine. However, the single machine problem of machine i with job-set J_i is $1||\sum_{j\in J_i} w_j^2 C_j$, which can be solved by the WSPT rule. If the order of two jobs is arbitrary in an optimal solution, i.e., $w_j^2/p_j = w_k^2/p_k$, then the leader's preference can be taken into account which is expressed in the ordering (4.1). □

Lemma 18. *[68]*If there exists no job j with $w_j^2 = 0$ and $w_j^1 > 0$, or the follower has to choose a non-delayed optimal schedule, then the pessimistic weighted completion time problem admits an optimal solution of finite value such that on each machine the jobs are ordered according to (4.2).*

Proof. Similar to the optimistic case. □

The following lemma shows that in general it is possible to change the followers' weights so that w^2 alone defines one unambiguous complete global ordering of jobs, and the sequences on individual machines will be subsequences of that global ordering. This is useful, because it ensures that – whenever this conversion can be performed – our propositions hold both for the optimistic and the pessimistic cases.

Lemma 19. *[68]*Any instance Π of the bilevel weighted completion time problem can be converted into an instance $\bar{\Pi}$ such that the followers' WSPT order is unique, and all the optimal solutions of $\bar{\Pi}$ are optimal for Π as well.*

Proof. We define new follower's job weights $\bar{w}^2$ as follows: In the optimistic case re-index the jobs (i.e., $j < k$ iff job j precedes k) with respect to partial order (4.1), break ties arbitrarily, while in the pessimistic case with respect to (4.2). We define the instance $\bar{\Pi}$ with n jobs having processing times p_j, leader's weights w_j^1 and followers' weights $\bar{w}_j^2 = (n-j+1)p_j$. Since $\bar{w}_j^2/p_j = n-j+1$, it follows that job j precedes job k iff j precedes k in the selected global ordering of jobs. Moreover, the WSPT order with respect to $\bar{w}_j^2$ induces a total order of jobs. □

4.1.4 Complexity

Below we prove that the decision version of the bilevel weighted completion time problem is NP-complete in the strong sense. The decision version of the bilevel scheduling problem asks whether there is a feasible solution with a leader's objective function value not worse than a given bound K. Notice that such a solution has to be optimal for the follower.

Proposition 20. *[68]* The bilevel weighted completion time problem is NP-complete in the strong sense.*

Proof. Membership to NP: The witness consists of a partitioning $J_1, \ldots, J_m$ of the jobs, and a completion time $\bar{C}_j$ for each job j. One can easily verify whether $\sum_{j=1}^{n} w_j^1 \bar{C}_j \leq K$. Moreover, for each machine i, an instance of the problem $1||\sum_j w_j C_j$ is specified by job-set J_i using weights w_j^2. Each of these problems can be solved in polynomial time by the WSPT rule and let W_i^* denote the optimum value for machine i. The job-completion times $\bar{C}_j$, $j \in J_i$, correspond to an optimal schedule, if the jobs in J_i do not overlap, and $\sum_{j \in J_i} w_j^2 \bar{C}_j = W_i^*$. All these computations can be done in polynomial time in the length of the input and that of the witness given above.

$\mathcal{NP}$*-hardness*: The bilevel problem contains the strongly $\mathcal{NP}$-hard $P||\sum w_j C_j$ problem, which can be seen by assigning $w_j^1 = w_j^2 := w_j$. Then, any solution is optimal (and feasible) to the bilevel problem if and only if it is an optimal solution to the parallel machine problem as well. □

Remark 3. *An open problem is whether the problem remains $\mathcal{NP}$-hard if $w_j^1 = 1$ or $p_j = p$ for all jobs j.*

Next we show the connection to the MAX m-CUT problem, which is as follows. Given a complete graph K_n with edge weights $c(i,j)$, determine a partitioning of the nodes into m nonempty subsets such that the total weight of edges connecting the nodes in the different subsets is maximal.

Now, the bilevel scheduling problem is equivalent to a MAX m-CUT problem (unless $m \geq n$, in which case the scheduling problem is trivial). This can be shown by assigning weights to the edges as follows: order the vertices with respect to (4.1) in the optimistic case, and (4.2) in the pessimistic case. The weight of edge (j,k) is

$$c(j,k) := p_j w_k^1 \quad \text{if job } j \text{ precedes job } k \text{ in the ordering.} \tag{4.3}$$

Theorem 19. *[68]* The optimal solution of the MAX m-CUT problem with edge weights (4.3) yields an optimal solution of the bilevel scheduling problem, and vice versa.*

Proof. Consider first the optimistic case. By Lemma 17, there is an optimal solution respecting the ordering (4.1) on each machine. In this ordering, the total weighted completion time on machine i with set of jobs J_i is $\sum_{j \in J_i} w_j^1 C_j = \sum_{j \in J_i} w_j^1 (p_j + \sum_{k \in J_i : k \prec j} p_k)$, where $k \prec j$ iff job k precedes job j. The term $\sum_{j \in J_i} w_j^1 p_j$ being constant, the problem can be reformulated as

follows:

$$\min\{\sum_{i=1}^{m}\sum_{j\in J_i}\sum_{k\in J_i:k\prec j} p_k w_j^1 \mid J_1,\dots J_m \text{ is a partitioning of } J\},$$

where $J=\{1,\dots,n\}$. Since the total weight of all arcs is $\sum_{j\in J}\sum_{k\prec j} p_k w_j^1$, the total weight of those arcs connecting nodes in different subsets of a partitioning $J_1,\dots,J_m$ of J is

$$\sum_{j\in J}\sum_{k\prec j} p_k w_j^1 - \sum_{i=1}^{m}\sum_{j\in J_i}\sum_{k\in J_i:k\prec j} p_k w_j^1.$$

Therefore, minimizing $\sum_{i=1}^{m}\sum_{j\in J_i}\sum_{k\in J_i:k\prec j} p_k w_j^1$ is equivalent to maximizing the total weight of those arcs connecting nodes in different subsets of a partitioning.

The proof of the pessimistic case goes along the same lines using Lemma 18 and ordering (4.2). □

Consequently, by solving the MAX m-CUT problem in the complete n-graph with appropriate edge-weights, we can solve the bilevel scheduling problem to optimality. Since the MAX m-CUT problem is $\mathcal{NP}$-hard in the strong sense in general even for $m=2$ [58], we do not have a polynomial time algorithm at hand for solving our bilevel scheduling problem. Next we study some special cases.

4.1.5 Special cases

In this section we consider two polynomially solvable special cases of the bilevel scheduling problem. We say that weights w^1 and w^2 induce the same ordering of jobs if $w_j^1/p_j \le w_k^1/p_k$ if and only if $w_j^2/p_j \le w_k^2/p_k$ for each pair of jobs j and k.

Proposition 21. *[68]*Suppose w^1 and w^2 induce the same ordering of jobs. Then an optimal solution of the bilevel scheduling problem is obtained by solving the (single level) problem $P||\sum w_jC_j$ with $w_j=w_j^1$.*

The above parallel machine scheduling problem is $\mathcal{NP}$-hard in general. An important special case is when all the weights are equal to 1.

Further special cases occur when $w^1\equiv 1$, but w^2 induces an increasing or decreasing processing time order, i.e., $w_j^2/p_j > w_k^2/p_k$ if and only if (1) $p_j < p_k$, or (2) $p_j > p_k$. For instance, the weights $w_j^2 = p_j+1$ induce an increasing processing time order, whereas the weights $w_j^2 = p_j-1$ induce a decreasing processing time order.

$w^1 \equiv 1$ and w^2 induces a non-decreasing processing time order

If $w^1 \equiv 1$, then it would be optimal for the leader to process the jobs in non-decreasing processing time order on each machine. Notice that w^2 induces the same order on the machines. Therefore, by Proposition 21 we know that the problem is equivalent to a single level parallel machine scheduling problem. Moreover, since $w^1 \equiv 1$, this single level problem takes the from $P||\sum_j C_j$, which can be solved in polynomial time by network flow techniques, cf. [13].

