Approximation Algorithms For 2-dimensional Packing and Related Scheduling Problems

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

Dipl.-Math. Christina Robenek

Kiel

2012

1. Gutachter: Prof. Dr. Klaus Jansen
Christian-Albrechts-Universität zu Kiel

2. Gutachter: Prof. Dr. Denis Trystram
LIG, ENSIMAG, Grenoble Institute of Technology INPG

Datum der mündlichen Prüfung: 21. September 2012

Datum der Genehmigung des Drucks: 25. Februar 2013

Zusammenfassung

Zentrales Thema dieser Dissertation sind approximative Algorithmen für Ablaufplanungs- und Packungsprobleme, zwei klassische Problemstellungen der kombinatorischen Optimierung. Die vorliegende Arbeit besteht aus drei Teilen. Im ersten Teil betrachten wir eine Verallgemeinerung des Strip Packing Problems, auch geometrisches Zuschnittsproblem genannt. Beim Strip Packing soll eine gegebene Menge von Rechtecken in einen Streifen fester Breite und unendlicher Höhe platziert werden, so dass die genutzte Gesamthöhe minimal wird. Bei dem generalisierten Problem stehen N Streifen zur Verfügung, in welchen die Rechtecke angeordnet werden sollen. Ziel ist es das Maximum der genutzten Gesamthöhen zu minimieren. Der zweite Teil beschäftigt sich mit einem verwandten Problem der Ablaufplanung. Hierbei soll statt der Rechtecke eine Menge von parallelen Aufträgen in N Plattformen von Prozessoren ausgeführt werden. Solch ein Auftrag kann mit einem Rechteck identifiziert werden, aber im Gegensatz zum Packen von Rechtecken dürfen die Aufträge in vertikale Scheiben geschnitten und diese dann in den Plattformen angeordnet werden, solange alle Scheiben eines Auftrags zur gleichen Zeit in ein und derselben Plattform starten. Im dritten Teil untersuchen wir die Ablaufplanung auf uniformen Prozessoren, eine fundamentale 1-dimensionale Problemstellung der Ablaufplanung. Hierbei sind die Aufträge nur durch eine Ausführungszeit gegeben und sollen einer Menge von Prozessoren zugewiesen werden, die verschiedene Geschwindigkeiten haben können.

Abstract

Main subject of this thesis are approximation algorithms for scheduling and packing, two classical geometrical problems in combinatorial optimization. It is divided into three parts. In the first part we consider a generalization of the strip packing problem or geometrical cutting stock problem. In strip packing a given set of rectangles has to be placed into a strip of fixed width and infinite height minimizing the total height used. In the generalized problem setup there are N strips available in which the rectangles have to be allocated and the objective is to minimize the maximum height used. The second part deals with a related scheduling problem. Here we are given parallel jobs instead of rectangles that have to be executed in N platforms of processors. Such a job can be identified with a rectangle, but in contrast to rectangle packing in parallel job scheduling we are allowed to cut the jobs into vertical slides and place them into the platforms as long as all slides of a job start at the same time in the same platform. In the third part we investigate scheduling on uniform processors, a fundamental 1-dimensional scheduling problem. Here the jobs are described by a processing time only and have to be assigned to a set of processors that may run at different speeds.

Acknowledgements

First I would like to thank my colleague Lars Prädel for interesting me in packing and scheduling and for always being a friend.
Thanks to my advisor Prof. Dr. Klaus Jansen for accepting me as a doctoral candidate and for always having an idea.
Thanks to my coauthors Pierre-Francois Dutot, Denis Trystram and especially to Marin Bougeret for lively discussions.
Thanks to my colleague Ute Iaquinto and former colleague Ulrich Schwarz for cake and other distraction.
Thanks to my persistent readers and colleagues Kati Land and Felix Kumm.
Thanks to Stiene Riemer for inspiring lunch breaks.
Thanks to all my friends that get along with and supported me over the past couple of years.
Thanks to my siblings for choosing the same destiny of writing a thesis.
Thanks to my parents for funding my diploma study.
Thanks most of all to my husband Olaf for always taking my hand through ups and downs.

Contents

1. Introduction

Within this thesis three geometrical problems in combinatorial optimization are considered in the context of approximation algorithms.

1.1 Multiple Strip Packing

One of the fundamental and well-studied geometrical problems in combinatorial optimization is Strip Packing. For a given set of rectangles the objective is to find an arrangement of the rectangles into a strip of fixed width and infinite height so that the total height used is minimized. The rectangles should be placed axis-parallel and neither are allowed to be rotated nor to overlap.

The problem is also known as geometrical cutting stock problem and originates e.g. in processes in textile industry. Here the strip is a spool of fabric and the goal is to use as few as possible of fabric while cutting rectangular cloths from it. In huge companies of course many of such spools are in parallel use. Therefore a natural generalization of the problem is Multiple Strip Packing where we are given more than one strip in which the rectangles have to be placed. The objective is to minimize the maximum over all heights used.

In Figure 1.1 we consider a packing of 6 rectangles given by pairs of width and height into a strip of width 1. The height of the packing is $7/6$. Since each of the rectangles R_1, R_4 and R_6 has width larger than half the width of the strip they cannot be packed next to each other. Consequently, there is no packing with height less than $7/6$ and the one depicted in Figure 1.1 is optimal. In Figure 1.2 for the same rectangles a packing with minimum height into 2 strips is illustrated. The packing has height $3/4$. Also this is a packing with minimum height (for 2 strips): There can be no packing with height less than the height of the largest rectangle.

Furthermore, we observe that the minimum height of a packing into one strip is an upper bound for the minimum height of a packing into 2 or more strips.

$$
\begin{aligned}
R_1 &= (3/4, 1/2)\\
R_2 &= (1/4, 1/3)\\
R_3 &= (1/3, 1/3)\\
R_4 &= (7/12, 7/12)\\
R_5 &= (1/12, 3/4)\\
R_6 &= (11/12, 1/12).
\end{aligned}
$$

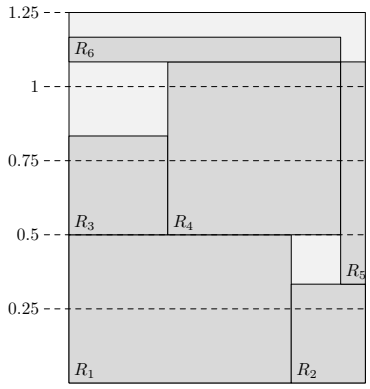

Figure 1.1: Strip packing with minimum height.

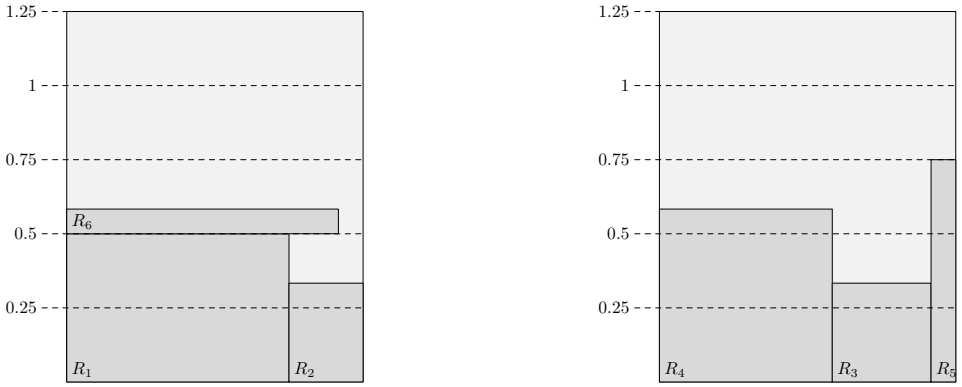

Figure 1.2: Optimum packing into 2 strips.

1.2 Scheduling Parallel Jobs in Platforms

Another classical 2-dimensional problem closely related to Strip Packing is Scheduling parallel Jobs. Here we are given a set of jobs and a set of processors. Every job has to be executed during a certain amount of time, its processing time, and requires a certain number of processors for execution. The objective is to find an assignment of the jobs to the processors, so that the latest finishing time of a job is minimized, i.e. a schedule for the jobs with minimum makespan. This problem arises from many practical applications where resource allocation is an issue. This is the case for abstract models of computing systems such as in personal computers appear.

In contrast to Strip Packing the jobs need not necessary to be assigned to processors of consecutive addresses. Both, Figure 1.3 and 1.4, show a feasible schedule for 6 jobs on 12 processors. While the schedule in Figure 1.3 corresponds to a feasible rectangle packing (where the jobs are identified with rectangles) the schedule depicted in Figure 1.4 represents no feasible packing since Job J_6 is run on two subsets of processors that are not contiguous. Even though it is sometimes possible to rearrange the jobs to achieve a feasible packing of the corresponding rectangles, actually cases are existing where there is no chance. Therefore, in general the minimum length of a feasible schedule is less than the minimum height of a packing of the corresponding rectangles.

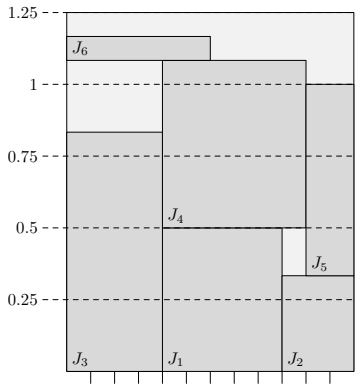

Figure 1.3: Schedule and packing.

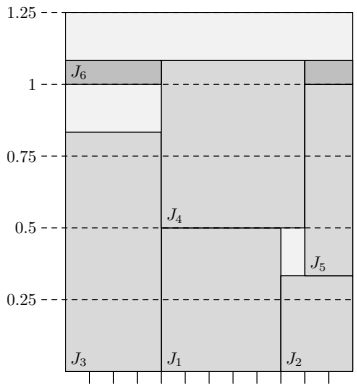

Figure 1.4: No feasible packing.

To match up with technological progress the aim is to understand and use the achievement of computational grids of computers. In this work we

investigate a problem where we are given a number of platforms where each platform contains a set of processors called Scheduling parallel Jobs in Platforms. Similar to Multiple Strip Packing the objective is to find a schedule of the jobs in the platforms of processors so that the maximum makespan over all platforms is minimized. Here, the platforms may contain different numbers of processors or run at different speeds. We also include the case that the jobs have release times, that is they are known a priori, but become available over time.

1.3 Scheduling on Uniform Processors

The last problem considered in this thesis is a 1-dimensional scheduling problem. Here we are also given a set of jobs and a set of processors, but the jobs are only given by a processing time and are executed on a single processor. For every processor a speed value is given. The objective is to find a non-preemptive schedule with minimum makespan. Similar to the problems above this problem is taken from industrial processes. One can think of the processors as machines that can manufacture different goods where each product needs a different amount of time. Figure 1.5 shows an example with 4 jobs and 2 processors where both processors have identical speeds. In Figure 1.6 a schedule for the same jobs with one processor running at speed 1 and the other running at speed 2 is given.

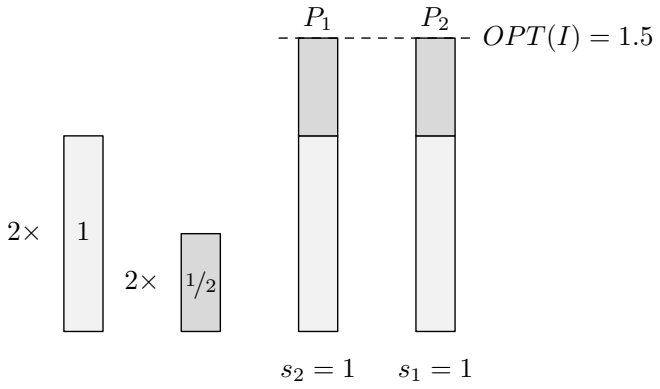

Figure 1.5: Optimum schedule for identical processors.

For all the problems described it is not possible to efficiently compute an optimum solution for a given instance, except for some special cases. As a consequence approximation algorithms are developed that compute

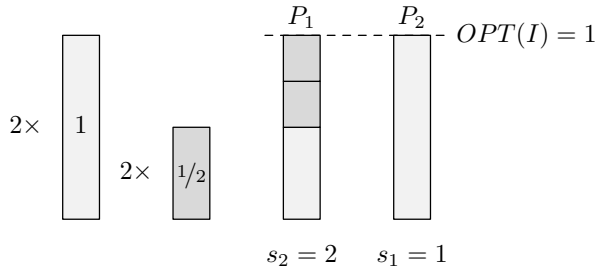

Figure 1.6: One faster processor.

near optimum or "good" solutions. In the following section we introduce some basic principles and notions from the theory of approximation algorithms without raising a claim of completeness.

