# SCHEDULING WITH NON-RENEWABLE RESOURCES


BY

Péter Györgyi

Supervisor: Tamás Kis

Operations Research Department

Eötvös Loránd University



Budapest, 2013.

# Contents

# Acknowledgement

I would like to thank my supervisor, Tamás Kis, for his guidance. He always had time for me and I am grateful for the constant help, for all the suggestions and corrections.

# Notations

| | |
|---|---|
| $\mathcal{J}$ | set of the jobs $\{J_1, J_2, \ldots, J_n\}$ |
| $n$ | $\lvert\mathcal{J}\rvert$ |
| $p_j$ | processing time of job $J_j$ |
| $s_j$ | starting time of job $J_j$ (for a given schedule) |
| $C_j$ | completion time of job $J_j$ (for a given schedule) |
| $\mathcal{R}$ | set of resources |
| $q$ | number of the time moments when a resource is supplied |
| $u_1, u_2, \ldots, u_q$ | time moments when positive amount of resource is supplied ($u_{q+1} := \infty$) |
| $b_i$ | amount of the supplied resource at $u_i$ (in case of one resource) |
| $b_{k,i}$ | amount of the supplied resource at $u_i$ from resource $k$ |
| $a_j$ | required resource for job $J_j$ (in case of one resource) |
| $a_{k,j}$ | required resource for job $J_j$ from resource $k$ |
| $r_j$ | release date of job $J_j$ |
| $d_j$ | deadline of job $J_j$ |
| $C_{max}^{Alg}$ | makespan of algorithm $Alg$ |
| $C_{max}^{*}$ | the optimal makespan |
| $L_{max}^{Alg}$ | maximum lateness of algorithm $Alg$ |
| $L_{max}^{*}$ | the optimal maximum lateness |

# Chapter 1

# Introduction

## 1.1 Overview

This thesis is a survey (with new results) on scheduling problems where non-renewable resource constraints arise. In these problems there is a given initial stock level from some non-renewable resources (raw materials, money) and time moments when there is a replenishment with a known amount from a resource. We have a set of jobs that we want to schedule, the jobs have some resource requirements. This means that we can schedule a job only if there are enough quantities from the required raw materials when starting the job (and after scheduling a job, we have to decrease our stock by the applied quantities). Sometimes it is possible that after the completion of a job it produces given quantities of raw materials.

The presented models have great practical importance. Problems like the further arise in e. g. truck scheduling in a transshipment terminal or for construction companies. In the preceding the trucks bring or carry away a known amount of items. We can keep the items in the stock of the terminal, however if the stock is empty we cannot start a truck which needs some items. Construction companies do not want to (or cannot) begin a part of their project if they do not receive all necessary money or other resources for that part.

Scheduling with non-renewable resource constraints has been studied first in [2] and [3] in the early 80s. Kellerer et al. [10] analysed a special case of the truck transshipment problem, called the stock size problem. Briskorn et al. [1] gave a wide outline of the complexity results in the theme. Grigoriev et al. [8] dealt with problems where the jobs only consume the raw materials. [3] and [5] also studied similar problems. The main result of this thesis can be found in [9] as well.

Chapter 2 deals with the stock size problem. We introduce the problem in section 2.1, show the main result (a 3/2-approximation) of [10] in section 2.2, and present

an example in section 2.3* that proves the tightness of the approximation.

Chapter 3 studies problems where the jobs only consume raw materials. Section 3.1 based on [8] summarizes the previous results about the minimization of the maximal lateness. We extend these results in section 3.2*. The rest of the chapter deals with makespan minimization. Section 3.3 describes the previous results, in section 3.4* and 3.5* we present two Polynomial Time Approximation Schemes (PTAS) for the problems $1|rm = 1, q = const|C_{max}$ and $1|r_j, rm = 1, q = const|C_{max}$ (for notations see the next section).

Chapter 4 describes the results of [3] where we do not have machines, only precedence constraints, and deadlines.

The last chapter summarizes the results of this thesis and the mentioned articles. The sections with new results are marked with a *.

## 1.2 Notations

We use the well-known $\alpha|\beta|\gamma$ notation of Graham et al. [7]. This was extended by Grigoriev et al. [8] with the restrictions $rm$: $rm = m$ means that there are $m$ resources (raw materials). If we do not indicate the number of the raw materials then there can be arbitrary number of raw materials. Further on $\mathcal{R}$ will denote the set of the raw materials. Let $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ be the set of the jobs. There is a given processing time $p_j \geq 0$ for all $J_j \in \mathcal{J}$ and if there are due dates ($d_j$), release dates ($r_j$) then these are given. There may be a precedence relation $\Pi$ between the jobs: $(j, k) \in \Pi$ means that $J_j$ must finish before $J_k$ may start. We use some common notations like $s_j$ is the starting time, and $C_j$ is the completion time of $J_j$ for a given schedule. In most of the problems there is only one machine, but the problems considered in chapter 4 do not use any machine (we use the notation '$-$' for it).

Denote $q$ the number of time moments when positive amount of raw material is supplied. These moments are $u_1, u_2, \ldots, u_q$, and sometimes it is convenient to use $u_{q+1} = \infty$. We use some separate notations for the problems where there is only one raw material: let $b_i$ denote the amount of the supplied raw material at $u_i$, and we use $a_j$ for the amount of the raw material that $J_j$ requires. If there are at least two raw materials let $a_{k,j}$ be the required raw material for $J_j$ from raw material $k$ and let $b_{k,i}$ be the amount of the supplied resource at $u_i$ from resource $k$. We suppose that for each raw material the total amount delivered is equal to the total requirements

of the jobs, i. e.,

$$\sum_{i=1}^{q} b_{k,i} = \sum_{J_j \in \mathcal{J}} a_{k,j} \qquad \forall k \in \mathcal{R}.$$

# Chapter 2

# The stock size problem

## 2.1 The problem and the mathematical model

We have a warehouse and $n$ jobs that we have to schedule on a single machine. Every job consumes or supplies some raw material(s). If we completed a job the amount of the raw material which we keep in our warehouse is changing. Initially, the warehouse is empty. If a job requires some raw material we can only schedule it if there is enough amount of raw material in the warehouse. On the other hand, if a job produces some raw material we can schedule it only if there is enough space in the warehouse. The jobs consume the same amount of raw material that they produce in total. Our task is to find an ordering of the jobs which needs the smallest inventory level and does not hurt any of the constraints. There are many natural examples with similar characteristics.

Firstly, we introduce a new notation. Denote $z_j$ the amount of raw material with which $J_j$ increases (we call them x-jobs) or decreases (y-jobs) the stock size. In the first case $z_j > 0$, in the second $z_j < 0$ (thus $\sum_{j=1}^{n} z_j = 0$). Let $\pi$ denote a permutation of the jobs. If we formalize the constraints we get the following:

$$\sum_{j=1}^{k} z_{\pi(j)} \geq 0, \quad \forall k = 1, \ldots, n$$

We have to find a permutation $\pi$ which minimizes the maximum inventory level, i. e.,

$$\min_{\pi} \max_{k} \left\{ \sum_{j=1}^{k} z_{\pi(j)} \right\}$$

## 2.2 Previous results, the GPS heuristic

It is possible to show that our problem is NP-hard. [10] Kellerer et al. presented three approximation algorithms. The best one (GPS) ensures that stock size will be at most 3/2 times of the optimum stock size (OPT). At the end of their article they asked whether this bound is tight or the worst case of their algorithm is better. We will present the GPS algorithm in this section and in the next section we will give an example showing that the bound 3/2 is tight.

Let $C \geq \max |z_i|$ be a flexible bound. Divide the jobs into two classes according to $C$: if $|z_j| \geq C/2$ the $J_j$ is a big-job, if $|z_j| < C/2$ then $J_j$ is a small-job. We will create some pairs from the big-jobs: the $i^{th}$-largest x-jobs and the $i^{th}$-largest y-job will form pairs. The subroutine STOCK will pair and sequence the jobs. STOCK either finds an ordering where a $3C/2$-size warehouse is enough or shows that there is no sequence of jobs for which the maximum size of the warehouse is at most $C$. It is easy to create a 3/2-approximation using STOCK and binary search (see later). We will prove that STOCK is a 3/2-relaxed decision procedure, which means the following:

**Definition 2.2.1** ([10],$\rho$-relaxed decision procedure)**.** Input: $\mathcal{J}, C, p_j, z_j; j \in \mathcal{J}$; output: NO or ALMOST. If the output is ALMOST then the procedure gives a permutation $\pi$, where a $\rho C$-size warehouse is enough. If the output is NO, then the optimal warehouse is bigger than C.

If an x-job is a big-job, then we call it big-x-job. Similarly we use the small-x-job, big-y-job, small-y-job expressions. Let $A = \{a_1, \ldots, a_{|A|}\}$ be the set of big-x-jobs, $B = \{b_1, \ldots, b_{|B|}\}$ be the set of big-y-jobs, $V = \{v_1, \ldots, v_{|V|}\}$ be the set of small-x-jobs, $W = \{w_1, \ldots, w_{|W|}\}$ be the set of small-y-jobs. Let $p := \min\{|A|, |B|\}$, $A_p := \{a_1, \ldots, a_p\}$, $B := \{b_1, .., b_p\}$, $A' := A \backslash A_p$, $B' := B \backslash B_p$. We assume that the jobs to be sorted in a non-increasing order:

$$z_{a_1} \geq z_{a_2} \geq \ldots \geq z_{a_{|A|}} \geq C/2 > z_{v_1} \geq z_{v_2} \geq \ldots \geq z_{v_{|V|}}$$
$$-z_{b_1} \geq -z_{b_2} \geq \ldots \geq -z_{b_{|B|}} \geq C/2 > -z_{w_1} \geq -z_{w_2} \geq \ldots \geq -z_{w_{|W|}}$$

Let $\sum_V := \sum_{j \in V} z_j$, $\sum_W := \sum_{j \in W} |z_j|$ and $\sum_p := \sum_{j \in A_p \cup B_p} z_j$.

Some remarks for the better understanding of STOCK:
(1) If we sequence a job, we automatically remove it from the sets $A, B, \ldots$
(2) We always refer to the actual elements of the sets (the not scheduled jobs). In

that sense, $a_i$ always denotes the actually $i^{th}$-biggest element of $A$ ($b_i, v_i$ and $w_i$ have similar meanings).
(3) If we sequence more than one job at the same time, we sequence the x-jobs first (in an arbitrary order) and then the y-jobs.

**Procedure** $STOCK_{3/2}(C)$ [10]

*Step 0.* Initialize list $L$ as an empty list and the current stock size $S$ to zero.

*Step 1.* (Branching step)
**If** $A = \emptyset$ **goto** Step 8.
**else if** $B = \emptyset$ **goto** Step 9.
**else if** there is a pair $a_i, b_i$ ($a_i \in A, b_i \in B$) with $z_{a_i} + z_{b_i} < 0$ and $S + z_{a_i} + z_{b_i} \geq 0$
**goto** Step 2.
**else if** there is a pair $a_j, b_j$ ($a_j \in A, b_j \in B$) with $z_{a_j} + z_{b_j} \geq 0$ and $S + z_{a_j} \leq 3C/2$
**goto** Step 3.
**else if** $z_{a_j} + z_{b_j} \geq 0$ for all pairs $a_j, b_j$ ($a_j \in A, b_j \in B$)
**begin**
**If** $S + \sum_p - \sum_W > C/2$ **goto** Step 4.
**else goto** Step 5.
**end**
**else if** $z_{a_i} + z_{b_i} < 0$ for all pairs $a_i, b_i$ ($a_i \in A, b_i \in B$)
**begin**
**If** $S + \sum_p + \sum_V < 0$ **goto** Step 6.
**else goto** Step 7.
**end**

*Step 2.* Take the pair $a_{i_0}, b_{i_0}$ with smallest index $i_0$ such that $z_{a_{i_0}} + z_{b_{i_0}} < 0$ and $S + z_{a_{i_0}} + z_{b_{i_0}} \geq 0$ holds and append it to list $L$. Set $S := S + z_{a_{i_0}} + z_{b_{i_0}}$. **Goto** Step 1.

*Step 3.* Take the pair $a_{i_0}, b_{i_0}$ with smallest index $i_0$ such that $z_{a_{i_0}} + z_{b_{i_0}} \geq 0$ and $S + z_{a_{i_0}} \leq 3C/2$ holds and append it to list $L$. Set $S := S + z_{a_{i_0}} + z_{b_{i_0}}$. **Goto** Step 1.