Proposition 22. *[68]* The bilevel weighted completion time scheduling problem with $w^1 \equiv 1$ and w^2 inducing a non-decreasing processing time order can be solved in polynomial time.*

$w^1 \equiv 1$, and w^2 induces a non-increasing processing time order

Suppose the jobs are indexed in non-increasing processing time order, i.e., $p_1 \geq p_2 \geq \cdots \geq p_n$. We claim that the bilevel scheduling problem admits an optimal solution such that each of the m machines processes consecutive jobs in the above ordering. Moreover, an optimal solution can be found in polynomial time. We say that job k is a *successor* of job j if k succeeds j in the non-increasing processing time order.

Lemma 20. *[68]* If $w^1 \equiv 1$, and w^2 induces a non-increasing processing time order, then the bilevel scheduling problem admits an optimal solution such that the set of jobs J_i assigned to machine i consists of the jobs $\pi_{i-1}+1, \ldots, \pi_i$, where $\pi_0 = 0$, and $1 \leq \pi_1 \leq \pi_2 \leq \cdots \leq \pi_m = n$.*

Proof. We will exploit the equivalence to the MAX m-CUT problem. Since $w_j^1 = 1$ for each job j, and w^2 induces a non-increasing processing time order, we can model the bilevel scheduling problem by a complete graph K_n with nodes identified with the jobs, and for each pair of distinct nodes (j,k), the weight of the edge incident with j and k is $c(j,k) := \max\{p_j, p_k\}$. Clearly, if job k is a successor of job j, then $c(j,k) = p_j$, otherwise $c(j,k) = p_k$.

Any optimal solution of the bilevel scheduling problem can be fully characterised by an assignment of jobs to machines, since the order of jobs assigned to a machine is determined by the decreasing processing time order. In terms of the MAX m-CUT problem, the optimal assignment is equivalent to an m-partition $S_1, \ldots, S_m$ of the nodes of K_n such that the total weight of those edges connecting nodes in distinct subsets in the partitioning is maximal. Let $C = \{(j,k) \mid \exists i \neq \ell$ such that $j \in S_i$ and $k \in S_\ell\}$. W.l.o.g. suppose $1 \in S_1$, and assume that there exist $k_1 \geq 1$ and $k_2 > k_1$ such that $\{1, \ldots, k_1 - 1, k_2\} \subset S_1$, but $k_1 \in S_i$ and $S_i \neq S_1$. We apply the following transformation to $S_1, \ldots, S_m$. Let $S^* = S_1 \cup S_i$, and then let S_i'

consist of the last $\max\{|S_1| - k_1 + 1, |S_i|\}$ jobs of S^* in the decreasing processing time order, $S'_1 := S^* \setminus S'_i$, and $S'_\ell := S_\ell$ for $\ell \in \{1, \ldots, m\} \setminus \{1, i\}$. Let C' consist of those edges of K_n connecting nodes in distinct subsets of the partitioning $S'_1, \ldots, S'_m$ of the nodes of K_n. We claim that the total weight of edges in C' is not smaller than that of C. Since $S'_1, \ldots, S'_m$ is equivalent to an assignment of jobs to machines (since the machines are identical), this means that the objective function of the bilevel scheduling problem does not increase with the above transformation. Since this transformation ensures that S_1 contains all the jobs $1, \ldots, k_1$, by repeated application we can make sure that S_1 consists of consecutive jobs. Repeating this argument for each subset of the partitioning, we obtain the desired optimal solution.

To prove our claim, let $C(S_1, S_i) \subseteq C$ denote the set of edges connecting the nodes in S_1 and S_i. Similarly, let $C'(S'_1, S'_i) \subseteq C'$ be the set of edges with endpoints in S'_1 and S'_i. Since C and C' differ only in the set of arcs connecting S_1 and S_i, and S'_1 and S'_i, respectively, we have

$$\sum_{(j,k)\in C} c(j,k) - \sum_{(j,k)\in C(S_1,S_i)} c(j,k) = \sum_{(j,k)\in C'} c(j,k) - \sum_{(j,k)\in C'(S'_1,S'_i)} c(j,k).$$

Therefore, it suffices to show that

$$\sum_{(j,k)\in C(S_1,S_i)} c(j,k) \leq \sum_{(j,k)\in C'(S'_1,S'_i)} c(j,k).$$

For each job $j \in S_1 \cup S_i$, let n_j and n'_j denote the number of arcs (j,k) in $C(S_1, S_i)$ and $C'(S'_1, S'_i)$, respectively, such that k succeeds j. We clearly have

$$\sum_{j\in S_1\cup S_i} n_j \leq |S_1| \cdot |S_i| \leq |S'_1| \cdot |S'_i| = \sum_{j\in S'_1} n'_j.$$

The first inequality trivially holds. The second inequality is ensured by the transformation. The last equality follows from the fact that any job in S'_i succeeds all jobs in S'_1. Since $n'_j \leq |S'_i|$ for every $j \in S'_1$, we also have $n'_j = |S'_i|$. Moreover, for each $j \in S_1$, $n_j \leq |S_i|$, and for each $j \in S_i$, $n_j \leq |S_1| - k_1 + 1$ (since jobs $1, \ldots, k_1 - 1$ all belong to S_1 and all jobs succeed them). Since $|S'_i| = \max\{|S_i|, |S_1| - k_1 + 1\}$, we also have $n_j \leq |S'_i|$. Now, we have

$$\sum_{(j,k)\in C'(S'_1,S'_i)} c(j,k) = \sum_{j\in S'_1} n'_j p_j.$$

On the other hand,

$$\sum_{(j,k)\in C(S_1,S_i)} c(j,k) = \sum_{j\in S_1\cup S_i} n_j p_j \leq \sum_{j\in S'_1} n'_j p_j.$$

Here, the last inequality follows, since $n_j \leq |S'_i|$ for each $j \in S_1 \cup S_i$, $n'_j = |S'_i|$, for each $j \in S'_1$, $\sum_{j \in S_1 \cup S_i} n_j \leq \sum_{j \in S'_1} n'_j$, $S_1 \cup S_i = S'_1 \cup S'_i$, and all jobs in S'_i succeed all jobs in S'_1. However, this implies our claim. □

Theorem 20. *[68]* The special case with $w^1 \equiv 1$ and w^2 inducing a decreasing processing time order can be solved in polynomial time.*

Proof. We define a directed graph. Each node is a tuple $([j_1, j_2], \ell)$, where $[j_1, j_2]$ is an interval of jobs, and ℓ is a depth label with $1 \leq \ell \leq m$. Moreover, there is an initial node $([0, 0], 0)$. We direct an arc from $([j_1, j_2], \ell)$ to $([j_3, j_4], \ell + 1)$ iff $j_3 = j_2 + 1$. In particular, there is an arc from the initial node to each node of the form $([1, j_2], 1)$. The *cost* of any arc directed to node $([j_3, j_4], \ell)$ is the total completion time of the jobs in the interval $[j_3, j_4]$, i.e., the sum $\sum_{j \in [j_3, j_4]} (m_j + 1)p_j$, where m_j is the number of those jobs $k \in [j_3, j_4]$ that succeed j. A shortest path from node $([0, 0], 0)$ to one of the form $([j, n], m)$ gives an optimal solution to the scheduling problem. Namely, such a path has $m + 1$ nodes with depth labels 0 through m, the intervals are disjoint and contain all the jobs, and for each $1 \leq \ell \leq m$, the node with depth label ℓ represents the assignment of jobs to the ℓ-th machine. Since this graph has $O(mn^2)$ nodes, and is acyclic, the shortest path can be found in polynomial time in the size of the input. □

4.1.6 Polynomially solvable special cases of the MAX m-CUT problem

For the sake of completeness, we reformulate the results of the previous section in terms of the MAX m-CUT problem.