1.4 Approximation Algorithms

For an instance $\mathcal{I}$ of an optimization problem Π we define with $\mathrm{OPT}(\mathcal{I})$ the value of the objective function for an optimum solution to $\mathcal{I}$. For an approximation algorithm A for Π we denote with $A(\mathcal{I})$ the value of the objective function for the solution produced for $\mathcal{I}$ by A. The *absolute ratio* or *multiplicative ratio* of an approximation algorithm is defined as the quotient

$$\frac{A(\mathcal{I})}{\mathrm{OPT}(\mathcal{I})}.$$

Consequently, the absolute ratio of an approximation algorithm for a minimization problem is larger or (in the best case) equal to 1 where for a maximization problem it is less or equal to 1. We say that an approximation algorithm A for a maximization problem Π has absolute ratio α, if

$$\frac{A(\mathcal{I})}{\mathrm{OPT}(\mathcal{I})} \geq \alpha \text{ for every instance } \mathcal{I}.$$

On the other hand for a minimization problem an approximation algorithm A has absolute ratio α, if

$$\frac{A(\mathcal{I})}{\mathrm{OPT}(\mathcal{I})} \leq \alpha \text{ for every instance } \mathcal{I}.$$

In both cases we call A an α-approximation for Π.
An algorithm for a minimization problem has *asymptotic ratio* α, if

$$\alpha \geq \limsup_{\mathrm{OPT}(\mathcal{I})\to\infty} \frac{A(\mathcal{I})}{\mathrm{OPT}(\mathcal{I})}.$$

Algorithm A has *polynomial running time*, if for every instance $\mathcal{I}$ of Π its running time is bounded by a polynomial in the *input size* or *encoding length* $\langle\mathcal{I}\rangle$. Standardly we assume all problems considered to be binary encoded. So the input size is recursively defined as follows

$$\begin{aligned}
\langle n\rangle &:= \lceil \log(|n|)\rceil + 1 \qquad \text{for } n \in \mathcal{Z}\\
\langle r\rangle &:= \langle p\rangle + \langle q\rangle \qquad \text{for } r = \frac{p}{q} \in \mathcal{Q}\\
\langle b\rangle &:= \sum_{i=1}^{n} \langle b_i\rangle \qquad \text{for } b \in \mathcal{Q}^n\\
\langle A\rangle &:= \sum_{i=1}^{n}\sum_{j=1}^{m} \langle a_{ij}\rangle \qquad \text{for } A = (a_{ij}) \in \mathcal{Q}^{n\times m}.
\end{aligned}$$

We use further the abbreviation $[n] := \{1, \ldots, n\}$.

A *polynomial time approximation scheme* (PTAS) for a minimization problem Π is a family of polynomial time approximation algorithms $(A_\varepsilon)_{\varepsilon>0}$, where for an instance $\mathcal{I}$ we have that $A_\varepsilon(\mathcal{I}) \leq (1+\varepsilon)\mathrm{OPT}(\mathcal{I})$. Note that the running time is allowed to be exponential in $1/\varepsilon$ what can lead to very large running times if ε is very small. Therefore we distinguish furthermore *efficient polynomial time approximation schemes* (EPTAS) that have running time bounded by $f(1/\varepsilon)poly(\langle\mathcal{I}\rangle)$ for a function f, and *fully polynomial time approximation schemes* (FPTAS) with running time bounded by a polynomial in both, $1/\varepsilon$ and $\langle\mathcal{I}\rangle$. In a similar way we define the notion for maximization problems.

An *asymptotic (fully) polynomial time approximation scheme* (A(F)PTAS) for a minimization problem Π is a family of (fully) polynomial time approximation algorithms $(A_\varepsilon)_{\varepsilon>0}$ each A_ε having asymptotic ratio $(1+\varepsilon)$.

1.5 Outline

This thesis is divided into three parts. In Part I we present approximation results for Multiple Strip Packing and an improved version of an AFPTAS for Strip Packing. Part II is dedicated to approximation algorithms for Scheduling parallel Jobs in Platforms. In the third part we present an EPTAS for Scheduling on Uniform Processors for that we investigate the relationship between linear program solutions and their corresponding integral solutions.

I Multiple Strip Packing

2. Introduction

2.1 Definitions

Strip Packing (SP) is the geometric version of the Cutting Stock problem or Bin Packing problem. In Bin Packing we are given n items with sizes $a_i \in (0, 1]$ that have to be placed into a minimal number of 1-dimensional bins of capacity 1. Strip Packing is a 2-dimensional generalization of Bin Packing. Here we are given a set of n rectangles $\mathcal{R}$ with height $h_j \in \mathcal{Q}_{\geq 0}$ and width $w_j \in \mathcal{Q}_{\geq 0}$ bounded by 1 for $j \in [n]$, and a rectangular region of width one and infinite height, called strip. The objective is to place all rectangles non-overlapping and axis-parallel into the strip where rotation of the rectangles is not allowed, so that the packing produced has minimum height. The optimum value for this height will be denoted with $\mathrm{OPT}_{\mathrm{SP}}(\mathcal{R})$. Multiple Strip Packing (MSP) is a generalization of Strip Packing. Here the input consists of a set of n rectangles $\mathcal{R}$ and a number N of identical strips $S_1, \ldots, S_N$ in which the rectangles have to be packed. The objective is to minimize $\max_{\ell \in [N]} h(\ell)$, where $h(\ell)$ denotes the height of the packing in strip S_ℓ. For an instance of MSP we simply write $\mathcal{R}$ suppressing the number of strips as it will always be equal to N. Furthermore we identify $\mathcal{R}$ with its set of indices $\{1, \ldots, n\}$. The optimum value of MSP for a given list of rectangles $\mathcal{R}$ will be denoted with $\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R})$.

For $j \in \mathcal{R}$ we define the *size* of a rectangle as $w_j h_j$ and consequently $SIZE(\mathcal{R}) := \sum_{j \in \mathcal{R}} w_j h_j$ for a set of rectangles. The total height of $\mathcal{R}$ is defined as $H(\mathcal{R}) := \sum_{j \in \mathcal{R}} h_j$ and the total width of $\mathcal{R}$ is given analogously. With $h_{\max} := \max_{j \in \mathcal{R}} h_j$ we denote the height of the highest rectangle given by the input $\mathcal{R}$.

By reduction from 3-Partition it follows that MSP is strongly $\mathcal{NP}$-hard. Furthermore, it can be derived that the best possible absolute ap-

proximation ratio for an approximation algorithm for MSP is 2. The existence of the inapproximability bound has been first shown by Zhuk in [67] by reduction from the common PARTITION problem. Thereby he also showed the $\mathcal{NP}$-hardness of the problem.

In 3-PARTITION the problem is to decide whether a given multiset $\mathcal{I}$ of $n = 3N$ positive integers $b_1, \ldots, b_n$ can be partitioned into N subsets $S_1, \ldots, S_N$ each containing a triple so that the numbers in each subset sum up to the same value.

Theorem 2.1. *Unless $\mathcal{P} = \mathcal{NP}$, there is no polynomial-time approximation algorithm for MSP with absolute ratio strictly less than* 2.

Proof. Consider an instance $\mathcal{I}$ of 3-PARTITION. By scaling we may assume that the total sum of the numbers given is N. By introducing a rectangle of height 1 and width b_i for each $i \in [n]$ we obtain an instance $R(\mathcal{I})$ of MSP with N strips. Then $\mathcal{I}$ is a "yes" instance of 3-PARTITION if and only if there is packing of $R(\mathcal{I})$ into N strips with height 1. For a "no" instance we have that $\mathrm{OPT}_{\mathrm{MSP}}(R(\mathcal{I}))$ is at least 2. So if for MSP there exists an approximation algorithm with absolute ratio $\alpha < 2$, it finds a packing of $R(\mathcal{I})$ of height α if and only if $\mathcal{I}$ is a "yes" instance. Therefore we can decide the 3-PARTITION problem for $\mathcal{I}$ in polynomial time. □

Moreover, unless $\mathcal{P} = \mathcal{NP}$, there is no PTAS or FPTAS for MSP and the best possible approximation is an AFPTAS.

2.2 Related Work

Since STRIP PACKING includes BIN PACKING as a special case it is strongly $\mathcal{NP}$-hard. By reduction from PARTITION it follows that there is no approximation algorithm with absolute ratio less than 1.5 for STRIP PACKING. A plenty of approximation algorithms have been developed during the last decades. In [14] Coffman et al. gave an overview about performance bounds for the shelf-orientated algorithms NFDH (Next Fit Decreasing Height) and FFDH (First Fit Decreasing Height). Those adopt an absolute ratio of 3, and 2.7, respectively. For a long time there was a huge gap between the inapproximability bound of 1.5 and the best known absolute ratio of an approximation algorithm for STRIPPACKING which was given by a 2-approximation [62, 58]. Recently, Harren et al. [26] presented

a $(5/3+\varepsilon)$-approximation for this problem. One of the most important results for Strip Packing is an asymptotic fully polynomial time approximation scheme given by Kenyon and Rémila based on a linear program relaxation for Bin Packing [45]. For any accuracy $\varepsilon > 0$ their algorithm produces a $(1+\varepsilon)$-approximative packing plus an additive height of $\mathcal{O}(1/\varepsilon^2)h_{\max}$, where $h_{\max}$ denotes the height of the tallest rectangle. In [39] Jansen and Solis-Oba gave an APTAS with small additive factor $\mathcal{O}(1)h_{\max}$, but running time exponential in $1/\varepsilon$.

As mentioned before, Multiple Strip Packing was first considered by Zhuk [67], who showed that there is no approximation algorithm with absolute ratio better than 2. Ye et al. presented an approximation algorithm with ratio $2+\varepsilon$ for this problem [65].

2.3 New Results

In this part we present several results for Multiple Strip Packing that have been partly published as an extended abstract in [5] and as a full article [8].

In Chapter 3 we show how to use the heuristics NFDH and FFDH to obtain approximation algorithms for MSP with the same asymptotic ratio as for Strip Packing. Furthermore we consider some variants of shelf-based algorithms for MSP. In Chapter 4 we present a polynomial-time approximation algorithm with absolute ratio 2, which is an improvement of the former result of $2+\varepsilon$ by Ye et al. [65] and best possible, unless $\mathcal{P} = \mathcal{NP}$. In Chapter 5 we present an improvement of the AFPTAS for Strip Packing of Kenyon and Rémila [45]. Here we were able to reduce the additive factor from $\mathcal{O}(1/\varepsilon^2)h_{\max}$ to $\mathcal{O}(1/\varepsilon\log(1/\varepsilon))h_{\max}$ and to slightly improve the running time from $\mathcal{O}(\varepsilon^{-7}+n\log n)$ to $\mathcal{O}(\varepsilon^{-6}\log(1/\varepsilon)+n\log n)$. We generalize this algorithm to several strips and directly achieve an AFPTAS for Multiple Strip Packing in Chapter 6. If the number of strips is sufficiently large we can can reduce the additive factor even more to $\mathcal{O}(1)h_{\max}$.

3. Shelf-based Algorithms for MSP

We start with some elementary results for the NFDH and FFDH heuristics [14] applied to several strips that will give an understanding of the problem. The greedy NFDH and FFDH heuristics are so-called *shelf-based* algorithms for Strip Packing. A *shelf* is a row of items placed next to each other aligned by their lower edges. The bottom of a shelf is either the bottom of the bin or at the same height as the upper edge of the tallest item packed in the shelf below. Typically the principle of operation of those algorithms is easy to grasp, but it can be difficult to analyze them.

The algorithm NFDH first orders the input rectangles by decreasing (non-increasing) height and packs the rectangles into shelves starting with the highest one. At any time there is only one shelf open. The current shelf is closed and a new one is opened if the current rectangle does not fit completely into the current shelf.

The FFDH algorithm also orders the rectangles by decreasing height and packs them into shelves. In contrast to NFDH, once a shelf is opened it remains open until the last rectangle (with smallest height) is packed. Furthermore, a rectangle is placed into the first shelf where it fits which is not necessarily the last one opened. For a set of n rectangles $\mathcal{R}$ the algorithm NFDH produces a packing of height at most $2\mathrm{OPT}_{\mathrm{SP}}(\mathcal{R}) + h_{\max}$ and FFDH produces a packing of height $1.7\mathrm{OPT}_{\mathrm{SP}}(\mathcal{R}) + h_{\max}$ in time $\mathcal{O}(n \log n)$.