*Step 4.* **While** $V \neq \emptyset$ and $S < -z_{b'_1}$
**begin**
Append $v_1$ to list $L$, $S := S + z_{v_1}$

**end**

**If** $S \geq z_{b'_1}$ append $b'_1$ to list $L$, $S := S + z_{b'_1}$. **goto** Step 1.

**else output** *NO*

*Step 5.* **While** $A \neq \emptyset$ and $B \neq \emptyset$

**begin**

**While** $S > C/2$

**begin**

Append $w_1$ to list $L$, $S := S + z_{w_1}$

**end**

Append $a_1$ and $b_1$ to list $L$, $S := S + z_{a_1} + z_{b_1}$

**end**

**If** $A = \emptyset$ **goto** Step 8.

**else goto** Step 9.

*Step 6.* Append $a'_1$ to list $L$, $S := S + z_{a'_1}$

**While** $W \neq \emptyset$ and $S > 3C/2 - z_{a_1}$

**begin**

Append $w_1$ to list $L$, $S := S + z_{w_1}$

**end**

**If** $S \leq 3C/2 - z_{a_1}$ append $a_1$ and $b_1$ to list $L$, $S := S + z_{a_1} + z_{b_1}$. **goto** Step 1.

**else output** *NO*

*Step 7.* **While** $A \neq \emptyset$ and $B \neq \emptyset$

**begin**

**While** $S + z_{a_1} + z_{b_1} < 0$

**begin**

Append $v_1$ to list $L$, $S := S + z_{v_1}$

**end**

Append $a_1$ and $b_1$ to list $L$, $S := S + z_{a_1} + z_{b_1}$

**end**

**If** $A = \emptyset$ **goto** Step 8.

**else goto** Step 9.

*Step 8.* **While** $V \neq \emptyset$

**begin**

**while** $S < C$

**begin**

Append $v_1$ to list $L$, $S := S + z_{v_1}$

**end**

Append the $b_1$ (if $B = \emptyset$ append $w_1$) to list $L$, $S := S + z_{b_1}$ (or $S := S + z_{w_1}$)

**end**

Append the remaining elements of $B \cup W$ to the list in arbitrary order. **Output** *ALMOST* **STOP**

*Step 9.* **While** $A \cup V \neq \emptyset$

**begin**

**while** $S > C/2$

**begin**

Append $w_1$ to list $L$, $S := S + z_{w_1}$

**end**

Append the $a_1$ (if $A = \emptyset$ append $v_1$) to list $L$, $S := S + z_{a_1}$ (or $S := S + z_{v_1}$)

**end**

Append the remaining elements of $W$ to the list in arbitrary order. **Output** *ALMOST* **STOP**.

**Theorem 2.2.2.** *([10]) Let* $C \geq \max |z_i|$. *Procedure STOCK is a* 3/2*-relaxed decision procedure.*

The theorem has a long technical proof, it can be found in [10].

We get heuristic GPS if we embed STOCK into a binary search. It is easy to see that $\mu := \max |z_i|$ is a lower bound and $2\mu = 2 \max |z_i|$ is an upper bound for the stock size.

**Heuristic GPS** ([10])

*Step 1.* Sort the x-jobs and the y-jobs into the following sequences:

$$z_{x_1} \geq z_{x_2} \geq \dots$$
$$-z_{y_1} \geq -z_{y_2} \geq \dots$$

We choose a precision $\varepsilon \geq 0$.

*Step 2.* Apply binary search in the $[\mu, 2\mu]$ interval by calling STOCK. If the output

of STOCK(C) is *NO* then we have found a new lower bound, in the other case we have found a new upper bound and a schedule. We finish the binary search when the difference between the bounds is less than $\varepsilon\mu$.

If we use last theorem it is easy to see the following:

**Theorem 2.2.3.** *([10]) For every $\varepsilon \geq 0$, heuristic GPS yields a stock sequence with the maximum stock size $S^{GPS}$ less than $3(1+\varepsilon)/2$ times the optimum stock size $S^*$. It can be implemented to run in $O(n \log n \log(1/\varepsilon))$ time.*

## 2.3* The tightness of the GPS algorithm

In this section we will show an instance of the stock size problem where the bound of 3/2 is tight (see *Figure 1.*). [10] mentioned that the problem of the tightness is interesting, but they couldn't find an instance for it.

Let

$$n = 6$$
$$z_1 = z_2 = 10$$
$$z_3 = z_4 = z_5 = z_6 = -5.$$

It is easy to see that the optimal stock size is 10: if we schedule in the order $J_1, J_3, J_4, J_2, J_5, J_6$, we get an optimal schedule.

Let us see what we get if we use the GPS heuristic. The binary search starts with $C = 10$ since $\mu = 10$. Every job in our instance is a big job ($C/2 = 5 \leq |z_j|, j \in \mathcal{J}$).

If we run $STOCK_{3/2}(10)$, after *Step 1.* we have to go to *Step 3.*, because $A, B \neq \emptyset$ and there is no pair $(a_i, b_i)$ with $z_{a_i} + z_{b_i} < 0$. At *Step 3.* we schedule $J_1$ and $J_3$ ($S = 0 + 10 - 5 = 5$) and we go back to *Step 1.*.

*Step 1.* sends us to *Step 3.* again ($A, B \neq \emptyset$, no pair $(a_i, b_i)$ with $z_{a_i} + z_{b_i} < 0$), where we schedule $J_2$ and $J_4$ ($S = 5 + 10 - 5 = 10$). After scheduling $J_2$ our inventory level is 15, which is $\frac{3}{2} \cdot 10$. At the end of this step we go back to *Step 1.* again.

Now we have only $J_5$ and $J_6$ to schedule. Both of them is an y-job so $A = \emptyset$. Hence, we have to go to *Step 8.*, where we schedule $J_5$ and $J_6$ in an arbitrary order. The output of $STOCK_{3/2}(10)$ is ALMOST, and it finishes here.

We found a good schedule at the lower border of the binary search, so we can finish here (since $\mu = 10$ is a lower bound then a schedule with a maximum inventory level 15 must be a good schedule). We have seen there is a schedule where the inventory level is at most 10, GPS provides a schedule where the maximum of the inventory level is 15. This means that the bound 3/2 is tight for GPS.

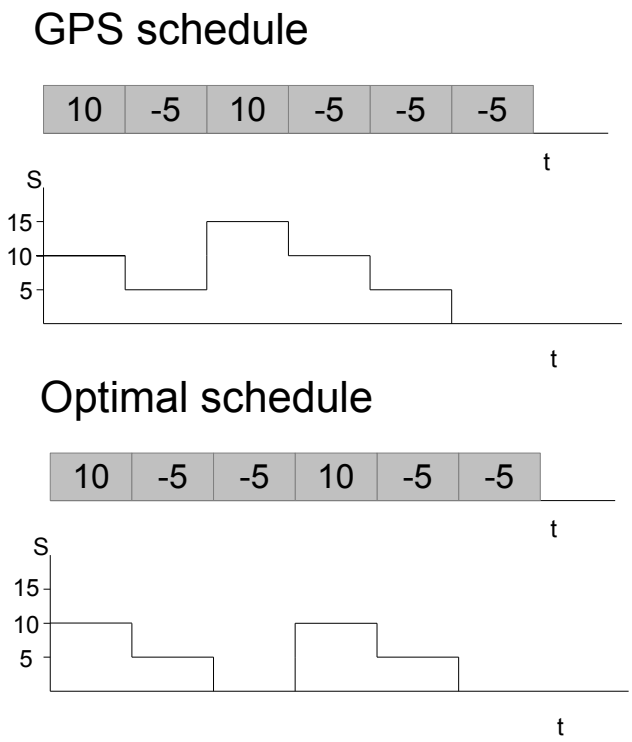


*Figure 1. A 'tight' instance of the stock size problem*

# Chapter 3

# Scheduling with raw materials

This chapter is dealing with the different subproblems of $1|rm|L_{max}$ and $1|rm|C_{max}$. In sections 3.1 and 3.3 we summarize the results of [8]. They proved that $1|rm = 2,\ p_j = 1|L_{max}$ is NP-hard. They also gave four 2-approximation algorithms for the problem $1|rm = 2, p_j = p|L_{max}$ (see section 3.1). In section 3.2 we generalize their results: we prove that these 2-approximation algorithms are also good for the problem $1|rm|L_{max}$. Sections 3.3, 3.4 and 3.5 are about the makespan minimization. First (as we mentioned) we show the results of [8] in section 3.3. In the next section we give a polynomial time approximation scheme for the problem $1|rm = 1|C_{max}$ if the number of the deliveries ($q$) is a constant. A modified version of this algorithm can treat the case when we have release time constraints, it can be found in section 3.5.

## 3.1 Minimizing $L_{max}$ if $p_j = p$

As usual we assume that all due dates are negative (otherwise we cannot give sensible estimations for the approximations) and the total delivered raw materials are exactly sufficient for the jobs. Recall that $u_q$ is the time moment when the last delivery arrives (thus $u_q \leq C^*_{max}$). Suppose that the due dates of the jobs satisfy $d_1 \leq d_2 \leq \ldots \leq d_n \leq 0$.

**Theorem 3.1.1.** *([8])* $1|rm = 1, p_j = 1|L_{max}$ *can be solved in polynomial time.*

*Proof.* If every job has unique due date, it is easy to see that the EDD order is optimal.

If two jobs have the same due date we can notice the following:

OBSERVATION 1: Let $d_k = d_{k+1}$ and $a_k \leq a_{k+1}$. Let $S$ and $S'$ be schedules where

the only difference is the position of $J_k$ and $J_{k+1}$, i. e.

$$\begin{aligned} s_k &= s'_{k+1} \\ s_{k+1} &= s'_k \\ s_j &= s'_j, \ \ \forall j \notin \{k, k+1\}. \end{aligned}$$

If $s_k > s_{k+1}$ and $S$ is feasible, then $S'$ is also feasible.

The next observation generalize the previous.

OBSERVATION 2: Let $I$ is an instance of the problem and let $d_k = d_{k+1}$, $a_k \leq a_{k+1}$. Let $I'$ be an instance of the problem, which has only one difference from $I$: $d'_k = d_k - 1$. Then any optimal solution for $I'$ is optimal for $I$ as well. Moreover, these instances have the same optimal solutions.

Suppose that if $d_j = d_{j+1}$ then $a_j \leq a_{j+1}$. The next algorithm constructs an instance where every job has a unique due date:

1. Set $j = n - 1$. While $j > 1$ do:
2. If $d_j = d_{j+1}$ set $d_j := d_j - 1$. Reconstruct the $d_1 \leq d_2, \ldots, d_n$ order (if $d_j = d_{j+1}$ then $a_j \leq a_{j+1}$).
3. If $d_j \neq d_{j+1}$, set $j := j - 1$.

With this algorithm we can construct an easily solvable instance from every instance. According to Observation 2 an optimal solution of the new instance is an optimal solution for the original. □

**Theorem 3.1.2.** *([8]) The problem $1|rm = 2, p_j = 1|L_{max}$ is strongly NP-hard.*

*Proof.* It can be found in [8]. It reduces the problem 3-PARTITION (known strongly NP-hard problem) to $1|rm = 2, p_j = 1|L_{max}$. □

**Definition 3.1.3** (active schedule)**.** A schedule is *active* if we cannot schedule any of the jobs earlier (if the order of the jobs is given).

**Definition 3.1.4** (active algorithm)**.** An algorithm is *active* if every output of the algorithm is an active schedule.

**Theorem 3.1.5.** *([8]) Any active approximation algorithm for $1|rm = 2, p_j = p|L_{max}$ has a worst case ratio of at most 3.*

*Proof.* There is a proof in [8], but we will give a proof for a more general theorem 3.2.1 in the next section. □

Now we define four approximation algorithms. All of them are an EDD-based algorithm and all of them are defined in [8]:

1. **Strict EDD:** Schedule the jobs in EDD order, all of them as early as possible.
2. **Lazy EDD:** Wait until all raw materials have arrived ($u_q$), and schedule them in EDD order without any idle time.
3. **Early EDD:** Starting at time moment $t = 0$, consider the set of jobs for which the required raw material have arrived, and select from this set the one with earliest due date, set $t \to t + 1$ and continue.
4. **First Fit EDD:** Take the jobs in EDD order, and schedule the first one from the list at the earliest possible time moment (when there is enough raw material for the job). Delete this job from the list, the raw material which it consumed and repeat.