Theorem 21. *[68]* Consider the complete graph K_n with a weight $p_j \geq 0$ associated with each node j. Let $c(j, k) = \max\{p_j, p_k\}$. Then the MAX m-CUT problem for K_n with edge-weights $c(j, k)$ is solvable in polynomial time.*

Proof. Order the nodes of K_n in non-increasing p_j order, and apply Theorem 19 and Theorem 20. □

Further on, Proposition 22 and Theorem 19 imply the following result:

Theorem 22. **Consider the complete graph K_n with a weight $p_j \geq 0$ associated with each node j. Let $c(j, k) = \min\{p_j, p_k\}$. Then the MAX m-CUT problem for K_n with edge-weights $c(j, k)$ is solvable in polynomial time.*

4.1.7 Beyond the $w^1 \equiv 1$ special case

If the job weights of the leader are not uniform, then the bilevel total weighted completion time problem is not easier to approximate than its single level special case. This can be seen by considering the instances with $w^2 \equiv w^1$. Therefore, we cannot hope for stronger approximation results than in the single level case in general. For the problem $P2||\sum_j w_j C_j$, Sahni [93] describes an $(1+\varepsilon)$-approximation algorithm with $O(n^2/\varepsilon)$ time complexity (an FPTAS). For any fixed number of machines, the FPTAS of Schuurman and Wöginger [95] for $Pm||\sum_j w_j C_j$ can be generalised to our problem. To apply their algorithm to the bilevel total weighted completion time problem, it suffices to note that the FPTAS for $Pm||\sum_j w_j C_j$ is based on a dynamic program which processes the jobs in decreasing w_j/p_j order. In our case, we apply the same algorithm for a job sequence (4.1) in the optimistic case, and (4.2) in the pessimistic case. Therefore, we have

Theorem 23. *[68]* There is an FPTAS for the bilevel total weighted completion time problem with a fixed number of machines.*

The time complexity of the FPTAS is $O(nL^m)$, where $L = \lceil \log_\Delta p_{sum} \rceil$ with $p_{sum} = \sum_{j=1}^{n} p_j$, and $\Delta = 1 + \frac{\varepsilon}{2n}$, ε being the desired relative error.

4.2 The bilevel order acceptance problem

In the *bilevel order-acceptance problem* There are n jobs and a single machine. Each job has a processing time p_j, a deadline d_j, and two non-negative weights, w_j^1 and w_j^2. The weight w_j^1 is the leader's penalty (loss of profit) of rejecting job j, and w_j^2 is the weight for the follower. The leader has to select a subset of jobs that have to be completed by their respective deadlines. However, the follower aims to minimize the weighted completion time of the accepted jobs, but it is not obliged to take into consideration the deadlines. More formally, the leader's objective is $\min \sum_j w_j^1 R_j$, where $R_j = 1$ if and only if job j is rejected. In turn, the follower's objective is $\min \sum_{j \in J'} w_j^2 C_j$, where $J' = \{j \mid R_j = 0\}$ is the set of accepted jobs, and C_j is the completion time of job j. The leader's objective function is the same in the optimistic and in the pessimistic cases. In the optimistic case, the leader has to accept a subset of jobs such that there exists *at least one* optimal sequence for the follower which respects all the job-deadlines of the accepted jobs. On the other hand, in the pessimistic case, the leader has to choose a subset of jobs such that *all* the optimal solutions of the follower observes all the deadlines of the accepted jobs.

This is a bilevel optimization problem, because the leader can only accept or reject the jobs, while the follower determines an optimal sequence without regarding the jobs' deadlines. If the follower's solution violates some of the job-deadlines, then this solution is unfeasible.

4.2.1 Preliminaries

The bilevel order acceptance problem is a generalization of the single-machine weighted number of late jobs problem. In that problem there are n jobs, each job has a processing time p_j, a due-date d_j, and a weight w_j. A sequence of jobs is sought such that the total weight of those jobs competed after their due-dates is minimal. This problem is denoted by $1||\sum_j w_j U_j$. In the decision version of the problem, there is also given a constant K and one asks whether there is a feasible solution with total weight of late jobs not greater than K. It is well-known that there always exists an optimal solution such that those jobs completed on time are processed in earliest due-date order (EDD order for short), i.e., if both of the jobs j and k are completed before their due-dates, then j is processed before k, if $d_j \leq d_k$ (cf. [20]).

4.2.2 Bilevel order acceptance and Pareto optimality

Consider the instance of the bilevel order acceptance problem in Table 4.1. There are only four candidate job sets that may be selected by the leader: $\{1\}$, $\{2\}$, $\{3\}$, and $\{1,2\}$, and it is easy to verify that no other job set may be completed on time. The job set $\{1,2\}$ admits two sequences, $S^1 = (1,2)$ and

Table 4.1: Problem data for the bilevel order acceptance problem.

Job j	p_j	d_j	w_j^1	w_j^2
1	1	1	2	1
2	1	2	2	3
3	2	2	3	10

$S^2 = (2,1)$. The leader's objective function value is $f_1^1 = f_1^2 = w_3^1 = 3$ for both schedules. However, schedule S^1 is not optimal for the follower, because in the WSPT order job 2 precedes job 1, since $w_2^2/p_2 = 3 > 1 = w_1^2/p_1$. Therefore, S^1 cannot be an optimal solution of the bilevel scheduling problem. On the other hand, in S^2 job 1 completes after its due-date, therefore, it is not feasible for the leader. In S^1 the job completion times are $C_1^1 = 1$ and $C_2^1 = 2$, whence the objective function value of the follower is $\sum_{j\in\{1,2\}} w_j^2 C_j^1 = 7$. The

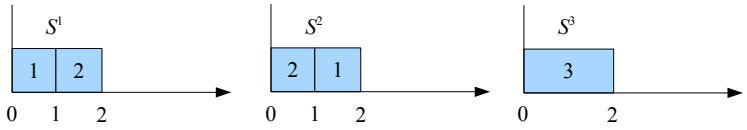

Figure 4.2: Three schedules for the bilevel order acceptance problem.

value of S^1 is $(f_1^1, f_2^1) = (3, 7)$, see Fig. 4.2. Now consider the job set $\{3\}$, the corresponding schedule is $S^3 = (3)$. Clearly, $(f_1^3, f_2^3) = (4, 20)$. It is easy to verify that S^3 is the unique optimal solution of the bilevel scheduling problem.

Comparing the values of S^1 and S^3, we see that S^1 strictly Pareto-dominates S^3.

4.2.3 Global ordering of jobs

Given the leader's decision about the selection of jobs J' to be completed on time, the follower sequences the jobs in WSPT order (with respect to weights w_j^2) to minimize its objective function. In case of ties, there is a distinction between the optimistic and the pessimistic cases. We define two global orderings of jobs such that the optimal optimistic and pessimistic solutions, respectively, are sub-sequences of the global job orders. In the optimistic case the global order is

$$\text{job } j \text{ precedes job } k \text{ if } w_j^2/p_j > w_k^2/p_k, \text{ or } (\ w_j^2/p_j = w_k^2/p_k \text{ and } d_j < d_k), \tag{4.4}$$

while in the pessimistic case it is

$$\text{job } j \text{ precedes job } k \text{ if } w_j^2/p_j > w_k^2/p_k, \text{ or } (\ w_j^2/p_j = w_k^2/p_k \text{ and } d_j > d_k). \tag{4.5}$$

Proposition 23. *[68] Both the optimistic and the pessimistic order acceptance problem always admits an optimal non-delayed schedule. In particular, in the pessimistic case, no job with $w_j^2 = 0$ can be accepted.*

Proof. First notice that it is feasible for the leader not to choose any jobs, so the set of feasible solutions is not empty. Moreover, in both the optimistic and the pessimistic case, the follower schedules those accepted jobs j with $w_j^2 > 0$ without any delay before them. The same holds in the optimistic case for accepted jobs j with $w_j^2 = 0$ as well, since such schedules are the most favorable for the leader.

Now consider the pessimistic case and suppose the leader accepts some jobs with $w_j^2 = 0$ (if such a job exists). Since the follower plays against the leader, it may answer a schedule which delays some of the accepted jobs j with $w_j^2 = 0$ beyond their deadlines. Such a schedule is though optimal for the follower, but it is unfeasible for the leader. Therefore, the leader cannot accept any jobs with $w_j^2 = 0$. □

If the follower must choose a non-delayed optimal schedule, then also in the pessimistic case jobs with $w_j^2 = 0$ can be accepted by the leader.