Theorem 3.1. *Let A be one of the shelf-based strip packing algorithms NFDH and FFDH with asymptotic ratio $\alpha > 1$, that creates for a set of rectangles $\mathcal{R}$ a packing into a single strip of unit width with height less than $\alpha\mathrm{OPT}_{SP}(\mathcal{R}) + h_{\max}$. For any $N \in \mathbb{N}$ there exists an algorithm A_N that packs $\mathcal{R}$ into N strips so that $\max_{\ell \in [N]} h(\ell) \leq \alpha\mathrm{OPT}_{MSP}(\mathcal{R}) + h_{\max}$.*

Proof. Consider Algorithm 1. We show that for a list of rectangles $\mathcal{R}$, the packing produced by A_N has height less than $\alpha\text{OPT}_{\text{MSP}}(\mathcal{R}) + h_{\max}$. Let $t \in \mathbb{N}$ be the number of shelves produced by A in Step 1 and H_j, $j \in \{1, \ldots, t\}$, the height of the jth shelf. Since there are no items intersecting the first line, after the last step of the algorithm the height $h(1)$ of the first strip S_1 is bounded by $h_{\max} + \frac{\sum_{j=1}^{t} H_j - h_{\max}}{N}$. For any strip S_ℓ, $\ell \in \{2, \ldots, N\}$, containing items from between the $(\ell-1)$th and the ℓth line and the ones intersecting the $(\ell-1)$th line, we have

$$h(\ell) \leq \frac{\sum_{j=1}^{t} H_j - h_{\max}}{N} + h_{\max} = \frac{A(\mathcal{R}) - h_{\max}}{N} + h_{\max}.$$

With OPT_{SP} as the optimum of SP we conclude

$$\begin{aligned} A_N(\mathcal{R}) = \max_i h_i &\leq \frac{A(\mathcal{R}) - h_{\max}}{N} + h_{\max} \\ &\leq \frac{\alpha\text{OPT}_{\text{SP}}(\mathcal{R}) + h_{\max} - h_{\max}}{N} + h_{\max} = \frac{\alpha\text{OPT}_{\text{SP}}(\mathcal{R})}{N} + h_{\max}. \end{aligned}$$

Since $(1/N)\text{OPT}_{\text{SP}}(\mathcal{R})$ is a lower bound for $\text{OPT}_{\text{MSP}}(\mathcal{R})$ the proof is complete. □

Algorithm 1 A_N

Input: $\mathcal{R}$

Output: A packing of $\mathcal{R}$ into N strips

1: Pack rectangles in $\mathcal{R}$ with A into one strip S. Let H be the height of S.
2: Cut out the first shelf and pack it into the first strip S_1.
3: **for all** $\ell \in \{0, 1, \ldots, N\}$ **do**
4: Draw a horizontal line trough S at height $\ell(H - h_{\max})/N$.
5: **end for**
6: **for all** $\ell \in \{0, 1, \ldots, N-1\}$ **do**
7: Pack all items intersecting the ℓth line and all items between the ℓth and $(\ell+1)$th lines into strip $\ell + 1$.
8: **end for**

Corollary 3.2. *The algorithms FFDH_N and NFDH_N generate packings for a set of rectangles $\mathcal{R}$ into N strips with height less than $1.7\text{OPT}_{MSP}(\mathcal{R}) + h_{\max}$ and $2\text{OPT}_{MSP}(\mathcal{R}) + h_{\max}$, respectively.*

Corollary 3.3. *Let $\mathcal{R}$ be an instance of MSP. In a packing generated by the above algorithm A_N we have*

$$\max_{\ell \in \{1,\dots,N\}} |h(\ell) - A_N(\mathcal{R})| \leq 2h_{\max},$$

where $h(\ell)$ denotes the height of the packing in strip S_ℓ.

Proof. By construction the height of the packing of the rectangles selected between the ℓth and $(\ell+1)$th line is at least $\frac{\sum_{j=1}^{t} H_j - h_{\max}}{N} - h_{\max}$ and at most $\frac{\sum_{j=1}^{t} H_j - h_{\max}}{N} + h_{\max}$. □

Another way to pack a set of rectangles with a modified version of the NFDH heuristic into N strips is Algorithm 2. The packing generated by

Algorithm 2

Input: $\mathcal{R}$

Output: A packing of $\mathcal{R}$ into N strips

1: Order the rectangles in $\mathcal{R}$ by non-increasing height.
2: For each $\ell \in [N]$ pack one shelf according to the NFDH heuristic in strip S_ℓ, that means starting in the lower left corner pack the rectangles next to each other on the baseline of strip S_ℓ, until the next rectangle does not fit. Draw a new baseline at the height of the highest rectangle (that clearly is the first one).
3: Take the strip S^- with the current lowest height h^- (if there is more than one, take the one with the smallest index) and pack one shelf according to the NFDH heuristic on top of the shelves.
4: Repeat Step 3 until all rectangles are packed.

that algorithm is very smooth, in the sense that the heights of the strips only differ by $h_{\max}$.

Lemma 3.4. *For a set of rectangles $\mathcal{R}$ Algorithm 2 generates a packing into N strips, so that*

$$\max_{\ell,k \in [N]} |h(k) - h(\ell)| \leq h_{\max}.$$

Proof. Let $a_1, \dots, a_r$ denote the shelves created by the algorithm and let b_i denote the height of the first rectangle placed in shelf a_i, $i \in \{1, \dots, r\}$, clearly $b_1 \geq \dots \geq b_r$. We show per induction on i that the claim is true after creating shelf a_i.

During step 2 the assertion is obviously true.

Let $1 \leq i_0 < r$ and assume that the assertion is true for the current packing with shelves a_i, $i \leq i_0$. Let S^- be the strip with the lowest current height h^- and h^+ the height of the currently highest strip S^+ after packing shelf a_{i_0}. The value $h(\ell)^\star$ denotes the height of strip S_ℓ after packing the next shelf a_{i_0+1}. Then $h^+ > h^- + b_{i_0+1}$ or $h^+ \leq h^- + b_{i_0+1}$. In the first case we conclude $h(\ell)^\star \in [h^-, h^+]$ for all $\ell \in [N]$. Since $|h^+ - h^-| \leq h_{\max}$ by induction hypothesis, the assertion follows. In the second case the assertion is true, because since $b_{i_0+1} \leq h_{\max}$ we have $h(\ell)^\star \in [h^-, h^- + h_{\max}]$ for all $\ell \in [N]$. □

This leads to a further result about rectangles with small width.

Theorem 3.5. *For a set of rectangles $\mathcal{R}$ with width $\leq \varepsilon$ for $\varepsilon \in (0,1]$ we obtain by Algorithm 2 a packing into N strips with height less than*

$$\frac{1}{1-\varepsilon}\mathrm{OPT}_{MSP}(\mathcal{R}) + 2h_{\max}. \tag{3.1}$$

Proof. Let $SIZE(S_\ell)$ denote the total area of the rectangles packed into S_ℓ. We consider the strip $S_{\min}$ with $SIZE(S_{\min}) = \min_{\ell \in [N]} SIZE(S_\ell)$. Let $a_1, \ldots, a_r$ be the ordered sequence of shelves constructed by Algorithm 2. Furthermore, let b_i and b'_i be the heights of the first and last rectangle placed in shelf $a_i, i \in \{1, \ldots, r\}$. We have $b_1 \geq b'_1 \geq \ldots \geq b_r \geq b'_r$. A shelf is closed when the next rectangle does not fit completely on the shelf. Notice that all narrow rectangles on shelf a_i have heights $\geq b'_i$. For $i \in \{1, \ldots, r-1\}$ the total width of the rectangles packed on shelf a_i is larger than $(1-\varepsilon)$. Therefore on these shelves a_i an area of $(1-\varepsilon)b'_i$ is fully covered by rectangles and thus $SIZE(S_{min}) \geq \sum_{i=1}^{r-1}(1-\varepsilon)b'_i$.
Let $h(S_{\min})$ denote the height of strip $S_{\min}$. With $SIZE(S_{\min}) \leq {}^{SIZE(\mathcal{R})}/_{N}$ and Lemma 3.4 we conclude

$$\begin{aligned} h_{\mathrm{MSP}} &\leq h(S_{\min}) + h_{\max} = \sum_{l=1}^{r} b_r + h_{\max} \leq \sum_{l=1}^{r-1} b'_r + 2h_{\max} \\ &\leq \frac{SIZE(S_{\min})}{1-\varepsilon} + 2h_{\max} \leq \frac{SIZE(\mathcal{R})}{N(1-\varepsilon)} + 2h_{\max} \leq \frac{\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R})}{1-\varepsilon} + 2h_{\max}. \end{aligned}$$

□

4. A 2-Approximation for MSP

Since there is no approximation algorithm for MSP with absolute ratio smaller than 2 (unless $\mathcal{P} = \mathcal{NP}$), a 2-approximation is best possible for MSP. In this section we show the following theorem.

Theorem 4.1. *For any $N \in \mathbb{N}$ there is a polynomial-time algorithm for MSP with absolute approximation ratio* 2*.*

To handle different sizes of N, different subroutines are used to obtain a 2-approximation. For $N = 1$ MSP is equivalent to Strip Packing and we can use the algorithm of Steinberg [62] or Schiermeyer [58] with absolute approximation ratio 2.

Theorem 4.2 (Steinberg [62])**.** *Let $\mathcal{R}$ be a set of n rectangles with heights h_j and widths w_j, $j \in [n]$ and Q be a rectangle with width u and height v. Let $h := \max_{j\in[n]} h_j$ and $w := \max_{j\in[n]} w_j$. If the following inequalities hold,*

$$w \leq u, \qquad h \leq v, \qquad 2SIZE(\mathcal{R}) \leq uv - (2w - u)_+(2h - v)_+ \tag{4.1}$$

then it is possible to pack $\mathcal{R}$ into the rectangle Q. (As usual, $x_+ = \max(x, 0)$).

For N sufficiently large, the algorithm of Caprara [11] for 2-Dimensional Bin Packing (2DBP) with asymptotic ratio 1.69... and running time $\mathcal{O}(n \log(n)) + T$ (where T is the running time of an AFPTAS for Bin Packing) gives us a 2-approximation:

The problem 2DBP is the 2-dimensional generalization of Bin Packing, where a set of rectangles with widths and heights bounded by 1 has to be packed into a minimum number of unit squares, called bins. In [59] it is shown that Caprara's algorithm already gives a 2-approximation for 2DBP if $\mathrm{OPT}_{\mathrm{2DBP}}(\mathcal{R}) \geq 1446$ for an instance $\mathcal{R}$. It should be possible to reduce this value for $\mathrm{OPT}_{\mathrm{2DBP}}(\mathcal{R})$ since it is due to the additive constant of the

AFPTAS for Bin Packing used. Recently, Jansen and Kraft presented an improved AFPTAS with small additive factor $\mathcal{O}(\log^2(1/\varepsilon))$ [36]. Unfortunately, the algorithm is quite nested so that it is complicated to extract the exact additive factor directly. Possibly, it can be found in the upcoming full version of the paper.

Given an instance $\mathcal{R}$ of MSP we can transform it into an instance $\widetilde{\mathcal{R}}$ of 2DBP with $\mathrm{OPT}_{\mathrm{2DBP}}(\widetilde{\mathcal{R}}) \leq N$ by scaling the height of any rectangle by $1/\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R})$. The value $\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R})$ can be found via binary search in $\mathcal{O}(\log(nh_{\max}))$ iterations (see Lemma 4.3 below). The algorithm for $N \geq 1446$ works as follows. For each candidate value for $\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R})$ we transform the instance $\mathcal{R}$ into $\widetilde{\mathcal{R}}$. If Capraras algorithm outputs a packing of $\widetilde{\mathcal{R}}$ into less than $2N$ bins we terminate. Stacking any two bins on top of each other and rescaling gives us a 2-approximation for MSP.

Lemma 4.3. *Let $\mathcal{R}$ be an instance of MSP. Binary Search needs at most $\mathcal{O}(\log(nh_{\max}))$ iteration to find the optimum height $\mathrm{OPT}_{MSP}(\mathcal{R})$.*

Proof. For each height $h_j \in \mathbb{Q}_{\geq 0}$, $j \in \mathcal{R}$, there exist $q_j, p_j \in \mathbb{N}$ with $h_j = {}^{p_j}/_{q_j}$. We have $Qh_j \in \mathbb{N}$, where $Q = \prod_{j=1}^{n} q_j$. Since $\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R})$ is equal to the sum of heights of rectangles in $\mathcal{R}$, we also have $Q \cdot \mathrm{OPT}(\mathcal{R}) \in \mathbb{N}$. So for the height ν of an optimal solution we conclude that $Qh_{\max} \leq Q\nu \leq Qnh_{\max}$. Since $\log_2(Qnh_{\max}) = \sum_{j\in\mathcal{R}} \log(q_j) + \log(n) + \log(h_{\max}) \leq |\mathcal{R}|$, Binary Search takes at most $\mathcal{O}(\log(nh_{\max}))$ iterations. □

In case $2 \leq N < 1446$ there is something more to do. Here we use a PTAS for Rectangle Packing with Area Maximization (RPA) found by Bansal et al. [3]. In RPA we are given a set $\mathcal{R}$ of n rectangles with widths w_j and heights h_j less than 1 and a bin of unit size. The objective is to find a feasible packing of a subset $\mathcal{R}'$ of the rectangles into the bin while maximizing the total area of the rectangles in $\mathcal{R}'$. Let us first consider $N = 2$. Algorithm 3 gives us a 2-approximation in this case.