*Remark* 3.1.6. In the next section we will make a small modification on algorithm Early EDD, because there was a small problem with it in the proof of the next theorem (see Remark 3.2.2).

**Example:** *Figure 2.* shows the difference among the four algorithms:

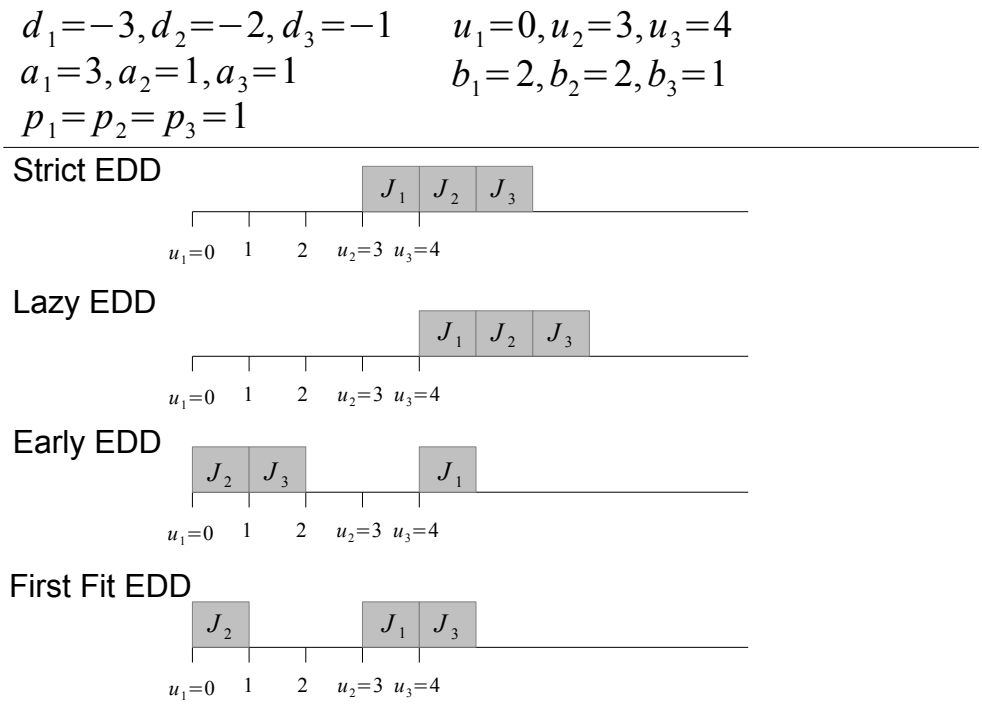


*Figure 2. The four EDD-based algorithms*

**Theorem 3.1.7.** *([8]) All four EDD-based algorithms have a worst case ratio of 2 for the problem* $1|rm = 2, p_j = p|L_{max}$.

*Proof.* It is easy to see that Strict EDD is always better than Lazy EDD, and First Fit EDD is better than Strict EDD (some jobs may starts earlier). The proof in [8] says Early EDD is always better than Lazy EDD, but this is not true (see Remark 3.2.2). In the next section we modify Early EDD, and we prove that the previous claim is correct for the modified Early EDD. This means that it is enough to prove the theorem for Lazy EDD, the other three must have a better maximum lateness. The rest of the proof can be found in [8], but we will prove a more general theorem (Theorem 3.2.3) in the next section. □

## 3.2* A generalization

Consider the problem $1|rm|L_{max}$. We will prove that the theorems of the previous section can be modified for our problem.

**Theorem 3.2.1.** *Any active approximation algorithm for* $1|rm|L_{max}$ *has a worst case ratio of at most 3.*

*Proof.* Let us verify the following bounds:

1. $L^*_{max} \geq p_1 - d_1$, because this is the minimum lateness of the first job ($d_1 \leq d_2 \leq \ldots \leq d_n < 0$).

2. $L^*_{max} \geq u_q$, because the jobs need all the raw materials, so the last job must start after $u_q$ and the deadlines are negative.

3. $L^*_{max} \geq \sum_{j=1}^{n} p_j$, because this is the earliest time moment when every job can finish, and the deadlines are negative.

Take an arbitrary active algorithm ($Alg$) and an arbitrary instance. The required raw materials arrive until $u_q$, so we have to schedule the jobs after $u_q$ without any idle time. This means every job finishes until $u_q + \sum_{j=1}^{n} p_j$, so the makespan of the schedule is at most $u_q + \sum_{j=1}^{n} p_j$ and the maximum lateness is at most $u_q + \sum_{j=1}^{n} p_j - d_1$. It is trivial from our bounds that

$$3L^*_{max} \geq u_q + \sum_{j=1}^{n} p_j - d_1 \geq L^{Alg}_{max}$$

□

As we mentioned we have to modify Early EDD, because the proof of Theorem 3.1.7 is not correct (see Remark 3.2.2). Note that, the original definition of Early EDD contains a trivial mistake (after scheduling a job we have to say $t \to t + p$ instead of $t \to t + 1$).

*Remark* 3.2.2. The original version of Early EDD did not take into consideration whether a job which is scheduled before $u_q$, finishes or not before $u_q$. That can cause problems: in the proof of Theorem 3.1.7 the authors of [8] tried to show that the Lazy EDD is always worse than the other three. This is not true, unless everything is integer and $p_j = 1$. In the next example Lazy EDD is better than Early EDD:

**Example:** Let $n = 2$, $p_1 = p_2 = 10$, $d_1 = -10$, $a_1 = 2$, $d_2 = 0$, $a_2 = 1$. At $t = 0$ one unit of raw material arrive and at $t = 1$ two units of raw material arrive. Let see what the algorithms do: Lazy EDD waits for $u_q = 1$, and then it schedules $J_1$ and at $t = 11$ it schedules $J_2$. The lateness of $J_1$ is equal with the lateness of $J_2$, so $L_{max}^{Lazy} = 21$. Early EDD schedules $J_2$ at $t = 0$ and $J_1$ at $t = 10$, so $L_{max}^{Early} = 30$.

The next definition corrects the previously mentioned flaw:

**Modified Early EDD:** Starting at time moment $t = 0$, consider the set of jobs for which the required raw materials have arrived and could finish until $u_q$. Select the $J_j$ job from this set with earliest due date, schedule it at $t$, set $t \to t + p_j$ and continue. If the set of the potential jobs is empty, but we still have jobs to schedule, let $t = u_x$ ($u_x$ is the next delivery time) and continue. After $u_q$ we schedule the remaining jobs in EDD order without any idle time.

Now let us see the generalized (and corrected) version of Theorem 3.1.7:

**Theorem 3.2.3.** *All four EDD-based algorithms (Strict EDD, Lazy EDD, Modified Early EDD and First Fit EDD) have a worst case ratio of 2 for the problem* $1|rm|L_{max}$.

*Proof.* Earlier we proved that (in the proof of Theorem 3.1.7) Lazy EDD is worse than algorithms Strict EDD and First Fit EDD in every case. First we prove that Modified Early EDD is always better than Lazy EDD: we can see from the definition that Lazy EDD does not schedule any job before $u_q$. Modified Early EDD may schedule some jobs before $u_q$, but put the remaining jobs in EDD order after $u_q$ without any idle time. This means, that every job starts later (or at the same moment) if we schedule with Lazy EDD. So it is enough to prove the theorem for the Lazy EDD, because it has always bigger maximum lateness than the lateness of any of the other three algorithms.

First let us see what happen if we schedule the jobs in EDD order from 0 without any idle time. Let $J_\ell$ be the job with the maximum lateness (this is equal with $\sum_{j=1}^{\ell} p_j - d_\ell$). This lateness must be smaller than $L^*_{max}$. Lazy EDD schedules the jobs in the same order, but it starts at $u_q$, so every lateness become longer with $u_q$. We get that the maximum lateness of Lazy EDD is $\sum_{j=1}^{\ell} p_j - d_\ell + u_q$. Since $u_q$ is smaller than $L^*_{max}$, we have

$$L^{Lazy}_{max} = \sum_{j=1}^{\ell} p_j - d_\ell + u_q \leq 2L^*_{max}$$

□

*Remark* 3.2.4. There is an easy example in [8], which shows that the bound of 2 can be achieved by all four EDD algorithms. Let

$$\begin{aligned} &rm = 1, \\ &p_1 = p_2 = \ldots = p_n = 1, \\ &d_1 = -2, \quad d_2 = d_3 = \ldots = d_n = -1, \\ &a_1 = n-1, \quad a_2 = a_3 = \ldots = a_n = 1 \end{aligned}$$

At $t = 0$ $n-1$ units of raw material become available and $n-1$ units of raw material arrive at $t = n-1$. All four algorithms schedule $\mathcal{J}_1$ at $t = 0$ and the other jobs from $t = n-1$ (without idle time), so $L^{Strict}_{max} = L^{Lazy}_{max} = L^{Early}_{max} = L^{FirstFit}_{max} = 2n-1$. In the optimal scheduling, we schedule $J_2, J_3, \ldots, J_n$ from $t = 0$ without lateness, and at $t = n-1$ we schedule $J_1$. This results a maximum lateness of $n+2$. If $n$ tends to infinity the worst case ratio of the algorithms ($(2n-1)/(n+2)$) reaches the bound of 2.

## 3.3 Makespan Minimization

This section is a review about the makespan minimization from [8]. Let the algorithm $\mathcal{A}$ be defined by the following: schedule the jobs in nondecreasing order of raw material consumption $a_j$.

**Theorem 3.3.1.** *([8]) $\mathcal{A}$ gives an optimal solution for the problem $1|rm = 1, p_j = p|C_{max}$.*

*Proof.* It is trivial with the standard interchanging method. □

**Theorem 3.3.2.** *([8]) The problem* $1|rm = 1|C_{max}$ *is strongly NP-hard.*

*Proof.* It can be found in [8]. It reduces the problem 3-PARTITION (known strongly NP-hard problem) to $1|rm = 1|C_{max}$. □

**Theorem 3.3.3.** *([8])* $1|rm = 1|C_{max}$ *with regular unit supply of raw material (i.e. at each time moment one unit of raw material is delivered) can be solved in polynomial time.*

*Proof.* We reduce the problem to the flow shop problem $F2||C_{max}$. It is known that Johnson's algorithm solve this flow shop problem in $O(n \log n)$ time. Consider an instance of our problem, we construct a corresponding flow shop instance: there are $n$ jobs with $p_{1,j} = a_j,\ p_{2,j} = p_j$ ($p_{i,j}$ is the processing time of job $J_j$ on machine $M_i$). We have to schedule every job first on $M_1$ then on $M_2$. It is trivial that this flow shop problem exactly models our problem (scheduling $J_j$ on $M_1$ corresponds to collecting the raw materials for $J_j$, while scheduling on $M_2$ corresponds to scheduling in the original problem), so we can find the optimal solution with Johnson's algorithm. □

**Theorem 3.3.4.** *([8]) The problem* $1|rm = 2, p_j = 1|C_{max}$ *is strongly NP-hard.*

*Proof.* It is easy based on the proof of Theorem 3.1.2. □

Now we would like the construct approximation algorithms for the problem $1|rm|C_{max}$. Let algorithm $\mathcal{A}_1$ be the following: schedule the jobs in arbitrary order after $u_q$ without idle time.