Lemma 21. *[68] There exists an optimal solution for the optimistic bilevel order acceptance problem such that the jobs are sequenced according to (4.4).*

Proof. The existence of an optimal schedule is ensured by Proposition 23. Let J^* be the leader's optimal selection of jobs. The follower schedules the jobs in J^* by the WSPT rule with job-weights w_j^2. In case of ties, i.e., $w_j^2/p_j = w_k^2/p_k$, it can always schedule first the job with smaller due-date, which is expressed in (4.4). □

Lemma 22. *[68] There exists an optimal solution for the pessimistic bilevel order acceptance problem such that the jobs are sequenced according to (4.5).*

Proof. The existence of an optimal solution is guaranteed by Proposition 23. Let J^* be the leader's optimal selection of jobs. If $w_j^2 > 0$ for all $j \in J^*$, or no job may be delayed unnecessarily, the follower schedules the jobs in J^* by the WSPT rule with job-weights w_j^2. The worst case for the leader is that in case of ties, i.e., $w_j^2/p_j = w_k^2/p_k$, the job with larger due-date is scheduled first, which is expressed in (4.5). □

Finally, similarly to Lemma 19, one can prove the following:

Lemma 23. *[68] Any instance Π of the bilevel order acceptance problem can be converted into an instance $\bar{\Pi}$ such that the followers' WSPT order is unique, and all the optimal solutions of $\bar{\Pi}$ are optimal for Π as well.*

4.2.4 Complexity

Given a constant K, in the optimistic case the decision version of the bilevel order acceptance problem asks whether there exists a subset $J' \subseteq J$ of jobs such that

1. $\sum_{j \in J \setminus J'} w_j \leq K$, and

2. there exists a WSPT order $\prec$ of the jobs in J' with $C_j \leq d_j$ for all $j \in J'$, where $C_j = p_j + \sum_{k \in J' : k \prec j} p_k$ is the completion time of job j in the order $\prec$.

In contrast, in the pessimistic case the decision problem asks whether there exists $J' \subseteq J$ such that

1. $\sum_{j \in J \setminus J'} w_j \leq K$, and

2. for every WSPT order $\prec$ of the jobs in J', $C_j \leq d_j$ for all $j \in J'$, where $C_j = p_j + \sum_{k \in J' : k \prec j} p_k$ is the completion time of job j in the order $\prec$.

Notice that if the WSPT order is unique for each subset of jobs, then the optimistic and the pessimistic cases of the problem coincide.

Proposition 24. *[68] The bilevel order acceptance problem is NP-complete both in the optimistic and in the pessimistic cases.*

Proof. Membership of NP follows from the fact that the follower's problem can be solved in polynomial time. Concerning $\mathcal{NP}$-hardness, for any instance Π_1 of the single-level $1||\sum_j w_j U_j$ problem, we define an instance of the bilevel order acceptance problem Π_2 with n jobs, processing times p_j, and job weights $w_j^1 = w_j$, and w_j^2 inducing an EDD order, i.e., $w_j^2/p_j \geq w_k^2/p_k$ if and only if $d_j \leq d_k$ (such weights clearly exist). Moreover, w_j^2 can be chosen such that the EDD order is unique. Therefore, the optimistic and the pessimistic cases coincide. Finally, Π_2 has the same constant K as Π_1.

First suppose Π_1 admits a solution with value at most K. Since we may assume that those jobs completed on time are processed in EDD order, this immediately yields a feasible solution for the bilevel scheduling problem with the same objective function value for the leader. Conversely, suppose the bilevel problem instance Π_2 admits a feasible solution with value at most K. Then this solution is also feasible for the single-level problem instance Π_1 with the same value. □

The problem is not strongly $\mathcal{NP}$-hard, since it is solvable in pseudo-polynomial time, see below.

Notice that the above proof shows that if the job weights w_j^2 induce an EDD order, then the bilevel problem becomes equivalent to the single level $1||\sum_j w_j U_j$ problem. But, this is not true in general.

4.2.5 A dynamic program for the general case

A modified version of the dynamic program proposed by Lawler & Moore [77] for $1||\sum_j w_j U_j$ is applicable for solving our bilevel problem. In that algorithm the jobs are processed in EDD order, i.e., the jobs are reindexed such that $d_1 \leq d_2 \leq \cdots \leq d_n$. Let $F_j(t)$ be the value of the optimal schedule of the problem involving jobs $1, \ldots, j$ that ends at time t. If $0 \leq t \leq d_j$ and job j is on time in the schedule corresponding to $F_j(t)$, then $F_j(t) = F_{j-1}(t - p_j)$. Otherwise, $F_j(t) = F_{j-1}(t) + w_j$. If $t > d_j$, then $F_j(t) = F_{j-1}(t) + w_j$ because job j is late.

We modify the algorithm of Lawler & Moore by processing the jobs in order (4.4) in the optimistic case, and in order (4.5) in the pessimistic case, and the jobs are reindexed to reflect the appropriate ordering. Let $T = \sum_{j \in J} p_j$. The recursion from $j-1$ to j is:

$$F_j(t) = \begin{cases} \min\{\ F_{j-1}(t - p_j),\ F_{j-1}(t) + w_j^1\ \} & \text{for } t = 0, \ldots, d_j, \\ F_{j-1}(t) + w_j^1 & \text{for } t = d_j + 1, \ldots, T, \end{cases}$$

with $F_0(0) = 0$, $F_j(t) = \infty$ for $t < 0$, and $j = 0, \ldots, n$; or $j = 0$ and $1 \leq t \leq T$.

The smallest $F_n(t)$ gives the optimum value, and by some book-keeping one also gets the optimal solution. Namely, let $v_j(t) = 0$ if $F_j(t) = F_{j-1}(t - p_j)$, and $v_j(t) = 1$ otherwise. Suppose $F_n(t^*) \leq F_n(t)$ for all t. Starting from $v_n(t^*)$, we can determine the optimal solution by visiting the jobs backward. In the general step, if $v_j(t) = 1$, then job j is rejected, and we proceed with $v_{j-1}(t)$. Otherwise, job j is accepted, and we proceed with $v_{j-1}(t - p_j)$. Repeating this until all the jobs are checked, we get a subset of accepted jobs. Since $v_j(t) = 0$ only if $t \leq d_j$, in the resulting solution all accepted jobs are completed on time. Moreover, the processing order respects the follower's WSPT order by construction.

The time and space complexity of the algorithm is $O(nT)$. Therefore, we have proved the following result.

Theorem 24. *[68] The bilevel order acceptance problem can be solved in $O(nT)$ time.*

4.2.6 Polynomial time algorithm for the $w^1 \equiv 1$ special case

This special case can be solved by a modified version of the Moore-Hogdson algorithm [81]. In that algorithm, the jobs are processed in EDD order.

Starting with an empty schedule, the jobs are appended to the end of the growing schedule one-by-one. If the appended job j completes late, then the job k in the partial schedule with the largest processing time p_k is rejected and removed from the schedule ($j = k$ is allowed). Note that removing at most one job from the schedule always yields a feasible schedule.

We modify the Moore-Hogdson algorithm by processing the jobs in the order (4.4) (optimistic case) or (4.5) (pessimistic case). Our modified algorithm appends jobs to the end of the schedule one-by-one and removes the actual lengthiest job when necessary exactly as the original Moore-Hogdson algorithm. The algorithm runs in $O(n \log n)$ time.

We adapt the proof of [13], page 86, of the soundness of the Moore-Hogdson algorithm to our more general case. In the following proof, $|\sigma|$ denotes the number of jobs in some schedule σ.

Theorem 25. *[68] The Modified Moore-Hogdson algorithm provides an optimal solution to the Bilevel order acceptance problem.*

Proof. If all the jobs can be processed on time, then the algorithm obviously finds this optimal solution. Otherwise, the algorithm removes at least one job from the schedule being constructed. Let j be the job that is removed first, in the step where job k is appended to the schedule. This implies that at least one of the jobs $1, ..., k$ is missing from every feasible schedule. Since j has the longest processing time among these jobs, any partial schedule σ' of the first k jobs that includes j but misses job h with $1 \leq h \leq k$, completes not earlier than the partial schedule $\sigma_{jh} = \sigma' \setminus \{j\} \cup \{h\}$ obtained by replacing j with h (the jobs are sequenced in the global order). Moreover, if no job is late in σ', then so is in σ_{jh}, because σ_{jh} schedules a subset of jobs in $\{1, \ldots, k-1\}$ and the global ordering of the latter yields a schedule in which no job is late. Hence, there exists an optimal schedule that does not contain j.