If $2 < N < 1446$ we use an extended version of the PTAS for RPA in [3] for several strips. Moreover, one can show the following

Theorem 4.4. *Given a constant number N of bins, a fixed value ε and a set of n rectangles $\mathcal{R}$ there is a polynomial time algorithm $A_{N,\varepsilon}$ that finds a subset $\mathcal{R}' \subset \mathcal{R}$ with total area at least $(1-\varepsilon)$ times the optimal value and a packing of $\mathcal{R}'$ into N bins or decides that no such subset exists.*

Algorithm 3 2-Approximation for MSP if N=2

Input: $\mathcal{R}$

Output: A packing of $\mathcal{R}$ into N strips

1: **for all** current guess ν of the height of an optimum solution $\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R})$ **do**

2: Scale the heights of the rectangles in $\mathcal{R}$ by $1/2\nu$ so that the packing corresponding to the optimum fits into a unit sized bin.

3: The set of resulting rectangles $\mathcal{R}_\nu$ is now considered as an instance of RPA with $\mathrm{OPT}_{\mathrm{RPA}}(\mathcal{R}_\nu) = SIZE(\mathcal{R}_\nu) \leq 1$. Apply the algorithm in [3] with accuracy $\varepsilon = 1/2$ and find a packing of a subset $\mathcal{R}'_\nu \subset \mathcal{R}_\nu$ with total area at least $(1-\varepsilon)SIZE(\mathcal{R}_\nu)$ into a unit sized bin. By rescaling the rectangles in $\mathcal{R}'_\nu$ obtain a packing for the first strip with height at most $2v$.

4: Since $SIZE(\mathcal{R}_\nu) \leq 1$ the remaining items in $\mathcal{R}_\nu \backslash \mathcal{R}'_\nu$ have total area $SIZE(\mathcal{R}_\nu \backslash \mathcal{R}'_\nu) \leq \varepsilon SIZE(\mathcal{R}_\nu) \leq 1/2$. Furthermore, any rectangle has height less than $1/2$. Therefore we can pack them with Steinberg's algorithm into a unit sized bin. Rescaling gives us a second strip of height at most 2ν for the correct choice of ν.

5: **end for**

For a detailed proof we refer to [59]. Together with the next assertion we can state that Algorithm 4 gives us a 2-approximation.

Lemma 4.5. *If $N \geq 3$ and $\mathcal{R}$ is a set of n rectangles with total area $SIZE(\mathcal{R}) \leq N/4$, then there exists a packing of $\mathcal{R}$ into N bins.*

Proof. Since for $h, w, u = 1$ and $v = N/2$ the inequalities (4.1) hold, we can apply Steinberg's algorithm. By this we get a solution for SP with height at most $N/2$. By drawing $\lceil N/2 \rceil + 1$ horizontal lines with distance one through the strip starting at the bottom we divide the strip into $\lceil N/2 \rceil$ bins of height one and $\lceil N/2 \rceil - 1$ sets of cut items. Packing each set of fractional items into an extra bin we use at most $\lceil N/2 \rceil + \lceil N/2 \rceil - 1 \leq N$ bins. □

The running time for both Algorithm 3 and Algorithm 4 is dominated by the running time of the Algorithm for RPA, which is doubly exponential in ε^{-1}, i.e. for $\varepsilon = \frac{1}{4}$ we have $\mathcal{O}(n^{256})$. It is not known wether RPA can be solved approximately in time singly exponential in $1/\varepsilon$.

Algorithm 4 2-Approximation for MSP if $2 < N < 1446$

Input: $\mathcal{R}$

Output: A packing of $\mathcal{R}$ into N strips

1: **for all** current guess ν of the height of an optimum solution $\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R})$ **do**

2: Scale the heights of the rectangles of $\mathcal{R}$ by $1/\nu$ so that the corresponding packing fits into N bins of height and width one.

3: The set of resulting rectangles $\mathcal{R}_\nu$ is now considered as an instance of RPA with $\mathrm{OPT}_{\mathrm{RPA}}(\mathcal{R}_\nu) = SIZE(\mathcal{R}_\nu)$. With the algorithm in [59] find for accuracy $\varepsilon := 1/4$ a packing of a subset $\mathcal{R}'_\nu \subset \mathcal{R}_\nu$ with total area $(1-\varepsilon)SIZE(\mathcal{R}_\nu)$ into N bins of unit size and. By rescaling the rectangles of $\mathcal{R}'_\nu$ we get N bins of height ν.

4: For the total area of the remaining rectangles in $\mathcal{R}_\nu \backslash \mathcal{R}'_\nu$ we have $SIZE(\mathcal{R}_\nu \backslash \mathcal{R}'_\nu) = \varepsilon SIZE(\mathcal{R}_\nu) \leq N/4$. Pack those rectangles according to Lemma 4.5 into N bins and rescale the rectangles. This results again in N bins of height at most ν.

5: Stack any two bins on top of each other and get a solution with height at most 2ν for the correct choice of ν.

6: **end for**

5. Improved Strip Packing Algorithm

In this chapter we roughly describe the AFPTAS of Kenyon and Rémila [45] for Strip Packing and present an improved version. We will use this improved algorithm later as a subroutine. Kenyon and Rémila showed the following

Theorem 5.1 (Kenyon & Rémila [45])**.** *There is an algorithm that for a set $\mathcal{R}$ of n rectangles with widths and heights ≤ 1 and for any accuracy $\varepsilon > 0$ generates a packing of $\mathcal{R}$ into a strip with height at most $(1+\varepsilon)\mathrm{OPT}_{SP}(\mathcal{R}) + (4M+1)h_{\max}$ for $M = (\frac{2+\varepsilon}{\varepsilon})^2 = \mathcal{O}(1/\varepsilon^2)$. The algorithm has running time polynomial in $\langle\mathcal{R}\rangle$ and ε^{-1}.*

By applying a different rounding technique than in the original algorithm we achieve a reduction of the additive factor from $\mathcal{O}(1/\varepsilon^2)h_{\max}$ to $\mathcal{O}(1/\varepsilon \log(1/\varepsilon))h_{\max}$.

Theorem 5.2. *For a set $\mathcal{R}$ of n rectangles with width and heights ≤ 1 and any accuracy $\varepsilon > 0$ Algorithm 7 generates a packing of $\mathcal{R}$ into a strip with height st most $(1+\varepsilon)\mathrm{OPT}_{SP}(\mathcal{R}) + (4M+1)h_{\max}$, where $M = \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$. The running time is in $\mathcal{O}(\varepsilon^{-6}\log(1/\varepsilon) + n\log n)$.*

5.1 The AFPTAS of Kenyon and Rémila

We give here only a rough description of the algorithm of Kenyon and Rémila, for our purpose this is sufficient. For an overall description and fully detailed proofs refer to [45] and [32].

For $\varepsilon' = \frac{\varepsilon}{2+\varepsilon}$ the rectangles in $\mathcal{R}$ are partitioned into wide $\mathcal{R}_{wide} := \{j \in \mathcal{R} | w_j > \varepsilon'\}$ and narrow rectangles $\mathcal{R}_{narrow} := \{j \in \mathcal{R} | w_j \leq \varepsilon'\}$. The set $\mathcal{R}_{wide}$ of wide rectangles is rounded to a set $\mathcal{R}_{sup}$ with only $\mathcal{O}(1/\varepsilon^2)$ different widths. The grouping and rounding technique used for the wide

rectangles is called "geometric rounding" and is described by Algorithm 5 and in Figure 5.1.

Algorithm 5 Rounding I

Input: $\mathcal{R}_{wide}$
Output: $\mathcal{R}_{sup}$

1: Order the rectangles in $\mathcal{R}_{wide}$ by non-increasing widths and pack them left-aligned on a stack with height $H := H(\mathcal{R}_{wide})$.
2: Let $M := (1/\varepsilon')^2$.
3: Draw $M-1$ horizontal lines through the stack with distance H/M starting at the bottom. Therefore we get M so-called *threshold* rectangles. A rectangle is a threshold rectangle if it either with its interior or with its lower edge intersects a line at height iH/M, $i \in \{0, 1, \ldots, M-1\}$.
4: **for all** $i \in \{0, 1, \ldots, M-1\}$ **do**
5: Round up the width of each rectangle between the lines iH/M and $(i+1)H/M$ to the width of the ith threshold rectangle. The widths of the rectangles below the first line are rounded up to the width of the undermost rectangle in the stack.
6: **end for**
7: Let $\mathcal{R}_{sup}$ denote the list of rounded rectangles.

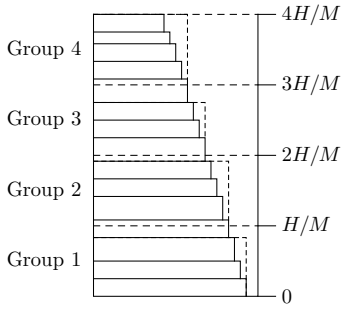

Figure 5.1: Rounding $\mathcal{R}_{wide}$ with Algorithm 5.

The main part of the algorithm is to produce a FRACTIONAL STRIP PACKING for the rectangles in $\mathcal{R}_{sup}$ using a linear program formulation for FRACTIONAL BIN PACKING. In doing so we introduce configurations $C_i := \{\alpha_{ij} : w_j | j \in [M] \wedge \sum_{j=1}^{M} \alpha_{ij} w_j \leq 1\}$, i.e. multisets of rounded widths where α_{ij} denotes the number of occurrence of width w_j in C_i, so that they fit next to each other into the strip. Then the fractional packing of the wide rectangle can be formulated as the following linear program.

$$\begin{aligned} &\min \sum_{i=1}^{q} x_i \\ &\text{s.t.} \sum_{j=1}^{q} \alpha_{ij} x_i \geq \beta_j \text{ for all } j \in \{1, \ldots, M\} \\ &\qquad x_i \geq 0 \text{ for all } i \in \{1, \ldots, q\}. \end{aligned} \tag{5.1}$$

The variable x_i indicates the height of configuration C_i, β_j is the total height of rectangles of width w_j in $\mathcal{R}_{sup}$ and q denotes the number of possible configurations. A feasible solution x of (5.1) corresponds to a solution of FRACTIONAL STRIP PACKING with input $\mathcal{R}_{sup}$. Note that $rank((\alpha_{ij})_{ij}) \leq M$ and hence a basic solution x of (5.1) has at most M nonzero entries. Therefore the FRACTIONAL STRIP PACKING for $\mathcal{R}_{sup}$ with height $FSP(\mathcal{R}_{sup})$ can be converted into an integral one increasing the height only by $2Mh_{\max}$. Moreover, the following holds.

Lemma 5.3. *[45]*

1. *The number of different width in $\mathcal{R}_{sup}$ is bounded by $M = \mathcal{O}(1/\varepsilon^2)$.*
2. $FSP(\mathcal{R}_{sup}) \leq FSP(\mathcal{R}_{wide}) \left(1 + \frac{1}{M\varepsilon'}\right)$
3. $SIZE(\mathcal{R}_{sup}) \leq SIZE(\mathcal{R}_{wide}) \left(1 + \frac{1}{M\varepsilon'}\right)$

Thus, an integral packing of $\mathcal{R}_{wide}$ with height $FSP(\mathcal{R}_{wide})(1 + \frac{1}{M\varepsilon'}) + 2Mh_{\max}$ is achieved. Finally, the narrow rectangles in $\mathcal{R}_{narrow}$ are added to the remaining space next to the integral packing and on top of it with a slightly modified NFDH heuristic. Since $FSP(\mathcal{R}_{wide}) \leq \mathrm{OPT}_{\mathrm{SP}}(\mathcal{R})$ with some small computations Theorem 5.1 follows.