**Theorem 3.3.5.** *([8])* $C_{max}^{\mathcal{A}_1} \leq 2C_{max}^*$ *and the ratio of 2 is tight.*

*Proof.* $C_{max}^{\mathcal{A}_1} = u_q + \sum_{j \in \mathcal{J}} p_j \leq 2C_{max}^*$. The next instance shows that the ratio of 2 is tight: $n = 2,\ , rm = 1,\ p_1 = b,\ p_2 = 1,\ a_1 = 0,\ a_2 = 1$. One unit of raw material becomes available at time moment $b$. Then $C_{max}^* = b + 1,\ C_{max}^{\mathcal{A}_1} = 2b + 1$. If $b$ tends to infinity then $C_{max}^{\mathcal{A}_1}/C_{max}^*$ tends to 2. □

Now consider a better algorithm $\mathcal{A}_2$: it first orders the jobs in nondecreasing $\sum_i a_{i,j}$ order. In each time moment it schedules the job with the smallest index for which we have enough raw material. It is trivial that $\mathcal{A}_2$ cannot be worse than $\mathcal{A}_1$ for any instance, but it is still a 2-approximation:

Consider the next instance: $rm = 2,\ a_{1,1} = n - 1,\ a_{2,1} = 0$. For $j = 2, \ldots, n$ : $a_{1,j} = 1,\ a_{2,j} = n + j - 3,\ \forall j :\ p_j = 1$. $n-1$ units of raw material 1 becomes available at time moment 0 and at time moment $n - 1$. At $t = 0, \ldots, n - 2$ $n + t - 1$ units of raw material 2 becomes available. It is easy to see that $C_{max}^* = n$ (it schedules the jobs in the order $2, 3, \ldots, n, 1$ from $t = 0$ without idle time) while $C_{max}^{\mathcal{A}_2} = 2n - 2$,

because it schedules $J_1$ at $t = 0$, and at $t = n-1, \ldots, 2n-3$ it schedules $J_{t-n+3}$. Thus for this instance $\lim_{n\to\infty} C^{\mathcal{A}_2}_{max}/C^*_{max} = 2$. Hence we get the following theorem:

**Theorem 3.3.6.** $\lim_{n\to\infty} C^{\mathcal{A}_2}_{max}/C^*_{max} \leq 2$, *and the ratio of 2 is tight.*

*Remark* 3.3.7. The instance in [8] is incorrect, but we have fixed it.

## 3.4* A PTAS for $1|rm = 1, q = const|C_{max}$

In this section we will give a Polynomial Time Approximation Scheme (PTAS) for the problem $1|rm = 1|C_{max}$ if the number of the deliveries ($q$) is constant. This scheme can be found in [9]. Let $u_1 = 0$ be the first delivery time, and let $b'_i := \sum_{\ell=1}^{i} b_\ell$. We formulate a mathematical program for modeling the problem. Our decision variables will be $x_{ij}$ and $C_i$, where $x_{ij}$ will be 1 if and only if $J_j$ starts between $u_i$ and $u_{i+1}$, otherwise it is 0. $C_i$ will be the completion time of those jobs which are assigned to the $i$th supply period.

$$\min C_q \tag{1}$$

$$s.t.$$

$$\sum_{j\in\mathcal{J}} p_j x_{ij} \leq C_i - \max\{u_i, C_{i-1}\}, \quad i = 1, \ldots, q \tag{2}$$

$$\sum_{j\in\mathcal{J}} a_j \left(\sum_{\ell=1}^{i} x_{\ell j}\right) \leq b'_i, \quad i = 1, \ldots, q \tag{3}$$

$$\sum_{i=1}^{q} x_{ij} = 1, \quad j \in \mathcal{J} \tag{4}$$

$$C_0 = 0 \tag{5}$$

$$x_{ij} \in \{0, 1\}, \quad i \in 1, \ldots, q; \quad j \in \mathcal{J} \tag{6}$$

Let $p_{sum} := \sum_{j\in\mathcal{J}} p_j$ be the sum of the processing times. Fix $\varepsilon > 0$ as an error ratio, $\delta := \varepsilon C$, where $C$ is a suitable constant.

**Definition 3.4.1** (big and small jobs)**.** If $p_j \geq \delta p_{sum}$, then $J_j$ is a *big job*. Otherwise $J_j$ is a *small job*. Let $\mathcal{B}$ be the set of big jobs and $\mathcal{S}$ be the set of small jobs.

Now we give an outline from our algorithm. We will expound each step more detailed further on.

1. First we assign the big jobs to the different time periods in every possible way. We will show that the number of the possibilities is not too much.

2. We present that the remaining problem similar to a special multidimensional knapsack problem (we have some extra constraints, but we can violate some of the constraints a bit).

3. We search an approximate solution of the (remaining) problem: we round the processing times of the jobs, and put them into sets according to the rounded processing times.

4. We will determine that how many jobs will be assigned from each set to each time period. We will try every distribution showing that the number of these distributions is polynomial in $n$.

5. Finally we prove that we must find a solution which is near enough from the optimal solution.

Recall that $J_j$ is a big job if $p_j \geq \delta p_{sum}$. This means the number of the big jobs is at most $1/\delta$, so we want to schedule constant amount of jobs into constant number of time periods. The number of these assignments is constant (let $c_{\mathbb{B}}$ denote this constant), so we finished the first step. Let $\bar{x}^B$ denote a big job assignment ($\bar{x}^B_{ij} = 1$ if and only if big job $J_j$ is assigned to $i$th time period).

Note that we do not have to deal with every case. $\bar{x}^B$ is eligible if it does not hurt any of the raw material constraints, and for every time period there is an order of the assigned jobs where each of them can start before the end of the time period (when the next supply comes). From now on we just deal with the eligible assignments.

Let $t_i(\bar{x}^B)$ be the termination time of the big jobs which are assigned to the $i$th time period if we schedule them as soon as possible (in the $i$th period the first job starts when the machine becomes free, and then we schedule the remaining assigned big jobs without idle time). If $t_i(\bar{x}^B) < u_{i+1}$ we can schedule small jobs into this 'gap'. Let $b^r_i(\bar{x}^B) = b'_i - \sum_{j\in\mathcal{B}} a_j \left(\sum_{\ell=1}^{i} \bar{x}^B_{ij}\right)$ the residual raw material if $\bar{x}^B$ is fix. We can formulate the following model for the problem of the small jobs scheduling:

$$OPT^S(\bar{x}^B) := \max \sum_{i<q, j\in\mathcal{S}} p_j x_{ij} \tag{7}$$

$s.t.$

$$\sum_{j\in\mathcal{S}} a_j \left( \sum_{\ell=1}^{i} x_{\ell j} \right) \leq b_i^r(\bar{x}^B), \quad i = 1, \ldots, q-1 \tag{8}$$

$$\sum_{j\in\mathcal{S}} p_j x_{ij} \leq \max\{0, u_{i+1} - t_i(\bar{x}^B)\} + \delta p_{sum}, \quad i = 1, \ldots, q-1 \tag{9}$$

$$\sum_{i=1}^{q-1} x_{ij} \leq 1, \quad j \in \mathcal{S} \tag{10}$$

$$x_{ij} \in \{0, 1\}, \quad i \in 1, \ldots, q-1; \quad j \in \mathcal{S} \tag{11}$$

In the inequalities (9) we let a small lateness ($\delta p_{sum}$), because it is possible that a job starts in the $i$th time period and finishes in the $(i+1)$th. Since every small job has a processing time at most $\delta p_{sum}$ we can schedule each of them in a mentioned place. This means that some of the jobs may start a little bit later, but in that case we filled the 'gap' completely.

The remaining task is to find a $\delta$-approximate solution of this program. If we can do this we search that solution for every big job assignment and choose the best. This solution with its big job assignment will be an $\varepsilon$-approximation for the original problem.

If we cannot assign a job to any of the 'gaps' we have to assign it somewhere after $u_q$. In our model we suppose that we do not have a job like this (if we have, we will schedule them at the end of the schedule). Note that this simplification depends on $\bar{x}^B$, so we use the notation $\mathcal{S}(\bar{x}^B)$ for the remaining small jobs.

It is enough to find an approximate solution which hurts some of the constraints (9) but the total violation is at most $\delta p_{sum}^{\mathcal{S}(\bar{x}^B)}$ ($p_{sum}^{\mathcal{S}(\bar{x}^B)} = \sum_{j\in\mathcal{S}(\bar{x}^B)} p_j$), because then we may schedule some jobs later, but this increases our makespan by at most $\delta p_{sum}^{\mathcal{S}(\bar{x}^B)}$, so the solution remains $\delta$-approximate. Let $OPT$ denote the optimum of this new problem.

The main idea of the small job scheduling came from [4]. In this article the authors give a PTAS for the Multiple Knapsack Problem. We use the first part ('guessing items') of their algorithm, but we have to modify it, because of the raw material constraints. On the other hand, we can allow small violation of the constraints in some places. Let $p_{max}^{\mathcal{S}(\bar{x}^B)} = \max_{j\in\mathcal{S}(\bar{x}^B)} p_j$, and $n' = |\mathcal{S}(\bar{x}^B)|$. First we search a value $\mathcal{O}$ which is just smaller than $OPT$ i. e., $(1+\delta)\mathcal{O} \geq OPT$. Since $p_{max}^{\mathcal{S}(\bar{x}^B)} \leq OPT \leq n' p_{max}^{\mathcal{S}(\bar{x}^B)}$, there must be a good value for $\mathcal{O}$ in the following (poly-

nomial size) set:

$$\mathcal{T} = \{p_{max}^{\mathcal{S}(\bar{x}^B)} \cdot (1+\delta)^i \mid 0 \ \leq \ i \leq \ 2\delta^{-1} \ln n', \ i \in \mathbb{N}\}$$

We get the upper bound from the $i \leq \log_{1+\delta} n' = \ln n' / \ln(1+\delta) \leq \ln n'/(\delta/2)$ inequalities. Now we modify the processing times of our jobs in the following way:

1. We choose a value for $\mathcal{O}$ from the set $\mathcal{T}$.
2. Discard all items where $p_j < \dfrac{\delta\mathcal{O}}{n'}$.
3. If $(1+\delta)^i \cdot \dfrac{\delta\mathcal{O}}{n'} \leq p_j < (1+\delta)^{i+1} \cdot \dfrac{\delta\mathcal{O}}{n'}$ for some $i \in \mathbb{N}$ then let $p'_j = (1+\delta)^i \cdot \dfrac{\delta\mathcal{O}}{n'}$.

Notice that with this transformation we can only lose $O(\delta)$ fraction of $OPT$ since we can lose at most $\dfrac{\delta\mathcal{O}}{n'}$ with one job.

Let $h$ be the number of the different processing times after the modification.

**Proposition 3.4.2.** $h \leq \dfrac{4}{\delta} \ln n'$.

*Proof.* We know that $(1+\delta)^h \cdot \dfrac{\delta\mathcal{O}}{n'} \leq \mathcal{O}$, so

$$h \ln(1+\delta) \leq \ln \frac{n'}{\delta} \tag{i}$$

$$h \ln(1+\delta) \leq 2 \ln n' \tag{ii}$$

$$h \leq \frac{2 \ln n'}{\delta/2} \tag{iii}$$

We get $(ii)$ from $(i)$ that we suppose $\dfrac{1}{\delta} \leq n'$ and similarly we use $\ln(1+\delta) \geq \delta/2$ to get $(iii)$. □

Let $S'$ denote the set of the remaining jobs, let $S_\ell = \{j \in S' : \ p'(j) = (1+\delta)^\ell \cdot \delta\mathcal{O}/n'\}$ and let $y_\ell = (1+\delta)^\ell \cdot \dfrac{\delta\mathcal{O}}{n'}$ denote the processing time of a job from $S_\ell$. We choose jobs from every $S_\ell$ to schedule them to the different time periods. Let $V_\ell^i$ the set of jobs which we will assign from $S_\ell$ to the $i$th time period. Putting $J_j \in S_\ell$ into $V_\ell^i$ correspond to assigning $J_j$ to the $i$th time period, so if $J_j \in V_\ell^i$ then $x_{ij} = 1$, otherwise $x_{ij} = 0$.

A $h(q-1)$-tuple will clearly determine the sets $V_\ell^i$ (see the next algorithm how) and thus the schedule. We prove that the number of these tuples is polynomial in $n'$ (see Lemma 3.4.6).

Now we show how we determine the sets $V_\ell^i$ from a $h(q-1)$-tuple. Initially let $i = 0$, every $V_\ell^i = \emptyset$ and let $k_\ell^i$ $(i = 1, \ldots, q-1; \ \ell = 1, \ldots, h)$ a given $h(q-1)$-tuple.

Suppose that the jobs in $S_\ell$ are ordered in non-decreasing raw material demand.

**Determining the $V_\ell^i$ from a $h(q-1)$-tuple (see *Figure 3.*):**

1. $i := i + 1$. For every $\ell = 1, \ldots, h$ do the following steps.
2. Choose the smallest number of jobs from the beginning of the ordered set $S_\ell$ whose cumulative (modified) processing time is at least $k_\ell^i(\delta\mathcal{O}/h)$. Put these jobs into $V_\ell^i$. Discard the cases when we do not have enough element in $S_\ell$.
3. Delete the assigned jobs from $S_\ell$. If $i < q-1$ go to 1., else STOP.