The rest of the proof goes by induction on the number of jobs n. Clearly, the algorithm is sound for $n = 1$. Assume it is correct for all instances involving $n - 1$ jobs. For n jobs, let σ be the schedule constructed by the algorithm and σ^* an optimal schedule with $j \notin \sigma^*$, where j is the first job removed while constructing σ. Observe that when our algorithm is applied to the problem involving jobs $\{1, ..., j-1, j+1, ..., n\}$ only, it constructs σ again, and this schedule is optimal for the reduced problem. Since σ^* is also a feasible solution for the reduced problem, we have $|\sigma| \geq |\sigma^*|$, and hence, σ is optimal for the original problem, too. □

4.2.7 Outlook to further results

The results of this chapter are of theoretical flavor, however, in Kis and Kovács [68] a linear programming based heuristic procedure is devised for solving the variant where the leader has unit weights for all the jobs, but the follower may have arbitrary job weights. The computational results show that our algorithm is able to find solutions 23% off the optimum in the worst case. Furthermore, in [74] we propose a practical algorithm for the Bilevel order acceptance problem based on constraint programming. Our method is far more efficient on a large variety of instances than general constraint programming methods. In Kis and Kovács [69] we have defined and analyzed the *bilevel lotsizing problem*, where both the leader and the follower solves an uncapacitated lot-sizing problem, but the follower's demand is set by the leader.

dc_1311_16

Chapter 5

Appendices

5.1 Numerical results for Section 2.1

Table 5.1: Number of times algorithms H, B^+ and B^- proved optimality in each class.

	$n = 10$			$n = 20$			$n = 50$		
s	$r = 3$	10	20	3	10	20	3	10	20
2	10 10/10	10 10/10	10 10/10	10 10/10	10 10/10	10 10/10	7 10/10	5 10/10	0 10/10
5	10 10/10	10 10/10	10 10/10	7 10/10	1 10/10	0 10/10	0 10/10	0 10/10	0 10/8
10	10 10/10	4 10/10	5 10/10	0 10/10	0 10/10	0 10/10	0 10/8	0 2/1	0 0/0
15	1 10/10	0 10/10	2 10/10	0 10/10	0 7/7	0 10/5	0 2/0	0 0/0	0 0/0
20	1 10/10	1 10/10	1 10/10	0 9/8	0 4/2	0 3/2	0 0/0	0 0/0	0 0/0

Table 5.2: The average of $ub(B^+)/ub(H)$ and, when different, the average of $ub(B^-)/ub(H)$.

	$n = 10$			$n = 20$			$n = 50$		
s	$r = 3$	10	20	3	10	20	3	10	20
2	1	1	1	1	1	1	1	1	1
5	1	1	1	0.99	0.99	0.99	0.95	0.96	0.99
10	1	1	0.99	0.86	0.97	0.98	0.73	0.94/0.96	0.97/0.99
15	0.88	0.98	0.98	0.63	0.94	0.97	0.63/0.74	0.92/1.08	1.03/1.11
20	0.89	0.98	0.98	0.71	0.91	0.97	0.56/0.69	0.92/1.1	0.96/1.21

Table 5.3: The average of $lb(B^+)/ub(B^+)$ and, when different, the average of $lb(B^-)/ub(B^-)$.

	$n = 10$			$n = 20$			$n = 50$		
s	$r = 3$	10	20	3	10	20	3	10	20
2	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1/0.99	0.96/0.90	0.93/0.88
15	1	1	1	1	0.99	1	0.86/0.68	0.85/0.70	0.80/0.70
20	1	1	1	0.99	0.97/0.95	0.98/0.96	0.77/0.51	0.66/0.49	0.72/0.53

Table 5.4: Some details of the computations of algorithm B^+.

	$n = 10$	$n = 20$	$n = 50$
Avg. time horizon T	28	36	45.08
Avg. CPU time in seconds	4.3	125	372
Avg. no. of search tree nodes	176	1697	1096
Avg. no. of flow cover cuts added	111	189	239
Avg. no. of fractional cuts added	9	12	13
Avg. no. of (S_1, S_2) cuts added	21	76	193

Table 5.5: Results on Case A and Case B instances of Tavares.

	rcap	ub(Tav)	Opt	CPU time	#frac cut	$\#(S_1, S_2)$
Case A	120	27*	27	0.04	0	0
	100	27*	27	0.06	0	0
	80	27*	27	0.23	1	7
	75	29	27	0.25	27	11
	70	29	27	0.29	19	17
	65	30	28	0.57	8	15
Case B	240	21*	21	0.09	4	1
	200	21*	21	0.11	0	0
	180	24	21	0.06	0	0
	120	27	25	14.46	0	93

Table 5.6: Results on PSPLIB instances

	#inst.	avg. lb/ub	min lb/ub	nodes	CPU time	#flow cut	#frac cut	$\#(S_1, S_2)$ cut
$j30^-$	413	1.00	1.00	60.95	1.46	6.60	3.36	15.84
$j30^+$	67	0.95	0.72	2613	442	72	11	554
$j60^-$	380	1.00	1.00	2.7	2.07	2.00	1.35	12.92
$j60^+$	100/86	0.93	0.67	591	2118	30	43	903

5.2 Numerical results for Section 2.2

Table 5.7: Average ub/lb values for algorithm B^+ (first row), and B^- (second row), respectively, in each class (n, r, s).

	$n = 10$			$n = 20$			$n = 50$		
s	$r = 3$	10	20	3	10	20	3	10	20
2	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1
15	1	1	1	1	1	1	1	1.02	1.06
	1	1	1	1	1	1	1.02	1.03	1.07
20	1	1	1	1	1	1.01	1.01	1.37	1.17
	1	1	1	1	1.01	1.01	1.01	1.5	1.16

Table 5.8: Average ub^+/ub^- (first row), and lb^+/lb^- (second row).

	$n = 10$			$n = 20$			$n = 50$		
s	$r = 3$	10	20	3	10	20	3	10	20
2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
5	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
10	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
15	1.000	1.000	1.000	1.000	1.000	1.000	0.981	0.999	0.994
	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.001	1.001
20	1.000	1.000	1.000	1.000	0.996	1.000	0.998	0.952	1.010
	1.000	1.000	1.000	1.000	0.999	0.999	1.003	1.013	0.998

Table 5.9: Averages of CPU time, search-tree nodes, Flow Cover, Gomory's fractional and (S_1, S_2) cuts for algorithm B^+.

	$n = 10$			$n = 20$			$n = 50$	
s	$r = 3$	10	20	$r = 3$	10	20	10	20
	0.034	0.028	0.017	0.023	0.025	0.027	0.055	0.067
	0.100	0.500	0.200	0.300	0.300	0.500	0.900	1.100
2	0.200	1.300	0.800	0.700	0.800	1.600	2.100	4.700
	0.900	1.000	1.000	1.700	2.400	2.100	4.300	6.400
	0.200	0.200	0.000	0.900	0.600	0.400	1.300	1.600
	0.039	0.048	0.049	0.108	0.164	0.128	0.856	1.201
	0.800	2.600	1.700	2.600	8.900	8.000	27.100	63.300
5	1.500	1.900	3.900	2.800	7.500	4.800	8.900	18.800
	3.000	3.900	7.000	6.500	9.700	10.200	12.300	18.400
	1.600	1.500	1.300	5.700	6.600	3.900	12.400	14.100
	0.156	0.247	0.211	0.430	2.222	1.561	34.712	171.150
	5.500	15.400	17.100	7.100	106.100	139.800	395.300	1244.900
10	2.600	7.300	8.300	2.100	18.400	19.200	24.100	50.500
	8.700	7.400	12.100	5.000	16.100	20.500	17.500	25.300
	6.500	7.100	5.300	15.900	28.800	19.900	84.400	161.800
	0.853	0.944	1.085	1.886	14.456	51.381	389.870	420.067
	39.900	56.600	96.900	12.700	409.300	2247.200	535.500	511.000
15	5.200	13.800	12.300	4.800	22.600	44.200	42.300	75.400
	3.600	12.400	15.600	8.500	16.100	29.600	14.100	24.800
	26.700	17.100	15.700	22.700	99.200	119.400	281.900	387.700
	1.386	4.328	2.011	11.133	97.577	212.791	420.080	420.113
	30.900	175.600	115.400	104.600	727.200	2734.300	202.857	164.375
20	7.200	22.000	15.200	6.700	39.700	50.000	54.429	85.875
	6.400	17.200	19.200	5.700	15.400	33.000	17.286	29.000
	22.400	34.400	31.400	60.400	176.400	215.000	288.143	312.000

Table 5.10: Average CPU time (first row) and search-tree nodes (second row) for algorithm B^-.