5.2 Improved Geometric Rounding

Our improved algorithm uses a different geometric rounding technique. For $\varepsilon > 0$ we partition the rectangles into wide $\mathcal{R}_{wide} := \{j \in \mathcal{R} | w_j > \varepsilon/2\}$ and narrow ones $\mathcal{R}_{narrow} := \mathcal{R} \setminus \mathcal{R}_{wide}$. We may assume that $\varepsilon SIZE(\mathcal{R}_{wide}) > 2(\lfloor \log(2/\varepsilon) \rfloor + 1)h_{\max}$. Otherwise we can achieve a packing of $\mathcal{R}_{wide}$ with

height less than

$$2SIZE(\mathcal{R}_{wide}) + h_{\max} \leq {}^{4}/_{\varepsilon}(\lfloor \log(2/\varepsilon) \rfloor + 1)h_{\max} + h_{\max} = \mathcal{O}(1/\varepsilon \log(1/\varepsilon))h_{\max}$$

by simply using the NFDH heuristic. We apply geometric grouping with parameter k introduced by Karmarkar and Karp in [42] for FRACTIONAL BIN PACKING to $\mathcal{R}_{wide}$ and obtain a set of rounded rectangles $\mathcal{J} \cup \mathcal{J}'$ as described in Algorithm 6.

Algorithm 6 Rounding II

Input: $\mathcal{R}_{wide}$
Output: $\mathcal{J} \cup \mathcal{J}'$

1: Let $k := \left\lfloor \frac{SIZE(\mathcal{R}_{wide})\varepsilon}{(\lfloor \log(2/\varepsilon) \rfloor + 1)h_{\max}} \right\rfloor$
2: **for all** $t \in \{0, 1 \ldots, \lfloor \log(2/\varepsilon) \rfloor\}$ **do**
3: partition $\mathcal{R}_{wide}$ into sets $W_t := \{j \in \mathcal{R}_{wide} | w_j \in (2^{-(t+1)}, 2^{-t}]\}$.
4: For each list W_t order the rectangles by non-increasing widths and pack them left-aligned on a stack.
5: Starting at the baseline draw horizontal lines at height $(ik2^t)h_{\max}$ for $i = \{0, \ldots, \lfloor h(W_t)/(k2^t h_{\max}) \rfloor\}$ through the stack. For every rectangle whose interior is cut by such a line we introduce two new rectangles, so that the stack is divided into $q(t) = \lfloor h(W_t)/(k2^t h_{\max}) \rfloor + 1$ groups $G_1(t), \ldots, G_{q(t)}(t)$ all having total height exactly $k2^t h_{\max}$ except maybe the last group $G_{q(t)}(t)$ of the narrowest rectangles having height $< k2^t h_{\max}$.
6: In each group $G_i(t)$ we round up the width of every rectangle to the width of the widest rectangles contained in this group and obtain $G'_i(t)$.
7: **end for**
8: Define $J_t := \bigcup_{i=2}^{q(t)} G'_i(t)$. Further let $\mathcal{J} := \bigcup_t J_t$, $\mathcal{J}' := \bigcup_t G'_1(t)$

For arbitrary sets of (indexed) rectangles $\mathcal{R}''$, $\mathcal{R}'$ we define a partial order $\leq_g$, so that $\mathcal{R}'' \leq_g \mathcal{R}'$ if and only if the stack built from the rectangles in $\mathcal{R}''$ can be geometrically included into the one of $\mathcal{R}'$. We immediately obtain $\mathcal{R}_{wide} \leq_g \mathcal{R}_{sup}$. For any set of rectangles $\mathcal{R}$ let $FSP(\mathcal{R})$ denote the height of an optimum solution of FRACTIONAL STRIP PACKING with input $\mathcal{R}$.

Lemma 5.4. *The rounded instance $\mathcal{J} \cup \mathcal{J}'$ has the following properties.*

1. *The number of different widths is bounded by*

$$M = 5/\varepsilon(\lfloor \log(2/\varepsilon) \rfloor + 1) = \mathcal{O}(1/\varepsilon \log(1/\varepsilon)).$$

2. $SIZE(\mathcal{R}_{wide}) \leq SIZE(\mathcal{J} \cup \mathcal{J}') \leq (1+\varepsilon)SIZE(\mathcal{R}_{wide})$.

3. $FSP(\mathcal{R}_{wide}) \leq FSP(\mathcal{J} \cup \mathcal{J}') \leq (1+\varepsilon)FSP(\mathcal{R}_{wide})$.

Proof. We have that

$$\begin{aligned} SIZE(W_t) &\geq 2^{-(t+1)} h(W_t) \\ &\geq 2^{-(t+1)}(q(t)-1)k2^t h_{\max} \geq \frac{kh_{\max}(q(t)-1)}{2} \end{aligned}$$

and thus $q(t) \leq \frac{2SIZE(W_t)}{kh_{\max}} + 1$ for all t.

With $\varepsilon SIZE(\mathcal{R}_{wide}) > 2(\lfloor \log(2/\varepsilon) \rfloor + 1)h_{\max}$ we conclude that

$$\begin{aligned} M = \sum_{t=0}^{\lfloor \log(2/\varepsilon) \rfloor} q(t) &\leq \sum_{t=0}^{\lfloor \log(2/\varepsilon) \rfloor} \frac{2SIZE(W_t)}{kh_{\max}} + 1 \\ &\leq \frac{2}{kh_{\max}} SIZE(\mathcal{R}_{wide}) + \lfloor \log(2/\varepsilon) \rfloor + 1. \\ &\leq \frac{2SIZE(\mathcal{R}_{wide})}{h_{\max}\left\lfloor \frac{\varepsilon SIZE(\mathcal{R}_{wide})}{(\lfloor \log(2/\varepsilon) \rfloor + 1)h_{\max}} \right\rfloor} + \lfloor \log(2/\varepsilon) \rfloor + 1 \\ &\leq \frac{2SIZE(\mathcal{R}_{wide})}{h_{\max}\left(\frac{\varepsilon SIZE(\mathcal{R}_{wide})}{(\lfloor \log(2/\varepsilon) \rfloor + 1)h_{\max}} - 1\right)} + \lfloor \log(2/\varepsilon) \rfloor + 1 \\ &= \frac{2SIZE(\mathcal{R}_{wide})(\lfloor \log(2/\varepsilon) \rfloor + 1)h_{\max}}{h_{\max}(\varepsilon SIZE(\mathcal{R}_{wide}) - (\lfloor \log(2/\varepsilon) \rfloor + 1)h_{\max})} \\ &\qquad + \lfloor \log(2/\varepsilon) \rfloor + 1 \\ &< \frac{2SIZE(\mathcal{R}_{wide})(\lfloor \log(2/\varepsilon) \rfloor + 1)}{\frac{\varepsilon}{2} SIZE(\mathcal{R}_{wide})} + \lfloor \log(2/\varepsilon) \rfloor + 1 \\ &= 4/\varepsilon(\lfloor \log(2/\varepsilon) \rfloor + 1) + \lfloor \log(2/\varepsilon) \rfloor + 1 \\ &\leq 5/\varepsilon(\lfloor \log(2/\varepsilon) \rfloor + 1). \end{aligned}$$

The total height of the rectangles in every group $G_1(t)$ is at most $k2^t h_{\max}$. Since each of the rectangles in $G_1(t)$ has width at most 2^{-t} we

have $SIZE(G_1(t)) \leq kh_{\max}$. Then

$$SIZE(\mathcal{R}_{wide}) \leq SIZE(\mathcal{J} \cup \mathcal{J}') \leq SIZE(\mathcal{J}) + (\lfloor \log(2/\varepsilon) \rfloor + 1)kh_{\max}$$

since there are $\lfloor \log(2/\varepsilon) \rfloor + 1$ groups $G_1(t)$ in $\mathcal{J}'$. With our choice of k the second assertion follows.

For all $t \in \{0, 1 \ldots, \lfloor \log(2/\varepsilon) \rfloor\}$ we have by construction

$$G'_1(t) \geq_g G_1(t) \geq_g G'_2(t) \geq_g \ldots \geq_g G_{q(t)}(t)$$

and thus

$$\mathcal{J} \cup \mathcal{J}' = \bigcup_t \bigcup_{i=1}^{q(t)} G'_i(t) \geq_g \bigcup_t W_t \geq_g \bigcup_t \bigcup_{i=2}^{q(t)} G'_i(t) = \mathcal{J}.$$

We have $FSP(\mathcal{J} \cup \mathcal{J}') \leq FSP(\mathcal{J}) + FSP(\mathcal{J}')$. Since at least 2^t of the rectangles in $G'_1(t)$ fit next to each other into the strip we conclude also $FSP(G'_1(t)) \leq kh_{\max}$. Thus, we have

$$FSP(\mathcal{J}') \leq (\lfloor \log(2/\varepsilon) \rfloor + 1)kh_{\max} \leq \varepsilon SIZE(\mathcal{R}_{wide}) \leq \varepsilon FSP(\mathcal{R}_{wide}).$$

We conclude

$$FSP(\mathcal{J} \cup \mathcal{J}') \leq FSP(\mathcal{J}) + \varepsilon FSP(\mathcal{R}_{wide}) \leq (1 + \varepsilon)FSP(\mathcal{R}_{wide}).$$

□

With the rounding technique presented in this section the additive factor of the AFPTAS for Strip Packing improves. The complete algorithm is given below. For the rest of the proof of Theorem 5.2 similar techniques are used as for the proof of Theorem 5.1 which can be found in [32]. The techniques used for the converting process can also be found fully detailed later in Chapter 6 in the proof of Theorem 6.6.

5.3 Running Time of the Algorithm

It is sufficient to solve the linear program (5.1) used in the algorithm approximately. As described in [32] the fractional Strip Packing with M different width can be formulated as a Max-Min Resource Sharing

Algorithm 7 Improved Strip Packing

Input: $\mathcal{R}$, $\varepsilon > 0$

Output: A packing of $\mathcal{R}$ into N strips

1: Partition $\mathcal{R}$ into $\mathcal{R}_{wide} := \{j \in \mathcal{R} | w_j > \varepsilon/2\}$ and
2: $\mathcal{R}_{narrow} := \{j \in \mathcal{R} | w_j \leq \varepsilon/2\}$.
3: Round the widths of the rectangles in $\mathcal{R}_{wide}$ with Algorithm 6 and obtain $\mathcal{J} \cup \mathcal{J}'$ with only $M = \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$ different widths.
4: Solve the linear program (5.1) with input $\mathcal{J} \cup \mathcal{J}'$.
5: Construct a feasible solution for $\mathcal{J} \cup \mathcal{J}'$ with height at most $FSP(\mathcal{J} \cup \mathcal{J}') + 2Mh_{\max}$.
6: Use modified NFDH to pack the rectangles in $\mathcal{R}_{narrow}$ into the remaining space and on top of the strips.

problem that can be solved approximately in time

$$\mathcal{O}(M(1/\varepsilon^2 + \log(M)) \max\{M + 1/\varepsilon^3, M \log\log(M/\varepsilon)\}).$$

For $M = \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$ this gives a running time of $\mathcal{O}(\varepsilon^{-6} \log(1/\varepsilon))$ for solving (5.1) approximately. As it will be shown in Chapter 6, for constructing a fractional packing and converting it into an integral one we need time $\mathcal{O}(M^2 n) = \mathcal{O}(\varepsilon^{-2} \log^2(1/\varepsilon) n)$. Thus, the running time of Algorithm 7 is in $\mathcal{O}(\varepsilon^{-6} \log(1/\varepsilon) + n \log n)$. This improves the running time of the original algorithm where the Max-Min Resource Sharing is solved for $M = \mathcal{O}(1/\varepsilon^2)$ which gives a running time of $\mathcal{O}(\varepsilon^{-7} + n \log n)$. Eventually, the running time could be further improved: The block problem of the corresponding Max-Min Resource Sharing problem is an unbounded knapsack problem with $M = \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$ item sizes. If it is possible to solve the knapsack problem with running time better than $\mathcal{O}(M + 1/\varepsilon^3)$ this would decrease the running time.

6. Approximation Schemes for MSP

In this chapter we present an AFPTAS for Multiple Strip Packing. The algorithm is based on a generalization of the improved version of the AFPTAS for Strip Packing by Kenyon and Rémila given in Chapter 5. For sufficiently large values of N the additive factor can even be more reduced.

Theorem 6.1. *There is an algorithm that for a set $\mathcal{R}$ of n rectangles with widths and heights ≤ 1 and for any accuracy $\varepsilon > 0$ generates a packing of $\mathcal{R}$ into N strips with height at most $(1+\varepsilon)\mathrm{OPT}_{MSP}(\mathcal{R}) + \mathcal{O}(1/\varepsilon \log(1/\varepsilon))h_{\max}$. For sufficiently large N, namely $N \in \Omega(1/\varepsilon^2 \log(1/\varepsilon))$, the height of the packing can be reduced to $(1+\varepsilon)\mathrm{OPT}_{MSP}(\mathcal{R}) + \mathcal{O}(1)h_{\max}$. The algorithm has running time $\mathcal{O}(\varepsilon^{-6}\log(1/\varepsilon) + n\log n)$.*

The number N of strips does not show up in the running time since we may assume $n \geq N$. For $n < N$ we only need to consider the first n strips to pack n rectangles.