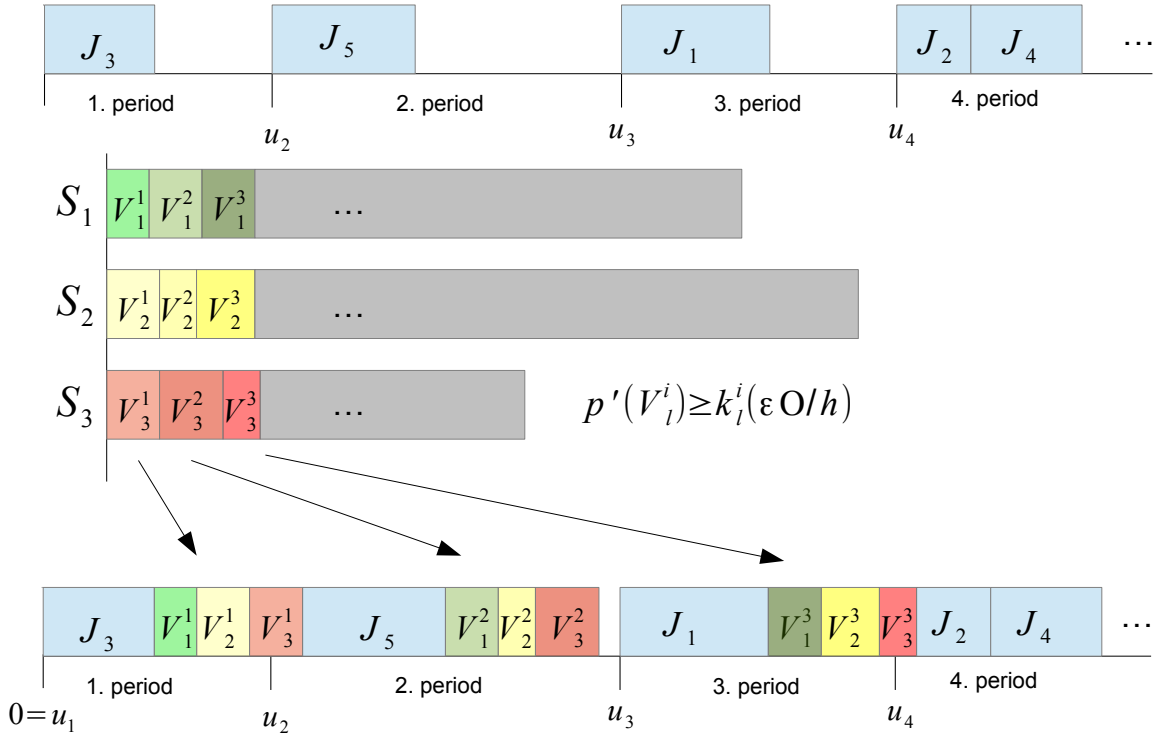


*Figure 3. Scheduling of the small jobs*

*Remark* 3.4.3. It is important that we choose jobs which require the smallest amount of raw material, because this secures that we do not hurt the raw material constraints (8) with our later chosen $h(q-1)$-tuple (see the proof of the Lemma 3.4.7).

*Remark* 3.4.4. We have $n'$ jobs and we put each of them into a set at most once, so this algorithm is polynomial in $n'$ ($O(n')$) for a given $h(q-1)$-tuple.

We can run the previous algorithm in polynomial number of times. Our aim is to find a tuple from which we can generate an approximate solution for problem (7)-(11). It is useless to deal with tuples that give us a 'too good solution'

(i.e. $\sum_{i<q,j\in\mathcal{S}'} p'_j x_{ij} \geq \mathcal{O}$). For a given $h(q-1)$-tuple the total modified processing time of the assigned jobs is at least $\sum_{i,\ell} k^i_\ell(\delta\mathcal{O}/h)$, so we can use the constraint $\sum_{l,i} k^i_\ell \leq h/\delta$. The following lemmas prove that it is enough to run the previous algorithm in polynomial number of times if we want to try every possible $h(q-1)$-tuple which does not hurt our constraints (and consists of only nonnegative integers). The first lemma is used by [4] and they have a very similar lemma instead of the second.

**Lemma 3.4.5.** *([4]) Let $f$ be the number of $g$-tuples of nonnegative integers such that the sum of tuple coordinates is at most $d$. Then $f = \binom{d+g}{g}$. If $d+g \leq \alpha g$, then $f = O(e^{\alpha g})$.*

*Proof.* The first part of the lemma is a well known result in combinatorics. If we use it for the second part we get: $f \leq \binom{\alpha g}{g} \leq (\alpha g)^g/g!$. We can approximate $g!$ by $\sqrt{2\pi g}(g/e)^g$ if we use Stirling's formula. So $f = O((e\alpha)^g) = O(e^{\alpha g})$. □

**Lemma 3.4.6.** *Let $h \leq 4\delta^{-1}\ln n'$. Then the number of the $h(q-1)$-tuples $(k^1_1, \ldots, k^{q-1}_h)$ such that $\sum_{\ell,i} k^i_\ell \leq h/\delta$, $k^i_\ell \in \mathbb{N}$ is $O(n'^{\,O(1/\delta^2)})$.*

*Proof.* It is elementary counting from the previous lemma ($\alpha = 1 + 1/\delta$). □

Now we only have to prove that we always find a tuple, which gives us a good approximate solution for problem (7)-(11):

**Lemma 3.4.7.** *There exists a $h(q-1)$-tuple $(k^1_1, \ldots k^{q-1}_h)$ such that we get a nearly allowable (i.e. the total violation in constraints (9) is at most $\delta p^{\mathcal{S}(\bar{x}^B)}_{sum}$) solution for problem (7)-(11) with the previously presented algorithm and*

$$\sum_{i,\ell} p(V^i_\ell) \geq (1-(2+q)\delta)\mathcal{O}$$

*Proof.* Let $x^*_{ij}$ denote the optimal solution of (7)-(11). We denote by $U$ the set of the jobs $j$ with $x^*_{ij} = 1$. $U_\ell := U \cap S_\ell$ and $U^i_\ell \subseteq U_\ell$ is the set jobs from $U_\ell$ which are scheduled in the $i$th time period in the optimal solution. We use the $V = \cup_{i,\ell} V^i_\ell$ notation.

At some time our algorithm enumerates the $h(q-1)$-tuple which we can get from the $U^i_\ell$ in the following way: for every $i$ and $\ell$ there is a nonnegative integer $k^i_\ell$ for which: $k^i_\ell(\delta\mathcal{O}/h) \leq p'(U^i_\ell) < (k^i_\ell+1)(\delta\mathcal{O}/h)$. We prove that this tuple is good for us. Denote by $x'_{ij}$ the value of the $x_{ij}$ which we get from our algorithm with this tuple.

Our task is to prove that the constraints (8) are true with this choice, the total violation in (9) is at most $\delta p^{\mathcal{S}(\bar{x}^B)}_{sum}$ and $p(V) \geq (1-O(\delta))\mathcal{O}$ (the other constraints are trivial). First we prove a proposition which will be useful later:

**Proposition 3.4.8.** *$|V_\ell^i| \leq |U_\ell^i|$ for every $i \in \{1, 2, \ldots, q-1\}$ and $\ell \in \{1, 2, \ldots, h\}$.*

*Proof.* $U_\ell^i, V_\ell^i \subseteq S_\ell$ so for every $j \in U_\ell^i \cup V_\ell^i$ : $p'_j = y_\ell$, thus $p'(V_\ell^i) = |V_\ell^i| y_\ell$ and $p'(U_\ell^i) = |U_\ell^i| y_\ell$. Since $p'(V_\ell^i)$ is the smallest number over $k_\ell^i(\delta\mathcal{O}/h)$ which is divisible by $y_\ell$ (from the definition of $V_\ell^i$) and $p'(U_\ell^i) \geq k_\ell^i(\delta\mathcal{O}/h)$ and it is also divisible by $y_\ell$, we get $|V_\ell^i| \leq |U_\ell^i|$ for every $i \in \{1, 2, \ldots, q-1\}$ and $\ell \in \{1, 2, \ldots, h\}$. □

*(continue the proof of lemma)* From the previous proposition we get $p'(V_\ell^i) \leq p'(U_\ell^i)$, so there exist the $c_{\ell,i}$ for which $p(V_\ell^i) \leq p(U_\ell^i) + c_{\ell,i}$ and $\sum_{\ell,i} c_{\ell,i} = \delta p_{sum}^{S(\bar{x}^B)}$. It is exactly (9) with the mentioned violation in other words.

It is enough to prove instead of (8) that our schedule uses less raw material until every $u_{i'}$ than the optimal solution (because the optimal solution fulfils (8)). Since $|V_\ell^i| \leq |U_\ell^i|$ for every $i$ and $\ell$, $\sum_{i=1}^{i'} |V_\ell^i| \leq \sum_{i=1}^{i'} |U_\ell^i|$ for every $\ell$ and $i'$. From the definition of $V_\ell^i$ we get that jobs of $\cup_{i \leq i'} V_\ell^i$ have the $\sum_{i \leq i'} |V_\ell^i|$ smallest $a_j$ among the jobs of $S_\ell$ for every $\ell$. Therefore, the total demand of those jobs in $\cup_{i \leq i'} V_\ell^i$ cannot exceed the total raw material need of the jobs in $\cup_{i \leq i'} U_\ell^i$ for every $\ell$. If we add up these inequalities for every $\ell$ we get the claim.

The last step of our proof shows that: $\sum_{i,\ell} p(V_\ell^i) \geq (1 - (2+q)\delta)\mathcal{O}$. We know that

$$k_\ell^i \left(\frac{\delta\mathcal{O}}{h}\right) \leq p'(V_\ell^i) \leq p'(U_\ell^i) < (k_\ell^i + 1)\left(\frac{\delta\mathcal{O}}{h}\right) \tag{12}$$

Thus

$$\forall i, \ell \ : \ p'(U_\ell^i) - p'(V_\ell^i) \ \leq \ \frac{\delta\mathcal{O}}{h} \tag{13}$$

$$p'(U) - p'(V) \ \leq \ (q-1)\delta\mathcal{O} \tag{14}$$

$$p(U) - p(V) \ \leq \ p(U) - p'(U) + p'(U) - p'(V) \ \leq \ 3\delta\mathcal{O} + (q-1)\delta\mathcal{O} \tag{15}$$

$$(1 - (2+q)\delta)\mathcal{O} \ \leq \ p(V) \tag{16}$$

In the first inequality of (15) we use the fact $p'(V) \leq p(V)$. In the second we use (14) and the inequality $(1+\delta)p'(U) + \delta\mathcal{O} \geq p(U)$ ($\delta\mathcal{O}$ came from the discarded 'very small' jobs ($p_j < \delta\mathcal{O}/n'$)) and then $p'(U) \leq OPT \leq (1+\delta)O \leq 2O$. □

**Theorem 3.4.9.** *The previous algorithm guarantees a schedule for the problem $1|rm = 1|C_{max}$ with constant number of deliveries, where $C_{max}^{alg} \leq (1+\varepsilon)C_{max}^*$. The run-time of the algorithm is polynomial in $n$.*

*Proof.* Once our algorithm will use the same big-job assignment as the optimal schedule. Consider the problems (7)-(11) when we use this big-job assignment.

According to Lemma 3.4.7 we find an $x'$ solution for problem (7)-(11) (with a small violation in (9)), where:

$$\sum_{i<q,j\in\mathcal{S}} p_j x'_{ij} \geq (1-O(\delta)) \sum_{i<q,j\in\mathcal{S}} p_j x^*_{ij} \geq \sum_{i<q,j\in\mathcal{S}} p_j x^*_{ij} - O(\delta)C^*_{max}$$

and $x^*$ is the optimal solution of (7)-(11), because $(1+\delta)\mathcal{O} \geq OPT \geq OPT^S(\bar{x}^B)$. $x'$ generates an assignment for us (with the mentioned big-job assignment). It is possible that there is not enough time in a 'gap' for the assigned small jobs to process them (see constraints (9)), so we have to delay some jobs. Since $(q-1)\delta p_{sum} + \delta p_{sum}^{\mathcal{S}(\bar{x}^B)} \leq O(\delta)C^*_{max}$ this lateness is not big. Summarizing our results we have a schedule where until $u_q + O(\delta)C^*_{max}$ we complete jobs with nearly the same total processing time (at most $O(\delta)C^*_{max}$ fewer) than the optimal solution completes until $u_q$. If we schedule our remaining jobs in arbitrary order without idle time as soon as possible, we get a good approximation ($C^{alg}_{max} \leq (1+\varepsilon)C^*_{max}$), because these jobs have a total processing time at most $O(\delta)C^*_{max}$ more than the total processing time of the jobs scheduled after $u_q$ in the optimal solution.