	$n = 10$			$n = 20$			$n = 50$		
s	$r = 3$	10	20	$r = 3$	10	20	$r = 3$	10	20
2	0.03	0.03	0.02	0.02	0.02	0.02	0.03	0.04	0.05
	0.10	0.50	0.20	0.60	0.40	0.20	0.40	0.50	0.70
5	0.03	0.04	0.05	0.08	0.16	0.11	0.19	0.84	1.05
	1.00	1.50	2.40	4.30	11.80	6.30	5.60	31.90	65.20
10	0.13	0.21	0.20	0.44	1.90	1.77	9.65	37.52	164.69
	5.40	16.60	14.20	11.90	108.50	191.50	144.70	474.10	1358.50
15	0.64	0.84	0.96	1.59	13.85	61.81	90.78	384.43	420.07
	39.00	65.40	99.80	16.00	547.10	3286.00	154.00	776.10	638.70
20	1.24	3.49	1.86	12.88	95.34	207.31	201.76	420.10	420.09
	32.40	176.00	104.60	146.80	921.90	4130.70	316.10	177.00	168.90

5.3 The $\alpha|\beta|\gamma$ notation of scheduling problems

The $\alpha|\beta|\gamma$ notation for succintly denote scheduling problems has been proposed by Graham et al. [40].

- The α field indicates the machining environment. Some of the possible values:
  - 1 : single machine
  - P : parallel identical machines
  - Pm : constant m parallel identical machines
- The β field contains the set of restrictions. Some of the restrictions:
  - r_j : job release dates
  - $p_j = 1$: processing times are uniformly 1
  - ddc : jobs are dedicated to machines
  - $pmtn$: jobs can be interrupted and resumed processing later on the same or on a different machine
  - res : additional renewable resources
  - nr : additional non-renewable resources
  - $nr = const$: constant number of non-renewable resources
  - $q = const$: constant number of supplies
  - dm : delivery of intermediate products
  - $dm = const$: constant number of product types to deliver
- The γ field holds the objective function. Some objective functions:
  - $C_{\max} := \max C_j$: maximum job completion time, or makespan
  - $L_{\max} := \max L_j$, where $L_j = C_j - d_j$: maximum lateness
  - $\sum w_j C_j$: total weighted job completion times

5.4 Approximation preserving reductions

In this section we recapitulate the basic definitions of approximation preserving reductions between pairs of optimization problems, and in particular we provide formal definitions of the Strict-, the PTAS-, and FPTAS-reductions. Our discussion closely follows [23] and [22], see also [4] and [87].

Formally, a *reduction* is a pair of functions f and g, where f maps the instances of optimization problem Π_1 to that of optimization problem Π_2, and g provides a feasible solution for instance I_1 of problem Π_1 from a feasible solution y for the corresponding instance $f(I_1)$ of Π_2. The following diagram illustrates the functions f and g:

$$\begin{array}{lccc} \text{Problems:} & \Pi_1 & & \Pi_2 \\ \text{Instances:} & I_1 & \longrightarrow_f & f(I_1) \\ & & & \downarrow \\ \text{Solutions:} & g(I_1, y) & {}_g\longleftarrow & y \end{array}$$

(f, g) is a *Strict-reduction* from problem Π_1 to problem Π_2 ($\Pi_1 \leq_{Strict} \Pi_2$) if f and g are computable in polynomial time in the size of their parameters, and for every instance I_1 of Π_1, and for every solution y to $f(I_1)$ we have

$$R_{\Pi_1}(I_1, g(I_1, y)) \leq R_{\Pi_2}(f(I_1), y).$$

A reduction (f, g) is a *PTAS-reduction* from problem Π_1 to problem Π_2 ($\Pi_1 \leq_{PTAS} \Pi_2$) if there exists a function $\alpha(\cdot)$ such that

i) for any instance I_1 of Π_1, and for any $\varepsilon > 0$, $f(I_1, \varepsilon)$ is an instance of Π_2 and it is computable in $t_f(|I_1|, \varepsilon)$ time,

ii) for any solution y of $f(I_1, \varepsilon)$, $g(I_1, y, \varepsilon)$ is a solution to I_1, and it is computable in $t_g(|I_1|, |y|, \varepsilon)$ time,

iii) for every fixed $\varepsilon > 0$, both $t_f(\cdot, \varepsilon)$ and $t_g(\cdot, \cdot, \varepsilon)$ are bounded by a polynomial, and

iv) α maps error parameters for problem Π_1 to that for problem Π_2 such that for every solution y to $f(I_1, \varepsilon)$:

$$R_{\Pi_2}(f(I_1, \varepsilon), y) \leq 1 + \alpha(\varepsilon) \text{ implies } R_{\Pi_1}(I_1, g(I_1, y, \varepsilon)) \leq 1 + \varepsilon. \quad (5.1)$$

The following statement is from [23].

Lemma 24. *Let Π_1 and Π_2 be optimization problems such that $\Pi_1 \leq_{PTAS} \Pi_2$. If Π_2 admits a PTAS, then there is a PTAS for Π_1 as well.*

The following lemma shows the connection between the Strict-reduction and the PTAS-reduction (for a proof see [22]):

Lemma 25. *Every Strict-reduction is a PTAS-reduction as well.*

Therefore, Lemma 24 remains valid if we replace the PTAS-reduction by Strict-reduction in the statement. Finally, an *FPTAS-reduction* is like a PTAS-reduction with the following modifications:

iii') Both $t_f(\cdot, \varepsilon)$ and $t_g(\cdot, \cdot, \varepsilon)$ must be bounded by a polynomial in $1/\varepsilon$ as well.

iv') α maps *instances* and error parameters for problem Π_1 to error parameters for Π_2 such that for every solution y to $f(I_1, \varepsilon)$:

$$R_{\Pi_2}(f(I_1, \varepsilon), y) \leq 1 + \alpha(I_1, \varepsilon) \text{ implies } R_{\Pi_1}(I_1, g(I_1, y, \varepsilon)) \leq 1 + \varepsilon. \quad (5.2)$$

That is, (5.2) replaces (5.1) in the definition of FPTAS.

v') α can be computed in polynomial time in $|I_1|$ and $1/\varepsilon$.

vi') There exists a two-variable polynomial $poly(\cdot, \cdot)$ such that $1/\alpha(I_1, \varepsilon) \leq poly(|I_1|, 1/\varepsilon)$ for any $\varepsilon > 0$.

Remark 4. *In the above definition, ε may be restricted $0 < \varepsilon \leq c$, where c is a positive constant, since we usually want to choose ε arbitrarily close to 0.*

In [23] the following statement was proved:

Lemma 26. *If there is an FPTAS-reduction (f, g) from problem Π_1 to problem Π_2, and if Π_2 admits an FPTAS, then there is an FPTAS for Π_1 as well.*

Observe that an FPTAS-reduction is not a PTAS-reduction in general. To see this, suppose we have a pair of optimization problems Π_1 and Π_2, and there is an FPTAS-reduction from Π_1 to Π_2 with $\alpha(I_1, \varepsilon) := \varepsilon/n$, where n is the number of some objects in I_1, and the n objects in I_1 are mapped to n objects in $f(I_1, \varepsilon)$. Moreover, suppose we have a PTAS for Π_2 of running time $O(n^{1/\omega})$, where ω is the desired error ratio. Now, the running time of the PTAS on instance $f(I_1, \varepsilon)$ with error parameter $\omega := \alpha(I_1, \varepsilon)$ is $O(n^{1/\alpha(I_1, \varepsilon)}) = O(n^{n/\varepsilon})$, which is not polynomial in n. Clearly, a PTAS-reduction is not an FPTAS-reduction in general, since the time complexity of computing f and g is not required to be bounded by a polynomial in $1/\varepsilon$, cf. condition iii) of the PTAS-reduction.