6.1 A first Approach

As in the proof of Theorem 3.1 in Chapter 3 we can easily convert a packing generated for one strip into a packing for N strips by cutting it into N parts of nearly the same height and distributing these parts among N strips. This leads us to the definition of Algorithm 8

Theorem 6.2. *For a set $\mathcal{R}$ of n rectangles with widths and heights ≤ 1 and accuracy $\varepsilon > 0$ Algorithm 8 generates a packing into N strips with height at most $(1+\varepsilon)\mathrm{OPT}_{MSP}(\mathcal{R}) + \mathcal{O}(1/\varepsilon \log(1/\varepsilon))h_{\max}$ and running time $\mathcal{O}(\varepsilon^{-6}\log(1/\varepsilon) + n\log n + N)$.*

Proof. Let $A(\mathcal{R}, \varepsilon)$ denote the ouput of Algorithm 7. By Step 2 every strip

Algorithm 8

Input: $\mathcal{R}$, $\varepsilon > 0$
Output: A packing of $\mathcal{R}$ into N strips
1: Pack the rectangles with Algorithm 7 into a single strip S.
2: **for all** $\ell \in \{0, 1, \ldots, N\}$ **do**
3: Draw a horizontal line through S at height $\ell A_\varepsilon^{KR}(\mathcal{R})/N$.
4: **end for**
5: **for all** $\ell \in \{0, 1, \ldots, N-1\}$ **do**
6: Pack all items intersecting the ℓth line and all items between the ℓth and $(\ell+1)$th lines into strip $S_{\ell+1}$.
7: **end for**

S_ℓ, $\ell \in [N]$ has height

$$\begin{aligned} h_i &\leq \frac{A(\mathcal{R}, \varepsilon)}{N} + h_{\max} \\ &\leq \frac{(1+\varepsilon)\mathrm{OPT}_{\mathrm{SP}}(L) + (4M+1)h_{\max}}{N} + h_{\max} \\ &\overset{N \geq 2}{\leq} \frac{(1+\varepsilon)\mathrm{OPT}_{\mathrm{SP}}(\mathcal{R})}{N} + (2M+1)h_{\max} + h_{\max} \\ &\leq (1+\varepsilon)\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R}) + (2M+2)h_{\max}, \end{aligned}$$

where the last inequality holds because $\mathrm{OPT}_{\mathrm{SP}}(\mathcal{R})/N$ is a lower bound for $\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R})$. The running time is dominated by the time used for Algorithm 7. For $N = \mathcal{O}(\varepsilon^{-2}\log(1/\varepsilon))$ or with the assumption $n \geq N$ it can be bounded by $\mathcal{O}(\varepsilon^{-6}\log(1/\varepsilon) + n\log n)$. □

6.2 Packing into a Large Number of Strips

Now let us consider the case that the number N of strips is very large. In this case it is possible to improve the additive constant to $\mathcal{O}(1)$ by balancing the configurations. Choose $\delta := \frac{2\varepsilon}{7+\varepsilon}$ and $N > 10/\delta^2(\lfloor \log(2/\delta) \rfloor + 1) = \Omega(1/\varepsilon^2 \log(1/\varepsilon))$. We divide the list of rectangles L into a list of narrow rectangles $\mathcal{R}_{narrow} := \{j \in \mathcal{R} | w_j \leq \delta/2\}$ and a list of wide rectangles $\mathcal{R}_{wide} := \{j \in \mathcal{R} | w_j > \delta/2\}$. Using Algorithm 6 (for $\varepsilon = \delta$) we round $\mathcal{R}_{wide}$ to an instance $\mathcal{J} \cup \mathcal{J}'$ with $M \leq 5/\delta(\lfloor \log(2/\delta) \rfloor + 1) = \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$ different widths (compare to Lemma 5.4). The first objective is to create a fractional packing of the rounded wide rectangles in $\mathcal{J} \cup \mathcal{J}'$ into N strips. To do this we approximately solve the linear program (5.1) for $\mathcal{J} \cup \mathcal{J}'$ as

described in Section 5.3. Let $h_0 := FSP(\mathcal{J} \cup \mathcal{J}')/N$. We first show there is a fractional packing of $\mathcal{J} \cup \mathcal{J}'$ that contains only $\mathcal{O}(1/\varepsilon \log(1/\varepsilon))$ different configurations and has height at most $(1+\delta)h_0$.

Lemma 6.3. *Let $x = (x_1, \ldots, x_q)$ be a solution of (5.1) for input $\mathcal{J} \cup \mathcal{J}'$ with at most $m \leq M$ nonzero entries $x_1, \ldots, x_m$. For $N > {}^{10}/_{\delta^2}(\lfloor \log(2/\delta) \rfloor + 1)$ we get a fractional packing of $\mathcal{J} \cup \mathcal{J}'$ into N strips with height at most $(1+\delta)h_0$ and at most $m' \leq 2M$ different configurations, so that there are at most 2 different configurations per strip. This can be done in time $\mathcal{O}(M^2 n + MN)$.*

Proof. First remember that there exists a solution x with at most M non-zero entries as $rank((\alpha_{ij})_{ij}) \leq M$. We fractionally pack the rectangles into the configurations. Imagine each configuration C_i as a bin with height x_i and width c_i and divide it into α_{ij} columns of widths w_j and height x_i. For each non-zero configuration C_i fractionally pack the rectangles in $\mathcal{J} \cup \mathcal{J}'$ of width w_j in a greedy manner into the columns of width w_j upto exact height x_i, starting with $j = 1$. This takes time in $\mathcal{O}(M^2 n)$ since there are at most M^2 columns in which at most n rectangles are packed. In this way each column contains a sequence of rectangles, which completely fits inside the column and possibly the top part of a rectangle that started in a previous column and the bottom part of a rectangle that is too tall to fit into this column. Assume w.l.o.g. $w_1 \geq w_2 \geq \ldots \geq w_M$. We start filling the columns for each width with the rounded fraction of the wide rectangle in $\mathcal{R}_{wide}$ that is split into a rectangle of width w_j and the next wider width w_{j-1} by the rounding Algorithm 6, if such a fraction exists. We finish packing each width w_j with the fraction of the wide rectangle in $\mathcal{R}_{wide}$ that is split into a rectangle of width w_j and the next smaller width w_{j+1}, if such a fraction exists.
Since $\sum_{i=1}^{m} \alpha_{ij} x_i \geq \beta_j$ there is maybe more than enough space for the rectangles of width w_j in the configurations. In this case we distribute the rectangles among the columns and delete the additional space. So we split a configuration C_i into two parts, one of the old type where the columns of width w_j are completely filled and one without columns of width w_j. This case may happen at most M times. So we have in total $m' = m + M \leq 2M$ configurations $C_1, \ldots C_{m'}$ with nonzero heights $x_1, \ldots, x_{m'}$.
Notice that there exist configurations with height larger or equal h_0, since otherwise we conclude $\sum_{i=1}^{m'} x_i < m' h_0 \leq 2M h_0 = {}^{10}/_{\delta}(\lfloor \log(2/\delta) \rfloor + 1)h_0 <$

Nh_0, which is a contradiction.

Consider a configuration C_i, $i \in [m']$. If $x_i \geq h_0$ we allocate an area of height h_0 for configuration C_i in $\lfloor x_i/h_0 \rfloor$ empty strips. If then $x_i/h_0 - \lfloor x_i/h_0 \rfloor \leq \delta h_0$, we assign to C_i additional space with height $(x_i/h_0 - \lfloor x_i/h_0 \rfloor)$ in a strip, that has already height h_0. If $x_i/h_0 - \lfloor x_i/h_0 \rfloor > \delta h_0$, we divide $(x_i/h_0 - \lfloor x_i/h_0 \rfloor)$ into at most $1/\delta$ horizontal slides with height less than or equal to δh_0. So we assign to C_i additional space of height δh_0 in no more than $1/\delta$ strips, which are already occupied until height h_0. In the same way we handle configurations of height less than h_0. Since there are at most $2M$ configurations with nonzero height, we get at most $2M/\delta = 10/\delta^2(\lfloor \log(2/\delta) \rfloor + 1) < N$ assignments of height δh_0, which can be distributed to N strips. Thus by this assignment policy, where the configurations are balanced, in each strip we allocate an area of height at most $(1+\delta)h_0$ for at most two different configurations. This step takes time $m'2N = \mathcal{O}(MN)$ as for any configuration we consider each strip at most twice. □

The next Lemma shows how to construct a feasible integral packing from a fractional packing.

Lemma 6.4. *Let $x = (x_1, \ldots, x_q)$ be a solution of (5.1) for input $\mathcal{J} \cup \mathcal{J}'$ with at most $m \leq M$ nonzero entries $x_1, \ldots, x_m$. For $N > 10/\delta^2(\lfloor \log(2/\delta) \rfloor + 1)$ we can convert x to a feasible integral packing of the rectangles in $\mathcal{J} \cup \mathcal{J}'$ with height at most $(1+\delta)h_0 + 2h_{\max}$ and at most two different configurations per strip.*

Proof. With Lemma 6.3 we obtain a fractional packing from x with height at most $(1+\delta)h_0$ and at most two different configurations per strip and $2M$ different configurations in total. Consider a strip S_ℓ with configurations C_i and C_k. Between height 0 and h_0 we reservate area for C_i of height $x_{i_\ell} = h_0$ and between height h_0 and $(1+\delta)h_0$ we reservate area for C_k of height $x_{k_\ell} = \delta h_0$. We assign to each reservated area additional space of height $h_{\max}$. Therefore in each strip S_ℓ the total height of the fractional packing is increased by $2h_{\max}$ to at most $(1+\delta)h_0 + 2h_{\max}$. Now we can convert this packing to a feasible integral packing of the rectangles in $\mathcal{J} \cup \mathcal{J}'$ in the following way:
Consider the columns of width w_j starting in the first strip where w_j appears. In each column but the last one (for w_j), there are a sequence of

rectangles and possibly two fractional rectangles. One is the top part of a rectangle, which does not fit completely into the previous column and the other one is the bottom part of a rectangle that is too tall to fit into this column. In each column we pack all rectangles belonging to the sequence and the completed fractional rectangle from the top. If the considered column is the last one for width w_j we have only one fractional rectangle at the bottom and possibly a rectangle that is a rounded fraction of a rectangle in $\mathcal{R}_{wide}$ that was split by rounding Algorithm 6 between width w_j and the next smaller width w_{j+1}. Here we add the fraction of width w_{j+1} belonging to the same rectangle in $\mathcal{R}_{wide}$. So we can guarantee that each configuration C_i with height x_{i_ℓ} in strip S_ℓ is filled up to height at least $x_{i_\ell} - h_{\max}$ with rectangles of $\mathcal{J} \cup \mathcal{J}'$. □

Since there are at most two different configurations per strip, the additive constant will be improved to $\mathcal{O}(1)h_{\max}$, while the running time remains the same as for Algorithm 8. For a smaller value of N the balancing argument for the configurations can not be applied.

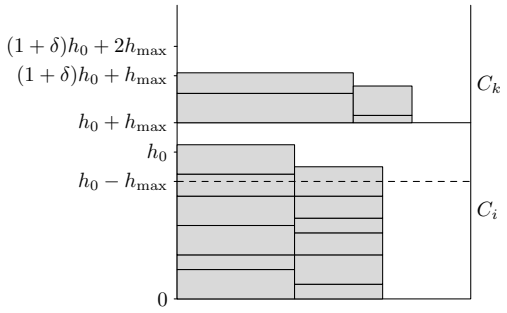

Figure 6.1: S_ℓ with C_i and C_k.

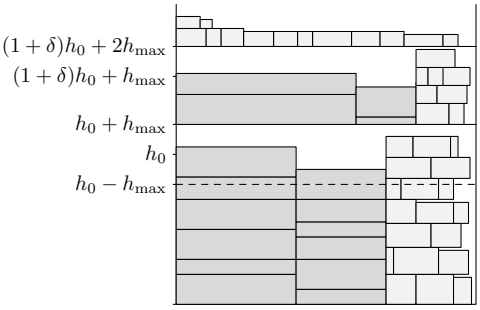

Figure 6.2: S_ℓ after packing the narrow rectangles.