Let us see the run-time of our algorithm. We have seen that the number of the big-job assignments is constant. After that we have to find a good value for $\mathcal{O}$ from the set $\mathcal{T}$. Since $|\mathcal{T}| = 2\delta^{-1}\ln n' + 1$, we can try each element of $\mathcal{T}$. In Lemma 3.4.6 we proved that the number of the tuples which we have to try is polynomial in $n$. Remark 3.4.4 shows that we can determine the sets $V^i_\ell$-s from a tuple in polynomial time. It is easy to see that creating a schedule from $V^i_\ell$-s is very fast algorithm ($O(n)$ running time). To sum up our results the whole algorithm is polynomial in $n$ ($c_{\mathbb{B}}O(\ln n')O(n'^{\,O(1/\delta^2)})O(n')O(n) = O(n^c)$ for a suitable constant $c$), so it is a PTAS. □

*Remark* 3.4.10. Our algorithm is not an FPTAS, because the number of the possible tuples is not polynomial in $1/\varepsilon$ (see Lemma 3.4.6).

*Remark* 3.4.11. If we have more than one raw material, but it is true that

$$a_{i,j} \leq a_{i,j'} \Rightarrow a_{i',j} \leq a_{i',j'} : \ \forall i,i' \in \mathcal{R},\ j,j' \in \mathcal{J}$$

then we can create a very similar PTAS.

## 3.5* A PTAS for $1|r_j, rm = 1, q = const|C_{max}$

Let $r_j$ denote the release time of $J_j$. Consider the problem $1|r_j, rm = 1, q = const|C_{max}$. Now we have release time constraints, i.e.

$$s_j \geq r_j \qquad \forall j \in \mathcal{J}$$

We divide the solution into two parts: first we deal with the special case when the number of the different release times until $u_q$ is constant, and then we show to reduce our original problem to this easier problem.

### 3.5.1* Constant number of different release dates until $u_q$

This algorithm will be similar to that presented in the previous section. We will often refer to that algorithm, now we just mention the main steps and highlight the differences. For details, remarks (and for some proofs) see the previous section.

Suppose that the number of the different release times smaller than $u_q$ is constant. Let $\mathcal{V} = \{v_1, \ldots, v_s\}$ be the set of release times and the $u_i$ (recall that the $u_i$ are the time points of the raw material deliveries). Assume that $0 \leq v_1 < \ldots < v_s$ and $v_{s+1} = \infty$. Let $b'_i := \sum_{\ell: u_\ell \leq v_i} b_\ell$. We formulate a mathematical program for modeling the problem. Our decision variables will be $x_{ij}$ and $C_i$, where $x_{ij}$ will be 1 if and only if $J_j$ starts in the $i$th time period between $v_i$ and $v_{i+1}$, otherwise it is 0. $C_i$ will be the completion time of those jobs which are assigned to the $i$th period.

$$\min C_s \tag{17}$$

$$s.t.$$

$$\sum_{j \in \mathcal{J}} p_j x_{ij} \leq C_i - \max\{v_i, C_{i-1}\}, \quad i = 1, \ldots, s \tag{18}$$

$$\sum_{j \in \mathcal{J}} a_j \left( \sum_{\ell=1}^{i} x_{\ell j} \right) \leq b'_i, \quad i = 1, \ldots, s \tag{19}$$

$$\sum_{i=1}^{s} x_{ij} = 1, \quad j \in \mathcal{J} \tag{20}$$

$$C_0 = 0 \tag{21}$$

$$x_{ij} \in \{0, 1\}, \quad i \in 1, \ldots, s; \quad j \in \mathcal{J} \tag{22}$$

$$\sum_{i=1}^{s} x_{ij} v_i \geq r_j, \quad j \in \mathcal{J} \tag{23}$$

Let $v_t = u_q$. After $v_t$ our problem is just the same as $1|r_j|C_{max}$ thus it is optimal to schedule the remaining jobs in non-decreasing order of their release times (every job as soon as possible, for a proof see [11]). Note that $t$ is constant.

It is easy to see that the number of the eligible big job assignments is a constant, so we can try every possibility (a big job assignment is not eligible if it hurts a release time constraint). Now we transform the rest of our problem into a maximization problem like we did it in the previous section (we use the same definitions):

$$OPT^S(\bar{x}^B) := \max \sum_{i<t, j\in\mathcal{S}} p_j x_{ij} \tag{24}$$

*s.t.*

$$\sum_{j\in\mathcal{S}} a_j \left( \sum_{\ell=1}^{i} x_{\ell j} \right) \leq b_i^r(\bar{x}^B), \quad i = 1, \ldots, t-1 \tag{25}$$

$$\sum_{j\in\mathcal{S}} p_j x_{ij} \leq \max\{0, v_{i+1} - t_i(\bar{x}^B)\} + \delta p_{sum}, \quad i = 1, \ldots, t-1 \tag{26}$$

$$\sum_{i=1}^{t-1} x_{ij} \leq 1, \quad j \in \mathcal{S} \tag{27}$$

$$x_{ij} \in \{0, 1\}, \quad i \in 1, \ldots, t-1; \quad j \in \mathcal{S} \tag{28}$$

$$x_{ij} = 0, \quad \forall (i,j) : \; r_j > v_i \tag{29}$$

The remaining task is to find a $\delta$-approximate solution of this program. If we can do this we search that solution for every big job assignment and choose the best. This solution with its big job assignment will be an $\varepsilon$-approximation for the original problem.

Like in the previous section, it is enough if we find an approximate solution which hurts some of the constraints (26) but the total violation is at most $\delta p_{sum}^{\mathcal{S}(\bar{x}^B)}$ ($p_{sum}^{\mathcal{S}(\bar{x}^B)} = \sum_{j\in\mathcal{S}(\bar{x}^B)} p_j$), this increases our makespan by at most $\delta p_{sum}^{\mathcal{S}(\bar{x}^B)}$, so the solution remains $\delta$-approximate. Let $OPT$ denote the optimum of this new problem.

Let $p_{max}^{\mathcal{S}(\bar{x}^B)} = \max_{j\in\mathcal{S}(\bar{x}^B)} p_j$, and $n' = |\mathcal{S}(\bar{x}^B)|$. Consider the polynomial size set $\mathcal{T}$ (see the previous section) and choose a value $\mathcal{O}$ in every possible way. Once we must choose a value $\mathcal{O}$ which is just smaller than $OPT$ ($(1+\delta)\mathcal{O} \geq OPT$). After this, modify the processing times of the jobs, like we did it in the previous section ($p'_j = (1+\delta)^i \cdot \frac{\delta\mathcal{O}}{n'}$). According to Proposition 3.4.2, the number of the different modified processing times ($h$) is at most $\frac{4}{\delta} \ln n'$.

Let $S'$ denote the set of the remaining jobs, let $S_\ell = \{j \in S' : \; p'(j) = (1+\delta)^\ell \cdot \delta\mathcal{O}/n'\}$ and let $y_\ell = (1+\delta)^\ell \cdot \delta\mathcal{O}/n'$ denote the processing time of a job from $S_\ell$. Let $S_\ell^k = \{j \in S_\ell : \; r_j = v_k\}$. We choose jobs from every $S_\ell^k$ to schedule them to the different time periods. Let $V_\ell^{i,k}$ the set of jobs which we will assign from $S_\ell^k$ to the $i$th time period. Putting $J_j \in S_\ell^k$ into $V_\ell^{i,k}$ correspond to assigning $J_j$ to the $i$th time period, so if $J_j \in V_\ell^{i,k}$ then $x_{ij} = 1$, otherwise $x_{ij} = 0$.

A $h(t-1)^2$-tuple will clearly determine the sets $V_\ell^{i,k}$ (see the next algorithm how) and thus the schedule. We prove that the number of these tuples is polynomial in $n'$ (see Lemma 3.5.1).

Now we show how we determine the sets $V_\ell^{i,k}$ from a $h(t-1)^2$-tuple. Initially let $i = 0$, every $V_\ell^{i,k} = \emptyset$ and let $k_\ell^{i,k}$ ($i = 1, \ldots, t-1; \; \ell = 1, \ldots, h, \; k = 1, \ldots, t-1$)

a given $h(t-1)^2$-tuple. Suppose that the jobs in $S_\ell^k$ are ordered in non-decreasing raw material demand.

**Determining the $V_\ell^{i,k}$ from a $h(t-1)^2$-tuple:**

1. $i := i+1$. For every $\ell = 1, \ldots, h$ and $k = 1, \ldots, t-1$ do the following steps.
2. Choose the smallest number of jobs from the beginning of the ordered set $S_\ell^k$ whose cumulative (modified) processing time is at least $k_\ell^{i,k}(\delta\mathcal{O}/h)$. Put these jobs into $V_\ell^{i,k}$. Discard the cases when we do not have enough element in $S_\ell^k$ and when $V_\ell^{i,k} \neq \emptyset$ for a pair $i < k$.
3. Delete the assigned jobs from $S_\ell^k$. If $i < t-1$ go to 1., else STOP.

We can run the previous algorithm in polynomial number of times (for a given tuple it is polynomial). Our aim is to find a tuple from which we can generate an approximate solution for problem (24)-(29). It is useless to deal with tuples that give us a 'too good solution' (i.e. $\sum_{i<s, j\in\mathcal{S}'} p'_j x_{ij} \geq \mathcal{O}$). For a given $h(t-1)^2$-tuple the total modified processing time of the assigned jobs is at least $\sum_{i,\ell,k} k_\ell^{i,k}(\delta\mathcal{O}/h)$, so we can use the constraint $\sum_{l,i} k_\ell^i \leq h/\delta$. The following lemma proves that it is enough to run the previous algorithm in polynomial number of times if we want to try every possible $h(t-1)^2$-tuple which does not hurt our constraints (and consists of only nonnegative integers):

**Lemma 3.5.1.** *Let $h \leq 4\delta^{-1}\ln n'$. Then the number of the $h(t-1)^2$-tuples $(k_1^{1,1}, \ldots, k_h^{t-1,t-1})$ such that $\sum_{\ell,i,k} k_\ell^{i,k} \leq h/\delta$, $k_\ell^{i,k} \in \mathbb{N}$ is $O(n'^{\,O(1/\delta^2)})$.*

*Proof.* It is elementary counting from Lemma 3.4.5 ($\alpha = 1 + 1/\delta$). □

The next lemma proves that we always find a tuple, which gives us a good approximate solution for problem (24)-(29):

**Lemma 3.5.2.** *There exists a $h(t-1)^2$-tuple $(k_1^{1,1}, \ldots k_h^{t-1,t-1})$ such that we get a nearly allowable (i.e. the total violation in constraints (26) is at most $\delta p_{sum}^{\mathcal{S}(\bar{x}^B)}$) solution for problem (24)-(29) with the previously presented algorithm and*

$$\sum_{i,\ell,k} p(V_\ell^{i,k}) \geq (1 - (3 + (t-1)^2)\delta)\mathcal{O}$$

*Proof.* Let $x_{ij}^*$ denote the optimal solution of (24)-(29). We denote by $U$ the set of the jobs $j$ with $x_{ij}^* = 1$. $U_\ell := U \cap S_\ell$ and $U_\ell^i \subseteq U_\ell$ is the set jobs from $U_\ell$ which are scheduled in the $i$th time period in the optimal solution. $U_\ell^{i,k} := \{j \in U_\ell^i : \ r_j = v_k\}$ We use the $V = \cup_{i,\ell,k} V_\ell^{i,k}$ notation.

At some time our algorithm enumerates the $h(t-1)^2$-tuple which we can get from the $U_\ell^{i,k}$ in the following way: for every $i, k$ and $\ell$ there is a nonnegative integer $k_\ell^{i,k}$ for which: $k_\ell^{i,k}(\delta\mathcal{O}/h) \leq p'(U_\ell^{i,k}) < (k_\ell^{i,k}+1)(\delta\mathcal{O}/h)$. We prove that this tuple is good for us. Denote by $x'_{ij}$ the value of the $x_{ij}$ which we get from our algorithm with this tuple.