As in the case of PTAS reductions, one can show the following:

Lemma 27. *Every Strict-reduction is an FPTAS-reduction as well.*

The next lemma follows from [23] and [87]:

Lemma 28. *The defined reductions are transitive.*

References

[1] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows: theory, algorithms, and applications. 1993.

[2] Arianna Alfieri, Tullio Tolio, and Marcello Urgo. A project scheduling approach to production planning with feeding precedence relations. *International Journal of Production Research*, 49(4):995–1020, 2011.

[3] Alper Atamtürk and Muhong Zhang. The flow set with partial order. *Mathematics of Operations Research*, 33(3):730–746, 2008.

[4] Giorgio Ausiello, Pierluigi Crescenzi, and Marco Protasi. Approximate solution of np optimization problems. *Theoretical Computer Science*, 150(1):1–55, 1995.

[5] Francisco Ballestín, Christoph Schwindt, and Jürgen Zimmermann. Resource leveling in make-to-order production: modeling and heuristic solution method. *International Journal of Operations Research*, 4(1):50–62, 2007.

[6] Lucio Bianco and Massimiliano Caramia. Minimizing the completion time of a project under resource constraints and feeding precedence relations: a lagrangian relaxation based lower bound. *4OR*, 9(4):371–389, 2011.

[7] Jacek Błażewicz, Klaus H Ecker, Erwin Pesch, Günter Schmidt, and Jan Weglarz. *Handbook on scheduling: from theory to applications.* Springer Science & Business Media, 2007.

[8] Jacek Błażewicz, Klaus H Ecker, Erwin Pesch, Günter Schmidt, and Jan Weglarz. *Scheduling computer and manufacturing processes.* Springer Science & Business Media, 2013.

[9] Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[10] Nils Boysen, Stefan Bock, and Malte Fliedner. Scheduling of inventory releasing jobs to satisfy time-varying demand: an analysis of complexity. *Journal of Scheduling*, 16(2):185–198, 2013.

[11] Dirk Briskorn, Byung-Cheon Choi, Kangbok Lee, Joseph Leung, and Michael Pinedo. Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, 207(2):605–619, 2010.

[12] Dirk Briskorn, Florian Jaehn, and Erwin Pesch. Exact algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, 16:105–115, 2013.

[13] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag Berlin Heidelberg, 2007.

[14] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European journal of operational research*, 112(1):3–41, 1999.

[15] AR Burgess and James B Killebrew. Variation in activity level on a cyclical arrow diagram. *Journal of Industrial Engineering*, 13(2):76–83, 1962.

[16] Alberto Caprara, Hans Kellerer, Ulrich Pferschy, and David Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123(2):333–345, 2000.

[17] Massimiliano Caramia and Paolo Dell'Olmo. Assessing the resource usage in scheduling with incompatibilities. *OR Spectrum*, 25(4):521–547, 2003.

[18] J Carlier and AHG Rinnooy Kan. Scheduling subject to nonrenewable-resource constraints. *Operations Research Letters*, 1(2):52–55, 1982.

[19] Jacques Carlier. *Problèmes d'ordonnancement à contraintes de ressources: algorithmes et complexité*. Université Paris VI-Pierre et Marie Curie, Institut de programmation, 1984.

[20] Bo Chen, Chris N Potts, and Gerhard J Wöginger. A review of machine scheduling: Complexity, algorithms and approximability. In *Handbook of combinatorial optimization*, pages 1493–1641. Springer, 1998.

[21] J Coelho. Personal communication, 2003.

[22] Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *Computational Complexity, 1997. Proceedings., Twelfth Annual IEEE Conference on (Formerly: Structure in Complexity Theory Conference)*, pages 262–273. IEEE, 1997.

[23] Pierluigi Crescenzi and Alessandro Panconesi. Completeness in approximation classes. *Information and Computation*, 93(2):241–262, 1991.

[24] Ronald De Boer. *Resource-constrained multi-project management*. PhD thesis, University of Twente, The Netherlands, 1998.

[25] Erik L Demeulemeester and Willy S Herroelen. *Project scheduling: a research handbook*, volume 49. Springer Science & Business Media, 2002.

[26] Stephan Dempe. *Foundations of bilevel programming*. Springer Science & Business Media, 2002.

[27] Stephan Dempe. Annotated bibliography on bilevel programming and mathematical programs with equilibrium constraints. *Optimization*, 52:333–359, 2003.

[28] Márton Drótos, Gábor Erdős, and Tamás Kis. Computing lower and upper bounds for a large-scale industrial job shop scheduling problem. *European Journal of Operational Research*, 197(1):296–306, 2009.

[29] Márton Drótos and Tamás Kis. Resource leveling in a machine environment. *European Journal of Operational Research*, 212(1):12–21, 2011.

[30] Márton Drótos and Tamás Kis. Scheduling of inventory releasing jobs to minimize a regular objective function of delivery times. *Journal of Scheduling*, 16(3):337–346, 2013.

[31] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

[32] Bin Fu, Yumei Huo, and Hairong Zhao. Coordinated scheduling of production and delivery with production window and delivery capacity constraints. *Theoretical Computer Science*, 422:39–51, 2012.

[33] János Fülöp. On the equivalence between a linear bilevel programming problem and linear optimization over the efficient set. Technical report, Laboratory of Operational Research and Decision Systems, Computer and Automation Research Institute, Hungarian Academy of Science, Budapest, 1993.

[34] Péter Gács and László Lovász. Khachiyan's algorithm for linear programming. In *Mathematical Programming at Oberwolfach*, pages 61–68. Springer, 1981.

[35] Evgeny R Gafarov, Alexander A Lazarev, and Frank Werner. Single machine scheduling problems with financial resource constraints: Some complexity results and properties. *Mathematical Social Sciences*, 62(1):7–13, 2011.

[36] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[37] Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129, 1976.

[38] Georgii V Gens and Eugenii V Levner. Computational complexity of approximation algorithms for combinatorial problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 292–300. Springer, 1979.

[39] Teofilo Gonzalez and Sartaj Sahni. Flowshop and jobshop schedules: complexity and approximation. *Operations research*, 26(1):36–52, 1978.

[40] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

[41] Alexander Grigoriev, Martijn Holthuijsen, and Joris van de Klundert. Basic scheduling problems with raw material constraints. *Naval Research Logistics (NRL)*, 52(6):527–535, 2005.

[42] Péter Györgyi and Tamás Kis. Approximation schemes for single machine scheduling with non-renewable resource constraints. *Journal of Scheduling*, 17:135–144, 2014.

[43] Péter Györgyi and Tamás Kis. Reductions between scheduling problems with non-renewable resources and knapsack problems. *Theoretical Computer Science*, 565:63–76, 2015a.

[44] Péter Györgyi and Tamás Kis. Approximability of scheduling problems with resource consuming jobs. *Annals of Operations Research*, 235(1):319–336, 2015b.

[45] Péter Györgyi and Tamás Kis. Approximation schemes for parallel machine scheduling with non-renewable resources. *European Journal of Operational Research*, 258(1):113–123, 2017.

[46] Leslie A Hall and David B Shmoys. Approximation schemes for constrained scheduling problems. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 134–139. IEEE, 1989.

[47] Elias Willem Hans. *Resource loading by branch-and-price techniques*. PhD thesis, 2001.

[48] Elias Willem Hans. Personal communication, 2003.

[49] Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 207(1):1–14, 2010.

[50] Willy Herroelen, Bert De Reyck, and Erik Demeulemeester. Resource-constrained project scheduling: a survey of recent developments. *Computers & Operations Research*, 25(4):279–302, 1998.