Our last step is to add the narrow rectangles $\mathcal{R}_{narrow}$. We use a modified version of the NFDH algorithm: For strip S_ℓ, as above, we pack narrow rectangles using NFDH into the empty space next to the configurations until the total height is at most $(1+\delta)h_0 + 2h_{\max}$. After that we repeat the process for strip $S_{\ell+1}$. If all strips are filled in this way, we draw a horizontal line at height $(1+\delta)h_0 + 2h_{\max}$ in each strip and pack the remaining narrow rectangles with Algorithm 2 on top (see Figure 6.1 and 6.2). Let h_{final} denote the height of the packing after adding the narrow rectangles. If $h_{final} > (1+\delta)h_0 + 2h_{\max}$, by Lemma 3.4 we can ensure that the maximum difference of the heights of two arbitrary strips is at most $h_{\max}$.

Lemma 6.5. *Let $N > 10/\delta^2(\lfloor \log(2/\delta) \rfloor + 1)$. If $h_{final} \geq (1+\delta)h_0 + 2h_{\max}$, then we have*

$$h_{final} \leq \frac{SIZE(\mathcal{J} \cup \mathcal{J}' \cup \mathcal{R}_{narrow})}{N(1-\delta/2)} + \delta h_0 + 6h_{\max}.$$

Proof. We consider the strip $S_{\min}$ with

$$SIZE(S_{\min}) = \min_{\ell \in [N]} SIZE(S_\ell).$$

Let C_i and C_k be the two configurations for $S_{\min}$ with heights $x_{i_\ell} = h_0$ and $x_{k_\ell} \leq \delta h_0$ and widths c_i, c_k, respectively. Let $a_1 < \ldots < a_r$ be the ordered sequence of shelves constructed by modified $NFDH$ in $S_{\min}$, such that $a_k \geq (1+\delta)h_0 + 2h_{\max}$ or $0 \leq a_k \leq h_0 - h_{\max}$ for all $k \in [r]$. Let $a_{s_1} < \ldots < a_{s_{r'}}$ be the subsequence of shelves with at least one rectangle. Furthermore, let b_{s_j} and b'_{s_j} be the heights of the first and the last rectangle placed in shelf $a_{s_j}, j \in \{1, \ldots, r'\}$. A shelf is closed, when the next narrow rectangle does not fit completely on the shelf. Notice that all narrow rectangles on shelf a_{s_j} have height $\geq b'_{s_j}$. Let $a_{\bar{s}_1} < \ldots < a_{\bar{s}_{\bar{r}}}$ be the subsequence of $a_{s_1} < \ldots < a_{s_{r'}}$ such that either $a_{\bar{s}_j} \geq (1+\delta)h_0 + 2h_{\max}$ or $0 \leq a_{\bar{s}_j} + b'_{\bar{s}_j} \leq h_0 - h_{\max}$. Keep in mind that the region $[0, c_i] \times [0, x_{i_\ell} - h_{\max}]$ of $S_{\min}$ is completely filled with rectangles from $\mathcal{J} \cup \mathcal{J}'$. We consider three cases for shelf a_k with $k < r$:

Case 1: On shelf a_k there is at least one narrow rectangle and $a_k \geq (1+\delta)h_0 + 2h_{\max}$ or $0 \leq a_k + b'_k \leq h_0 - h_{\max}$. In this case there exists $j \in \{1, \ldots, \bar{r}-1\}$ with $k = \bar{s}_j$. Therefore, an area of at least $b'_{\bar{s}_j}(1-\delta/2)$ is

covered by wide and narrow rectangles from $\mathcal{J} \cup \mathcal{J}' \cup \mathcal{R}_{narrow}$.
Case 2: On shelf a_k there is at least one narrow rectangle with $a_k + b'_k > h_0 - h_{\max}$ and $a_k \leq h_0 - h_{\max}$. In this case $(h_0 - h_{\max} - a_k)(1 - \delta/2)$ is covered by wide and narrow rectangles.
Case 3: On shelf a_k there is no narrow rectangle and $a_k \leq h_0 - h_{\max}$. This case may happen, when the wide rectangles in configuration C_i have already a total width larger than $1 - \delta/2$. In this case an area of at least $(h_0 - h_{\max} - a_k)(1 - \delta/2)$ is covered by wide rectangles in $\mathcal{J} \cup \mathcal{J}'$. Cases 2 and 3 may happen only once per strip.
Let

$$X := \sum_{k \notin \{\bar{s}_1, \ldots, \bar{s}_{\bar{r}-1}\}} (h_0 - h_{\max} - a_k)\,.$$

Then $(1 - \delta/2)(X + \sum_{j=1}^{\bar{r}-1} b'_{\bar{s}_j})$ is a lower bound for $SIZE(S_{\min})$ which itself is bounded from above by $1/N SIZE(\mathcal{J} \cup \mathcal{J}' \cup \mathcal{R}_{narrow})$. On the other hand, the height of the final packing is at most $X + \sum_{j=1}^{\bar{r}} b_{\bar{s}_j} + 4h_{\max} + \delta h_0$ plus $h_{\max}$ (by Lemma 3.4 after packing the narrow rectangles on top). This gives

$$\begin{aligned}
h_{final} &\leq X + \sum_{j=1}^{\bar{r}} b_{\bar{s}_j} + \delta h_0 + 5h_{\max} \\
&\leq X + \sum_{j=1}^{\bar{r}-1} b_{\bar{s}_j} + \delta h_0 + 6h_{\max} \\
&\leq \frac{SIZE(S_{\min})}{1 - \delta/2} + \delta h_0 + 6h_{\max} \\
&\leq \frac{SIZE(\mathcal{J} \cup \mathcal{J}' \cup L_{narrow})}{N(1 - \delta/2)} + \delta h_0 + 6h_{\max}.
\end{aligned}$$

□

Theorem 6.6. *If $N > 10/\delta^2(\lfloor \log(2/\delta) \rfloor + 1)$ the Algorithm 9 generates for a set $\mathcal{R}$ of n rectangles a packing of height at most $(1 + \varepsilon)\mathrm{OPT}_{MSP}(\mathcal{R}) + \mathcal{O}(1)h_{\max}$ in time $\mathcal{O}(\varepsilon^{-6} \log(1/\varepsilon) + \varepsilon^{-2} \log^2(1/\varepsilon) n \log(n))$.*

Proof. If $h_{final} > (1 + \delta)h_0 + 2h_{\max}$ we conclude with Lemma 6.5 and since $\delta = \frac{2\varepsilon}{7+\varepsilon}$ we have

$$
\begin{aligned}
h_{final} &< \frac{SIZE(\mathcal{J} \cup \mathcal{J}') + SIZE(\mathcal{R}_{narrow})}{N(1-\delta/2)} + \frac{\delta FSP(\mathcal{J} \cup \mathcal{J}')}{N} + 6h_{\max} \\
&\overset{\text{Lem.5.4}}{\leq} \frac{(1+\delta)SIZE(\mathcal{R}_{wide}) + SIZE(\mathcal{R}_{narrow})}{N(1-\delta/2)} \\
&\qquad + \frac{\delta(1+\delta)FSP(\mathcal{R}_{wide})}{N} + 6h_{\max} \\
&\leq \frac{1+\delta}{1-\delta/2}\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R}) + \delta(1+\delta)\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R}) + 6h_{\max} \\
&\leq \frac{1+3\delta}{1-\delta/2}\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R}) + 6h_{\max} \\
&= (1+\varepsilon)\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R}) + 6h_{\max},
\end{aligned}
$$

where the third inequality holds because $SIZE(\mathcal{R})/N$ and $FSP(\mathcal{R}_{wide})/N$ are lower bounds for $\mathrm{OPT}_{\mathrm{MSP}}(\mathcal{R})$. On the other hand, it follows immediately from Lemmas 6.4 and 5.4 that

$$
\begin{aligned}
h_{final} &\leq (1+\delta)\frac{FSP(\mathcal{J} \cup \mathcal{J}')}{N} + 2h_{\max} \\
&\leq \frac{(1+\delta)^2 FSP(\mathcal{R}_{wide})}{N} + 2h_{\max} \\
&\leq (1+\varepsilon)OPT_{MSP}(\mathcal{R}) + 2h_{\max}.
\end{aligned}
$$

As in Section 5.3, the linear program (5.1) in step 4 can be solved in time $\mathcal{O}(\varepsilon^{-6}\log(1/\varepsilon))$. Step 5 takes time $\mathcal{O}(MN + M^2 n)$ which is $\mathcal{O}(M^2 n)$, assuming $n \geq N$. Together with the rounding and sorting steps the running time sums up to $\mathcal{O}(\varepsilon^{-6}\log(1/\varepsilon) + \varepsilon^{-2}\log^2(1/\varepsilon)n\log(n))$. □

Algorithm 9

Input: $\mathcal{R}$, $\varepsilon > 0$

Output: A packing of $\mathcal{R}$ into N strips

1: Set $\delta := {}^{2\varepsilon}/_{(7+\varepsilon)}$
2: Partition $\mathcal{R}$ into $\mathcal{R}_{wide} := \{j \in \mathcal{R} | w_j > \delta/2\}$ and $\mathcal{R}_{narrow} := \{j \in \mathcal{R} | w_j \leq \delta/2\}$.
3: Round the widths of the rectangles in $\mathcal{R}_{wide}$ with Algorithm 6 and obtain $\mathcal{J} \cup \mathcal{J}'$ with only $M = \mathcal{O}(1/\varepsilon\log(1/\varepsilon))$ different widths.
4: Solve the linear program (5.1) for input $\mathcal{J} \cup \mathcal{J}'$.
5: Construct a feasible solution for $\mathcal{J} \cup \mathcal{J}'$ into N strips with height at most $(1+\delta)h_0 + 2h_{\max}$ by balancing the configurations.
6: Use modified NFDH to pack the rectangles in $\mathcal{R}_{narrow}$ into the remaining space and on top of the strips.

II Scheduling Parallel Jobs in Platforms

7. Introduction

7.1 Definitions

This part considers the problem of Scheduling parallel Jobs in Platforms (SPP). We are given a set $\mathcal{J} = \{1, \ldots, n\}$ of n jobs, where a job $j \in \mathcal{J}$ is described by a pair (p_j, q_j) of a processing time $p_j \in \mathbb{Q}_{>0}$ and the number of processors $q_j \in \mathbb{N}$ that are required to execute j. We are also given a set $\mathcal{B}$ of N platforms $P_1, \ldots, P_N$, where each P_ℓ contains a set M_ℓ of $|M_\ell| = m_\ell$ processors for $\ell \in [N]$.

In general we assume that the m_ℓ may be different, that are *heterogeneous platforms.* If all the values m_ℓ are equal we have *identical platforms*.

A schedule is an assignment $a : \mathcal{J} \to \bigcup_{\ell=1}^{N} 2^{M_\ell} \times \mathbb{Q}_{\geq 0}$, that assigns every job j to a starting time t_j and to a subset $A_j \subset M_\ell$ of the processors of a platform P_ℓ with $|A_j| = q_j$. Obviously, a job j can only be scheduled in platform P_ℓ if $m_\ell \geq q_j$. A schedule is feasible if every processor in every platform executes at most one job at any time. The objective is to find a feasible schedule with minimum makespan $\max_{\ell \in [N]} C_{\max}^{(\ell)}$, where $C_{\max}^{(\ell)} = \max_{\{j | A_j \subset M_\ell\}} t_j + p_j$ denotes the local makespan for platform P_ℓ. We denote with $\mathrm{OPT}_{\mathrm{SPP}}(\mathcal{J}, \mathcal{B})$ the optimum value for the makespan of a schedule for the jobs in $\mathcal{J}$ into the platforms in $\mathcal{B}$. We identify $\mathcal{J}$ with its set of indices $\{1, \ldots, n\}$.

For $j \in \mathcal{J}$ we define the *size* of a jobs as $q_j p_j$ and consequently $SIZE(\mathcal{J}) := \sum_{j \in \mathcal{J}} q_j h_j$ for a set of jobs. With $p_{\max} := \max_{j \in \mathcal{J}} p_j$ we denote the largest processing time of a job.

For $N = 1$ the problem is equal to Scheduling parallel Jobs, in the relevant literature denoted with $P|size_j|C_{\max}$. This problem is strongly $\mathcal{NP}$-hard even for a constant number of processors $m \geq 5$ [17]. Similar as for Strip Packing by reduction from Partition it can be shown that there is no approximation algorithm for $P|size_j|C_{\max}$ with ratio < 1.5, unless

$\mathcal{P} = \mathcal{NP}$.