Our task is to prove that the constraints (25) are true with this choice, the total violation in (26) is at most $\delta p_{sum}^{S(\bar{x}^B)}$, we do not hurt (29) and $p(V) \geq (1-(3+(t-1)^2)\delta)\mathcal{O}$ (the other constraints are trivial).

Similarly Proposition 3.4.8 we get $|V_\ell^{i,k}| \leq |U_\ell^{i,k}|$ for every $i, k$ and $\ell$. From that we get $p'(V_\ell^{i,k}) \leq p'(U_\ell^{i,k})$, so there exist the $c_{\ell,i,k}$ for which $p(V_\ell^{i,k}) \leq p(U_\ell^{i,k}) + c_{\ell,i,k}$ and $\sum_{\ell,i,k} c_{\ell,i,k} = \delta p_{sum}^{S(\bar{x}^B)}$. It is exactly (26) with the mentioned violation in other words.

It is enough to prove instead of (25) that our schedule uses less raw material until every $v_{i'}$ than the optimal solution (because the optimal solution fulfils (25)). Since $|V_\ell^{i,k}| \leq |U_\ell^{i,k}|$ for every $i, k$ and $\ell$, $\sum_{i=1}^{i'} |V_\ell^{i,k}| \leq \sum_{i=1}^{i'} |U_\ell^{i,k}|$ for every $\ell, k$ and $i'$. From the definition of $V_\ell^{i,k}$ we get that jobs of $\cup_{i\leq i'} V_\ell^{i,k}$ have the $\sum_{i\leq i'} |V_\ell^{i,k}|$ smallest $a_j$ among the jobs of $S_\ell^k$ for every $\ell$ and $k$. Therefore, the total demand of those jobs in $\cup_{i\leq i'} V_\ell^{i,k}$ cannot exceed the total raw material need of the jobs in$\cup_{i\leq i'} U_\ell^{i,k}$ for every $\ell$ and $k$. If we add up these inequalities for every $\ell$ and $k$ we get the claim.

According to the definition of $k_\ell^{i,k}$, $V_\ell^{i,k} = \emptyset$ if $i < k$ (since $U_\ell^{i,k} = \emptyset$ if $i < k$) so the $V_\ell^{i,k}$ fulfil (29).

The last step of our proof shows that: $\sum_{i,\ell,k} p(V_\ell^{i,k}) \geq (1-(3+(t-1)^2)\delta)\mathcal{O}$. We know that

$$k_\ell^{i,k}\left(\frac{\delta\mathcal{O}}{h}\right) \leq p'(V_\ell^{i,k}) \leq p'(U_\ell^{i,k}) < (k_\ell^{i,k}+1)\left(\frac{\delta\mathcal{O}}{h}\right) \tag{30}$$

Thus

$$\forall i,\ell,k \ : \ p'(U_\ell^{i,k}) - p'(V_\ell^{i,k}) \ \leq \ \frac{\delta\mathcal{O}}{h} \tag{31}$$

$$p'(U) - p'(V) \ \leq \ (t-1)^2\delta\mathcal{O} \tag{32}$$

$$p(U) - p(V) \ \leq \ p(U) - p'(U) + p'(U) - p'(V) \ \leq \ 3\delta\mathcal{O} + (t-1)^2\delta\mathcal{O} \tag{33}$$

$$(1-(3+(t-1)^2)\delta)\mathcal{O} \ \leq \ p(V) \tag{34}$$

In the first inequality of (33) we use the fact $p'(V) \leq p(V)$. In the second we use (34) and the inequality $(1+\delta)p'(U) + \delta\mathcal{O} \geq p(U)$ ($\delta\mathcal{O}$ came from the discarded 'very small' jobs ($p_j < \delta\mathcal{O}/n'$)) and then $p'(U) \leq OPT \leq (1+\delta)O \leq 2O$. □

**Theorem 3.5.3.** *The previous algorithm guarantees a schedule for the problem* $1|r_j, rm = 1|C_{max}$ *with constant number of deliveries if there is constant number*

*of different the $r_j$, where $C^{alg}_{max} \leq (1+\varepsilon)C^*_{max}$. The run-time of the algorithm is polynomial in $n$.*

*Proof.* Similar to the proof of Theorem 3.4.9. □

### 3.5.2* $1|r_j, rm = 1, q = const|C_{max}$

In this subsection we deal with the general release time case. We reduce this case to the one with a constant number of different release times until $u_q$ (see the previous subsection).

Let $\varepsilon$ be fix. First we round up every release time not greater than $u_q$ to the nearest $t\varepsilon u_q$ where $t \in \mathbb{Z}$ (if $(t-1)\varepsilon u_q < r_j \leq t\varepsilon u_q$ then let $r'_j = t\varepsilon u_q$). After the rounding procedure we have a constant number of different release times ($1/\varepsilon$ at most) so we can use the algorithm described in the previous subsection. With this algorithm we can find an $\varepsilon$-approximate solution of the rounded problem. We show that this solution is also an $\varepsilon$-approximate solution for the original problem.

**Lemma 3.5.4.** *Let $C^r_{max}$ denote the optimum of the rounded problem. Then*

$$C^r_{max} \leq C^*_{max} + \varepsilon u_q.$$

*Proof.* Consider an optimal schedule of the original problem ($S$). Let $S'$ be a schedule where every job starts $\varepsilon u_q$ later than in schedule $S$. $S'$ does not hurt rounded release times because we know that

$$\begin{aligned} r_j &\leq s_j && \forall j \in \mathcal{J}, \\ s_j + \varepsilon u_q &= s'_j && \forall j \in \mathcal{J}, \\ r'_j &\leq r_j + \varepsilon u_q && \forall j \in \mathcal{J}, \end{aligned}$$

thus

$$r'_j \leq r_j + \varepsilon u_q \leq s_j + \varepsilon u_q = s'_j \qquad \forall j \in \mathcal{J}.$$

So $S'$ is a feasible schedule for the rounded problem, hence

$$C^r_{max} \leq C^{S'}_{max} = C^*_{max} + \varepsilon u_q.$$

□

**Theorem 3.5.5.** *Let $S^r$ be the schedule that we can find with the algorithm of the previous subsection for the rounded problem. The makespan of this schedule ($C^{Alg}_{max}$) is at most $(1+\varepsilon)^2 C^*_{max}$.*

*Proof.* We know that $u_q \leq C^*_{max}$. From Theorem 3.5.3 and Lemma 3.5.4 we get

$$C^{Alg}_{max} \leq (1+\varepsilon)C^r_{max} \leq (1+\varepsilon)\left(C^*_{max} + \varepsilon u_q\right) \leq (1+\varepsilon)^2 C^*_{max}.$$

□

# Chapter 4

# Precedence constraints

The problem of this chapter has a new property. Now, we do not want to schedule our jobs on machines, it is possible to schedule any number of jobs at the same moment. On the other hand we have some precedence constraints: for some pairs $(J_j, J_k)$ we have given nonnegative constants $q_{jk}$ which turn up in the following precedence constraints:

$$s_k \geq C_j + q_{jk}$$

Throughout this chapter we suppose that our problem can be described with a directed acyclic graph (DAG), where the jobs are the nodes and we have an arc from $J_j$ to $J_k$ with a length $\ell_{jk} = p_j + q_{jk}$ if and only if we have a precedence constraint $s_k \geq C_j + q_{jk}$. We have one non-renewable raw material, we use the previously introduced notations to describe it. This chapter is based on [3].

## 4.1 Minimizing $C_{max}$

We introduce two new nodes ($n_0$, $n_*$) and some new arcs to the DAG: we direct arcs from $n_0$ to each $J_j$ node with a 0 length and from every $J_j$ to $n_*$ with a length of $p_j$. Let $D = (V, A)$ denote this new DAG. First we search the critical path in $D$. Let $t^*$ be the length of the (directed) longest path from $n_0$ to $n_*$. We can recursively calculate for every $J_j$ the latest possible time $t_j$ when it can start if we want to finish our scheduling until $t_*$ ($t_j = \min\{t_k - \ell_{jk} | \ (j,k) \in A\}$). Note that the schedule, where $J_j$ starts at $t_j$ is optimal if we do not take into consideration the raw material constraints.

**Proposition 4.1.1.** *There is an optimal schedule and a nonnegative constant $\delta$ where $s_j = t_j + \delta$, $\forall j$.*

*Proof.* Let $\delta := C^*_{max} - t_*$ ($\delta \geq 0$, because $t_*$ is the makespan of problem where we do not have raw material constraints). The schedule $S$, where $s_j = t_j + \delta,\ \forall j$ does not break the precedence constraints and its makespan is $C^*_{max}$. Let $S'$ be an arbitrary optimal schedule. It is easy to see that $s_j \geq s'_j,\ \forall j \in \mathcal{J}$. Since $S'$ satisfy the raw material constraints, $S$ is a feasible schedule. □

Let $A_\delta(t) = \sum_{t_j+\delta \leq t} a_j$, $B(t) = \sum_{u_i \leq t} b_i$, we have to find the smallest $\delta$ for which $A_\delta(t_j) \leq B(t_j),\ \forall j$. Since $A_\delta(t)$ and $B(t)$ are both stepwise increasing functions, the optimal $\delta^*$ is the smallest nonnegative number for which $A_0(t_j - \delta^*) \leq B(t_j),\ \forall j$.

*Remark* 4.1.2. This algorithm has a time complexity of $O(|A| + n \log n + q \log q)$ ($q$ is number of the deliveries). For details see [3].

## 4.2 Deadline constraints

In this section we still want to minimize $C_{max}$, but now we have deadline constraints i.e. $C_j \leq d_j,\ \forall j$. The main idea of our algorithm is still the shifting the graph of $A_0(t)$. Let $\delta_j = d_j - t_j - p_j$ and denote $\pi$ the permutation for which $\delta_{\pi(1)} \leq \delta_{\pi(2)} \leq \cdots \leq \delta_{\pi(n)}$. We calculate the optimal $\delta^*$ with binary search according to where it locates in the previous sequence. For a given $\delta^*$ we may hurt some of the deadline constraints (if $\delta_{\pi(i)} \leq \delta^* \leq \delta_{\pi(i+1)}$ then jobs $J_{\pi(1)}, \ldots, J_{\pi(i)}$ hurt the constraint). If $J_j$ hurts its deadline constraint, we schedule it at $s_j = d_j - p_j$, discard it from the problem and reduce the suitable $b_i$ by $a_j$. A more formal version of this algorithm can be found in [3]. It is easy to see, that the running time of this new algorithm is still $O(|A| + n \log n + q \log q)$.

## 4.3 Other cost functions

Assume that the $f_j(t),\ j = 1, \ldots, n$ are arbitrary continuous nondecreasing cost functions. We want to minimize $\max_j\{f_j(C_j)\}$ in our schedule so we search the smallest $\gamma$ for which $f_j(C_j) \leq \gamma,\ \forall j$. A given $\gamma$ generates deadlines $d_j(\gamma)$: $d_j(\gamma) = \min\{f_j^{-1}(\gamma),\ \min\{d_k(\gamma) - p_k - q_{jk} : (j,k) \in A\}\}$ ($d_*(\gamma) := \infty$). Since these values are not necessarily integers we cannot create a polynomial binary search based on the previous section. Let $U(j) = \{k \in V \setminus \{n_*\} :\ \exists (j,k) \text{ path in } D\}$ the set of successors and let $L_{jk}$ denote the length of the maximal length path from $j$ to $k$.

We need the smallest gamma such that:

$$\sum\{a_j : \ d_j(\gamma) - p_j \leq u_i\} \leq B(u_i) \qquad \forall i \tag{1}$$

$$\sum\{a_j : \ \min\{f_j^{-1}(\gamma) - p_j, \\ \min\{f_k^{-1}(\gamma) - p_k - L_{jk} : \ k \in U(j)\}\} \leq u_i\} \leq B(u_i) \qquad \forall i \tag{2}$$

We used the definition of $d_j(\gamma)$. Let $\gamma_{ij} = \max\{f_j(u_i + p_j), \ \max\{f_k(u_i + p_k + L_{jk} : k \in U(j))\}\}$, so the problem is to find the smallest $\gamma$ such that

$$\sum\{a_j : \ \gamma \leq \gamma_{ij}\} \leq B(u_i) \qquad \forall i \tag{3}$$

$G_i(\gamma) = \sum\{a : \ \gamma \leq \gamma_{ij}\}$ is a stepwise nonincreasing function of $\gamma$. For a fix $i$ we can find the optimal $\gamma_i^*$ in $O(n)$ time (see [3]). $\gamma* = \max_i\{\gamma_i^*\}$. The running time of the whole algorithm is $O(n|A| + qn^2)$, but $O(|A| + n \log n + qn)$ is also enough with a small modification ([3]).