[51] A. J. Hoffman and J. B. Kruskal. Integral boundary points of convex polyhedra. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 233–246. Princeton University Press, Princeton, 1956.

[52] Han Hoogeveen. Multicriteria scheduling. *European Journal of operational research*, 167(3):592–623, 2005.

[53] Oscar H Ibarra and Chul E Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM (JACM)*, 22(4):463–468, 1975.

[54] ILOG. *ILOG CPLEX 7.5, User's manual*. ILOG S.A, Gentilly, France, 2001.

[55] Michael Jünger, Gerhard Reinelt, and Stefan Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. *Combinatorial Optimization, Dimacs*, 20:111–152, 1995.

[56] AHG Rinnooy Kan. *Machine scheduling problems: classification, complexity and computations*. Springer Science & Business Media, 2012.

[57] John K Karlof and Wei Wang. Bilevel programming applied to the flow shop scheduling problem. *Computers & operations research*, 23(5):443–451, 1996.

[58] Richard M Karp. Reducibility among combinatorial problems. In Thatcher JW Miller RE, editor, *Complexity of computer computations*, pages 85–103. Plenum Press, 1972.

[59] Hans Kellerer, Vladimir Kotov, Franz Rendl, and Gerhard J Wöginger. The stock size problem. *Operations Research*, 46(3-supplement-3):S1–S12, 1998.

[60] Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an fptas for the knapsack problem. *Journal of Combinatorial Optimization*, 8(1):5–11, 2004.

[61] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

[62] Hans Kellerer and Vitaly A Strusevich. Scheduling parallel dedicated machines under a single non-shared resource. *European Journal of Operational Research*, 147(2):345–364, 2003.

[63] Hans Kellerer and Vitaly A Strusevich. Scheduling problems for parallel dedicated machines under multiple resource constraints. *Discrete Applied Mathematics*, 133(1):45–68, 2003.

[64] Tamás Kis. A branch-and-cut algorithm for scheduling of projects with variable-intensity activities. *Mathematical programming*, 103(3):515–539, 2005.

[65] Tamás Kis. Rcps with variable intensity activities and feeding precedence constraints. In *Perspectives in modern project scheduling*, pages 105–129. Springer, 2006.

[66] Tamás Kis. Approximability of total weighted completion time with resource consuming jobs. *Operations Research Letters*, 43(6):595–598, 2015.

[67] Tamás Kis, G Erdős, András Márkus, and József Váncza. A project-oriented decision support system for production planning in make-to-order manufacturing. *ERCIM news*, (58):66–67, 2004.

[68] Tamás Kis and András Kovács. On bilevel machine scheduling problems. *OR spectrum*, 34(1):43–68, 2012.

[69] Tamás Kis and András Kovács. Exact solution approaches for bilevel lot-sizing. *European Journal of Operational Research*, 226(2):237–245, 2013.

[70] Rainer Kolisch and Arno Sprecher. Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. *European journal of operational research*, 96(1):205–216, 1997.

[71] Rainer Kolisch, Arno Sprecher, and Andreas Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management science*, 41(10):1693–1703, 1995.

[72] Stavros G Kolliopoulos and George Steiner. Partially ordered knapsack and applications to scheduling. *Discrete Applied Mathematics*, 155(8):889–897, 2007.

[73] András Kovács, Péter Egri, Tamás Kis, and József Váncza. Proterv-2: An integrated production planning and scheduling system. In *International Conference on Principles and Practice of Constraint Programming*, pages 880–880. Springer, 2005.

[74] András Kovács and Tamás Kis. Constraint programming approach to a bilevel scheduling problem. *Constraints*, 16(3):317–340, 2011.

[75] Eugene L Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management science*, 19(5):544–546, 1973.

[76] Eugene L Lawler. Knapsack-like scheduling problems, the moore-hodgson algorithm and the 'tower of sets' property. *Mathematical and Computer Modelling*, 20(2):91–106, 1994.

[77] Eugene L Lawler and J Michael Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969.

[78] Robert C Leachman, Abdurrezak Dincerler, and Sooyoung Kim. Resource-constrained scheduling of projects with variable-intensity activities. *IIE transactions*, 22(1):31–40, 1990.

[79] Zrinka Lukač, Kristina Šorić, and Višnja Vojvodić Rosenzweig. Production planning problem with sequence dependent setups as a bilevel programming problem. *European Journal of Operational Research*, 187(3):1504–1512, 2008.

[80] A Márkus, József Váncza, T Kis, and A Kovács. Project scheduling approach to production planning. *CIRP Annals-Manufacturing Technology*, 52(1):359–362, 2003.

[81] J Michael Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management science*, 15(1):102–109, 1968.

[82] Anulark Naber and Rainer Kolisch. Mip models for resource-constrained project scheduling with flexible resource profiles. *European Journal of Operational Research*, 239(2):335–348, 2014.

[83] George L Nemhauser and Laurence A Wolsey. *Integer and Combinatorial Optimization*. Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1988.

[84] Klaus Neumann and Christoph Schwindt. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56(3):513–533, 2002.

[85] Klaus Neumann and Jürgen Zimmermann. Resource levelling for projects with schedule-dependent time windows. *European Journal of Operational Research*, 117(3):591–605, 1999.

[86] Klaus Neumann and Jürgen Zimmermann. Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. *European Journal of Operational Research*, 127(2):425–443, 2000.

[87] Pekka Orponen and Heikki Mannila. On approximation preserving reductions: Complete problems and robust measures (revised version). *Department of Computer Science, University of Helsinki*, 1990.

[88] Manfred Padberg and Giovanni Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, 1987.

[89] Manfred W Padberg, Tony J Van Roy, and Laurence A Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33(4):842–861, 1985.

[90] Christos H Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

[91] Maurice Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58(1-3):263–285, 1993.

[92] Rafał Różycki, Grzegorz Waligóra, and Jan Węglarz. Scheduling preemptable jobs on identical processors under varying availability of an additional continuous resource. *International Journal of Applied Mathematics and Computer Science*, 26(3):693–706, 2016.

[93] Sartaj K Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM (JACM)*, 23(1):116–127, 1976.

[94] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, New York, 1986.

[95] Petra Schuurman and Gerhard Wöginger. Approximation schemes—a tutorial. In RH Möhring, CN Potts, AS Schulz, GJ Wöginger, and LA Wolsey, editors, *Lectures on Scheduling*. 2009.

[96] Roman Slowinski. Preemptive scheduling of independent jobs on parallel machines subject to financial constraints. *European Journal of Operational Research*, 15:366–373, 1984.

[97] Wayne E Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[98] Chi-Shiang Su, Jason Chao-Hsien Pan, and Tsung-Shin Hsu. A new heuristic algorithm for the machine scheduling problem with job delivery coordination. *Theoretical Computer Science*, 410(27):2581–2591, 2009.

[99] Luís Valadares Tavares. *Advanced models for project management*, volume 16. Springer Science & Business Media, 1998.

[100] Luís Valadares Tavares. A review of the contribution of operational research to project management. *European Journal of Operational Research*, 136(1):1–18, 2002.

[101] Aysegül Toker, Suna Kondakci, and Nesim Erkip. Scheduling under a non-renewable resource constraint. *Journal of the Operational Research Society*, 42(9):811–814, 1991.

[102] Vicente Valls, Ángeles Pérez, and Sacramento Quintanilla. Skilled workforce scheduling in service centres. *European Journal of Operational Research*, 193(3):791–804, 2009.

[103] József Váncza, T Kis, and A Kovács. Aggregation-the key to integrating production planning and scheduling. *CIRP Annals-Manufacturing Technology*, 53(1):377–380, 2004.

[104] Jan Węglarz. Project scheduling with continuously-divisible, doubly constrained resources. *Management Science*, 27(9):1040–1053, 1981.

[105] Jan Węglarz, Joanna Józefowska, Marek Mika, and Grzegorz Waligóra. Project scheduling with finite or infinite number of activity processing modes–a survey. *European Journal of Operational Research*, 208(3):177–205, 2011.

[106] Bruce M Woodworth and Charles J Willie. A heuristic algorithm for resource leveling in multi-project, multi-resource scheduling. *Decision Sciences*, 6(3):525–540, 1975.