# Chapter 5

# Summary, other results of the topic

This chapter has two aims: to summarize our results and to place them between the previous results. We present the results in tables denoting the sources of the results. In section 5.1 we examine the problems where the jobs can deliver raw materials, while in section 5.2 there is a review about the problems where the jobs only consume raw materials.

## 5.1 Producer and consumer jobs

In this section we only consider problems where there is only one raw material. Let $\mathcal{J}^+$ denote the set of the producer jobs (the so-called x-jobs in chapter 2) and $\mathcal{J}^-$ the set of the consumer jobs (y-jobs). Recall that $z_j$ is the amount of raw material with which $J_j$ increases or decreases the stock size. For simplicity we introduce some new notations for the constraints:

$p^+$: $p_j = p^+,\ \forall j \in \mathcal{J}^+$
$d^+$: $d_j = d^+,\ \forall j \in \mathcal{J}^+$
$z^+$: $z_j = z^+,\ \forall j \in \mathcal{J}^+$
$k^+$: the number of the extant constraints out of $p^+, d^+, z^+$
$p^-$: $p_j = p^-,\ \forall j \in \mathcal{J}^-$
$d^-$: $d_j = d^-,\ \forall j \in \mathcal{J}^-$
$z^-$: $z_j = z^-,\ \forall j \in \mathcal{J}^-$
$k^-$: the number of the extant constraints out of $p^-, d^-, z^-$

Table 1. summarizes the results (just the 'hardest' polynomial problems and 'easiest' NP-hard problems). "?NP" means that it is open whether it is strongly

or ordinary NP-hard ("sNP" - strongly NP-hard; "oNP" - ordinary NP-hard; P - polynomial). $U_j = 1$ if $J_j$ is tardy otherwise it is 0. Some earlier (weaker) result can be found in [2]. There is a more detailed version of the results in [1].

| **Objective** | **Constraints** | **Results** | **Source(s)** |
|---|---|---|---|
| Stock size | - | sNP, 3/2-approx. | [6], [10], chap. 2 |
| $\sum w_j C_j$ | $k^+, k^- = 2$ | P, $O(n \log n)$ | [1] |
| $\sum w_j C_j$ | ($p^+$ or $w^+$) and $k^- = 3$ | sNP | [1] |
| $\sum w_j C_j$ | ($p^-$ or $w^-$) and $k^+ = 3$ | sNP | [1] |
| $L_{max}$ | $p^+$ or $z^+$ | P, $O(n^2 \log(\sum p_j))$ | [1] |
| $L_{max}$ | $d^+$ and $k^- = 3$ | oNP | [1] |
| $L_{max}$ | $d^+, p^-, z^-$ | sNP | [1] |
| $\sum U_j$ | $d^+, p^-, k^+ = 2$ | P, $O(n^3)$ | [1] |
| $\sum U_j$ | $p^+, d^-$ and ($p^-$ or $z^-$) | P, $O(n^6)$ | [1] |
| $\sum U_j$ | $p^+, z^+, p^-, z^-$ | P, $O(n^2)$ | [1] |
| $\sum U_j$ | $d^+, z^+, d^-, z^-$ | P, $O(n^2)$ | [1] |
| $\sum U_j$ | ($d^+, k^- = 3$) or ($d^-, k^+ = 3$) | oNP | [1] |
| $\sum U_j$ | $p^+, z^+$ | ?NP | [1] |
| $\sum U_j$ | $d^+, p^-, z^-$ | sNP | [1] |

*Table 1. Problems in case of producer and consumer jobs.*

There are still some problems which are open (according to [1]). These problems are listed in Table 2.:

| **Objective(s)** | **Constraints** |
|---|---|
| $\sum w_j C_j,\ \sum U_j$ | $z^+, k^- = 3$ |
| $\sum w_j C_j,\ \sum U_j$ | $z^+, p^-, w^-$ |
| $\sum w_j C_j,\ \sum U_j$ | $z^+, p^-, z^-$ |
| $\sum w_j C_j,\ \sum U_j$ | $z^+, w^-, z^-$ |
| $\sum w_j C_j,\ \sum U_j$ | $z^+, z^-$ |
| $\sum w_j C_j,\ \sum U_j$ | $k^+ = 3, z^-$ |
| $\sum w_j C_j,\ \sum U_j$ | $p^+, w^+, z^-$ |
| $\sum w_j C_j,\ \sum U_j$ | $w^+, z^+, z^-$ |
| $\sum w_j C_j,\ \sum U_j$ | $p^+, z^+, z^-$ |

| **Objective(s)** | **Constraints** |
|---|---|
| $\sum U_j$ | $p^+, z^+, p^-$ |
| $\sum U_j$ | $p^+, p^-, z^-$ |
| $\sum U_j$ | $p^+, p^-$ |
| $\sum U_j$ | $p^+, z^-$ |
| $\sum U_j$ | $z^+, p^-$ |

*Table 2. Open problems in case of producer and consumer jobs.*

## 5.2 Only consumer jobs

We classify these problems according to their objectives. Recall that $q$ is the number of the deliveries and we use $|A|$ for the size of the precedence graph (number of the arcs). First let us see the problems where we want to minimize the makespan in Table 3.:

| **Problem** | **Results** | **Source(s)** |
|---|---|---|
| $1\|rm\|C_{max}$ | sNP, 2-approx. | [8], sec. 3.3 |
| $1\|rm = 1\|C_{max}$ | sNP | [2], [5], [8] |
| $1\|rm = 1, p_j = p\|C_{max}$ | P | [8], sec. 3.3 |
| $1\|rm = 1, p_j = 1, prec\|C_{max}$ | sNP | [2] |
| $1\|rm = 1, a_j = 1\|C_{max}$ | P, $O(n \log n)$ | [2] |
| $1\|rm = 1, p_j = 1, a_j = 1, prec\|C_{max}$ | P, $O(\|A\| + q)$ | [2] |
| $1\|rm = 1, reg.\ supp.\|C_{max}$ | P, $O(n \log n)$ | [8], sec. 3.3 |
| $1\|rm = 1, q = 2\|C_{max}$ | FPTAS | [9] |
| $1\|rm = 1, q = const\|C_{max}$ | PTAS | [9], sec. 3.4 |
| $1\|rm = 1, r_j, q = const\|C_{max}$ | PTAS | sec. 3.5 |
| $PDm\|rm = 1, p_j = 1, prec\|C_{max}$ | open if $m \geq 3$ fix | [2] |
| $PDm\|rm = 1, a_j = 1\|C_{max}$ | oNP | [2] |
| $PDm\|rm = 1, p_j = 1, r_j\|C_{max}$ | P, $O(n^2 + q)$ | [2] |
| $PDm\|rm = 1, p_j = 1, r_j\|C_{max}$ | P | [2] |
| $-\|rm = 1, prec\|C_{max}$ | P, $O(\|A\| + n \log n + q \log q)$ | [3], sec. 4.1 |
| $-\|rm = 1, d_j, prec\|C_{max}$ | P, $O(\|A\| + n \log n + q \log q)$ | [3], sec. 4.2 |

*Table 3. Makespan minimization in case of just consumer jobs.*

Table 4. summarizes the problems where the objective function is $\sum C_j$ and we collect the remaining problems in Table 5. ($T_j = \max\{0, C_j - d_j\}$):

| Problem | Results | Source(s) |
|---|---|---|
| $1\|rm = 1\|\sum C_j$ | sNP | [2], [5] |
| $1\|rm = 1, p_j = 1\|\sum C_j$ | P | [2] |
| $1\|rm = 1, p_j = 1, prec\|\sum C_j$ | sNP | [2] |
| $1\|rm = 1, p_j = 1, a_j = 1, prec\|\sum C_j$ | P | [2] |
| $1\|rm = 1, p_j = 1, r_j\|\sum C_j$ | NP | [2] |
| $PDm\|rm = 1\|\sum C_j$ | sNP | [2] |
| $PDm\|rm = 1, p_j = 1\|\sum C_j$ | P | [2] |
| $PDm\|rm = 1, p_j = 1, a_j = 1, prec\|\sum C_j$ | NP | [2] |
| $PDm\|rm = 1, p_j = 1, r_j\|\sum C_j$ | NP | [2] |
| $-\|rm = 1\|\sum C_j$ | P | [2] |
| $-\|rm = 1, p_j = 1\|\sum C_j$ | P | [2] |
| $-\|rm = 1, p_j = 1, prec\|\sum C_j$ | sNP | [2] |
| $-\|rm = 1, p_j = 1, a_j = 1, prec\|\sum C_j$ | NP | [2] |
| $-\|rm = 1, p_j = 1, r_j\|\sum C_j$ | NP | [2] |

*Table 4. $\sum C_j$ in case of consumer jobs only.*

| Problem | Results | Source(s) |
|---|---|---|
| $1\|rm\|L_{max}$ | sNP, 2-approx. | sec. 3.2 |
| $1\|rm = 1\|L_{max}$ | sNP | [5] |
| $1\|rm = 1, p_j = 1\|L_{max}$ | P, $O(n^2)$ | [8], sec. 3.1 |
| $1\|rm = 2, p_j = p\|L_{max}$ | sNP, 2-approx. | [8], sec. 3.1 |
| $1\|rm = 1, p_j = p\|\sum T_j$ | NP | [5] |
| $1\|rm = 1, d_j = d\|\sum T_j$ | sNP | [5] |
| $1\|rm = 1, d_j = d, a_j = a\|\sum T_j$ | NP | [5] |
| $1\|rm = 1\|\sum U_j$ | sNP | [5] |
| $-\|rm = 1\|\sum U_j$ | P, $O(n^2 + nq)$ | [2] |
| $-\|rm = 1, p_j = 1, prec\|\sum U_j$ | oNP | [2] |

*Table 5. Other problems in case of consumer jobs only.*

# Bibliography


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

[2] J. Carlier: Problèmes d'ordonnancements à contraintes de ressources: Algorithmes et complexité, doctorat d'état (1984) (french)

[3] J. Carlier, A. H. G. Rinnooy Kan: Scheduling Subject to Nonrenewable-Resource Constraints, *Operations Research Letters*, Volume 1, Number 2 (1982)

[4] C. Chekuri, S. Khanna: A Polinomial Time Approximation Scheme for the Multiple Knapsack Problem, *SIAM Journal on Computing*, Volume 35, Issue 3 (2006), 713–728.

[5] E. R. Gafarov, A. A. Lazarev, F. Werner: Single machine scheduling problems with financial resource constraints: Some complexity results and properties, *Mathematical Social Sciences*, 62 (2011)

[6] M. R. Garey, D. S. Johnson: Approximation Algorithms for Bin Packing Problems: A Survey. In *Analysis and Design of Algorithms in Combinatorial Optimization.*, Ausiello and Lucertini (eds.), Springer, New York (1981)

[7] R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan: Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics*, 5 (1979), 287–326.

[8] A. Grigoriev, M. Holthuijsen, J. van de Klundert: Basic Scheduling Problems with Raw Material Constraints, *Naval Research Logistics*, Vol. 52 (2005), 527–535.

[9] P. Györgyi, T. Kis: Approximation schemes for single machine scheduling with non-renewable resource constraints, *submitted to publication* (2013)

[10] H. Kellerer, V. Kotov, F. Rendl, G. J. Woeginger: The Stock size problem, *Operations Research* **7** (1996), S1–S12.

[11] E. L. Lawler: Optimal sequencing of a single machine subject to precedence constraints, *Management Sci.* **19** (1973), 544–546.

[12] P. Schuurman, G. J. Woeginger: Approximation Schemes - A Tutorial, *To appear in the book "Lectures on Scheduling", edited by R. H. Moehring, C. N. Potts, A. S. Schulz, G. J. Woeginger, L. A. Wolsey*