If we constrain the co-domain of the assignment a further and assume that all platforms contain the same number of processors, the problem is equivalent to Multiple Strip Packing: In addition to $A_j \in \bigcup_{\ell=1}^{N} 2^{M_\ell}$ we postulate that A_j is equal to a set of consecutively numbered processors for every job $j \in \mathcal{J}$. Every job then corresponds to a rectangle of width q_j and height p_j. Keep in mind here, that in general because of this contiguity constraint, algorithms for SPP cannot be directly applied to Multiple Strip Packing, since rectangles may be cut. But the optimal value for Multiple Strip Packing is an upper bound for the optimal value for the corresponding SPP problem. Interestingly, fractional versions of both problems coincide and therefore a solution of fractional (Multiple) Strip Packing gives a fractional solution for SPP.

Using the same reduction from 3-Partition as for MSP in Theorem 2.1 it can be derived that SPP is strongly $\mathcal{NP}$-hard even for identical platforms and that there exists no approximation algorithm with absolute ratio strictly better than 2, unless $\mathcal{P} = \mathcal{NP}$. Here, as for the fractional problems the contiguity constraint plays no role for proving the hardness result.

7.2 Related Work

There are several approximation algorithms for Scheduling parallel Jobs. To name a few, the best known is List Schedule by Garey and Graham [21]. It was shown by Feldmann et al. that List Schedule has absolute approximation ratio $(2 - 1/m)$ [18] using a dynamic and slightly different model. If the number of processors is bounded by a constant the problem admits a PTAS [2, 37]. In case that the number of machines is polynomially bounded in the number of jobs a $(1.5 + \varepsilon)$-approximation for the contiguous case (where a job has to be executed on processors with consecutive adresses) and a $(1 + \varepsilon)$-approximation for the non-contiguous problem were given in [40]. Recently, for an arbitrary number of processors Jansen gave a tight approximation algorithm with absolute ratio $1.5 + \varepsilon$ in [34]. Also for an arbitrary number of processors the contiguous case of $P|size_j|C_{\max}$ is closely related to Strip Packing as described above. For an overview about the related work for Strip Packing see Chapter 2.

A similar problem is Scheduling malleable Jobs. Here the process-

ing time of a job j depends on the number of allotted machines and can be described by a function $p_j : [m_N] \longrightarrow \mathcal{Q}^+ \cup \infty$, where $p_j(k)$ is the length of job j running on k parallel processors of a platform. The PTAS in [37] does also apply for malleable jobs if the number of processors is constant. Interestingly, in [40] it can be derived from the paper that the algorithms can also be applied to malleable jobs without using the assumption $m \leq poly(n)$. In [54] Mounié et al. presented a $(1.5+\varepsilon)$-approximation for scheduling malleable jobs with processing times given by monotone functions where the jobs are assigned to processors of consecutive addresses. The running time of this algorithm depends on the number of processors. An AFPTAS for scheduling malleable jobs on an arbitrary number of processors is given in [31].

For Scheduling parallel Jobs in Platforms (SPP) Tchernykh et al. presented in [63] an algorithm with absolute ratio 10. Earlier Remy claimed in [56] that the approximation ratio 2 of List Schedule is preserved when applied to SPP with identical platforms while in [63] and again later in [60] it is shown that List Schedule cannot even guarantee a constant approximation ratio for this problem. Schwiegelshohn et al. [60] achieved absolute approximation ratio 3 for SPP, and ratio 5 for the SPP with release times.

For SPP with identical platforms, we proposed a low cost approximation algorithm with absolute ratio $5/2$ in [6]. Recently, we presented a low-cost tight 2-approximation for this problem in case that no job does require more than half of the processors [10]. We were also able to extend our result in [6] to a fast $5/2$-approximation to SPP for heterogeneous platforms under the additional constraint that every job can be scheduled in each platform [7].

7.3 New Results

In this part we present several results for Scheduling parallel Jobs in Platforms that have been partly published as an extended abstract in [9] and as a full article [8].

In Chapter 8 we present an AFPTAS for SPP with additive factor $\mathcal{O}(1/\varepsilon \log(1/\varepsilon))p_{\max}$, where $p_{\max}$ denotes the largest processing time of a job. The Algorithm also applies to platforms that run at different speeds.

We show further that with only few modifications the algorithm is also able to handle malleable jobs and release times. Moreover, we assign the jobs to sets of consecutive processors, so the algorithm can also be applied for packing rectangles into strips of different width.

In Chapter 9 we give a polynomial time algorithm with absolute ratio $(2+\varepsilon)$ for SPP nearly closing the gap between the inapproximability bound of 2 and the currently best algorithm with ratio 3 (for the general case without additional constraints).

8. Approximation Schemes for SPP and Variants

In this chapter we present an AFPTAS for Scheduling parallel Jobs in (heterogeneous) Platforms. Note that Algorithm 9 for Multiple Strip Packing in Chapter 6 can be applied to SPP with identical platforms and we conclude from Theorem 6.1.

Corollary 8.1. *There is an algorithm that for a set $\mathcal{J}$ of n parallel jobs, a set of N identical platforms $\mathcal{B}$ and any accuracy $\varepsilon > 0$ generates a schedule for $\mathcal{J}$ into the platforms in $\mathcal{B}$ of length at most $(1+\varepsilon)\mathrm{OPT}_{SPP}(\mathcal{J},\mathcal{B}) + \mathcal{O}(1/\varepsilon \log(1/\varepsilon))p_{\max}$. For N sufficiently large, namely $N \in \Omega(1/\varepsilon^2 \log(1/\varepsilon))$, the length of the schedule can be reduced to $(1+\varepsilon)\mathrm{OPT}_{SPP}(\mathcal{J},\mathcal{B}) + \mathcal{O}(1)p_{\max}$. The algorithm has running time in $\mathcal{O}(\varepsilon^{-6}\log(1/\varepsilon) + n\log n)$.*

The number N of platforms does not show up in the running time of the above algorithm because in case of identical platforms we may assume $n \geq N$. If $N > n$ we only need to consider the first N platforms to schedule all jobs.

This following algorithm is also able to handle platforms running at different speeds, malleable jobs and jobs with release times. We first describe the algorithm for a more general version of SPP where each platform P_ℓ is assigned a speed value $s_\ell \in \mathbb{Q}_{>0}$, so that all processors of P_ℓ are running at speed s_ℓ. Therefore, the *duration time* of a job j in P_ℓ is given as $t_j^\ell := \frac{p_j}{s_\ell}$ if $q_j \leq m_\ell$ else $t_j^\ell := \infty$. We assume furthermore by scaling $\min_\ell s_\ell = 1$ and define $t_{\max} := \max_{j,\ell}\{t_j^\ell | t_j^\ell < \infty\}$, which is less than $p_{\max} := \max_j p_j$ as $\min_\ell s_\ell = 1$. The objective still is to find a feasible schedule that minimizes the global makespan, i.e. the latest finishing time of a job. We denote the problem SPP with different speed values as SPPS. Note that if all speed values are equal to one, then SPPS is equivalent to

SPP.

Theorem 8.2. *There is an algorithm that for a set $\mathcal{J}$ of n parallel jobs, a set $\mathcal{B}$ of N heterogeneous platforms of different speeds and any accuracy $\varepsilon > 0$ generates a schedule for $\mathcal{J}$ into the platforms in $\mathcal{B}$ with makespan at most*

$$(1+\varepsilon)\mathrm{OPT}_{SPPS}(\mathcal{J},\mathcal{B}) + \mathcal{O}(1/\varepsilon \log(1/\varepsilon))p_{\max}$$

The running time of the algorithm is in

$$\mathcal{O}(\varepsilon^{-6}\log(1/\varepsilon)N^2n^2\log^2(n)).$$

The algorithm is based on a linear program relaxation. This allows migration and preemption of jobs. That is, a job is allowed to be split into fractions that are executed in different platforms (if they fit). Emanating from the solution of the LP we compute a unique assignment of almost all jobs to the platforms. This is done by grouping the jobs similar as in Algorithm 6 and then rounding the fractions of jobs using a result of Lenstra et al. [50] ; i.e. the number of remaining fractional jobs per platform will be bounded by $\mathcal{O}(1/\varepsilon \log(1/\varepsilon))$. Remarkably, the rounding technique needs except an (approximate) solution of the LP no extra information about the speed values. For each platform we reschedule the obtained integral jobs with Algorithm 7, our improved version of the AFPTAS for Strip Packing of Kenyon and Rémila. Finally, the fractional jobs are scheduled behind. An overview of the algorithm is given below in Algorithm 10.

8.1 Relaxed Schedule

Let $\mathcal{J}$ be an instance of SPPS and let T be the makespan of an optimum schedule for $\mathcal{J}$. To simplify the structure of the schedule instead of handling the specific processing times t_j^ℓ we consider each platform as a two-dimensional bin of width m_ℓ and height Ts_ℓ and schedule the jobs concerning their lengths p_j within this bin. Furthermore, we abandon the constraint that a job has to be scheduled non-preemptively and within only one platform. We represent the schedule of a job (p_j, q_j) as a (finite) sequence of pairs $(I_i, Q_i)_{i\in I(j)}$, $I(j) \subseteq \mathbb{N}$, where every $I_i \subseteq [0,T]$ is a time interval and every Q_i is a set of processors so that there is a uniquely defined platform $P_{\ell_i} \in \mathcal{B}$ with $Q_i \subseteq M_{\ell_i}$ and $|Q_i| = q_j$. The intervals I_i and

Algorithm 10

Input: $\mathcal{J}$, $\mathcal{B}$, $\varepsilon > 0$

Output: A schedule for $\mathcal{J}$ into the platforms in $\mathcal{B}$

1: Solve the linear program (8.1) and get a fractional schedule where preemption and migration are allowed.
2: **for** $\ell \in \{1, \ldots, N\}$ **do**
3: In platform P_ℓ identify the (fractionally) assigned jobs with rectangles $\mathcal{R}^\ell$.
4: Group these rectangles as described in Algorithm 11 according their widths into $M + 1$ groups obtaining a set $\mathcal{R}'^{\ell}_{wide}$ partitioned into M groups and $\mathcal{R}^\ell_{narrow}$.
5: **end for**
6: Via a general assignment problem (8.7) round the assignment of the rectangles (and therefore the assignment of the jobs) obtaining sets of rounded rectangles $\widetilde{\mathcal{R}}^\ell_{wide}$, $\widetilde{\mathcal{R}}^\ell_{narrow}$ and fractional rectangles F^ℓ for $\ell \in [N]$.
7: **for all** $\ell \in [N]$ **do**
8: Pack $\widetilde{\mathcal{R}}^\ell_{wide} \cup \widetilde{\mathcal{R}}^\ell_{narrow}$ with Algorithm 7 into platform P_ℓ.
9: Schedule the fractional jobs in F^ℓ greedily on top of the schedule corresponding to the packing obtained before.
10: **end for**

sets Q_i may appear several times, but a pair (I_i, Q_i) may only appear more than once if I_i is a single point. Additionally, we assume that the following conditions hold:

(i) the time intervals for job j within the same platform do not overlap except maybe at the endpoints, i.e. for all $\ell \in [N]$
$\bigcup_{\substack{i,i' \in I(j), i \neq i' \\ \ell_i = \ell = \ell_{i'}}} \left(\overset{\circ}{I}_i \cap \overset{\circ}{I}_{i'} \right) = \emptyset$, where $\overset{\circ}{A}$ denotes the interior of a set A.

(ii) $\sum_{\ell=1}^{N} s_\ell \sum_{\{i \in I(j) | Q_i \subset P_\ell\}} |I_i| \geq p_j$ (covering constraint).

(iii) at any time for every processor there is at most one job running on it.

Keep in mind that under this constraints a job is allowed to be split among the platforms and may be executed in two different platforms at the same time, but never in parallel with itself within the same platform (except for a discrete time, when one piece starts and another ends). It can be executed on two different (not necessary disjoint) subsets of processors

within the same platform during different time intervals, where only the endpoints of the time intervals may overlap. An example how such a relaxed schedule can look like is given in Figure 8.1: Assume that $T = 10/s_{\ell_1}$ and job j needs to be scheduled on $q_j = 3$ processors for $p_j = 7.5$ operations. So in P_{ℓ_1} it is scheduled on processors $\{7,8,9\}$ during time $[0, 1/s_{\ell_1}]$ and on processors $\{2,3,4\}$ during time $[5/s_{\ell_1}, 7/s_{\ell_1}]$. In P_{ℓ_2} it is scheduled on processors $\{1,2,3\}$ during time $[0, 3/s_{\ell_2}]$ and in P_{ℓ_3} it is scheduled on processors $\{3,4,5\}$ during time $[3.5/s_{\ell_3}, 5/s_{\ell_3}]$. This gives $1 + 2 = 3$ operations in P_{ℓ_1}, 3 operations in P_{ℓ_2} and 1.5 operations in P_{ℓ_3} (this fulfills the covering constraint).

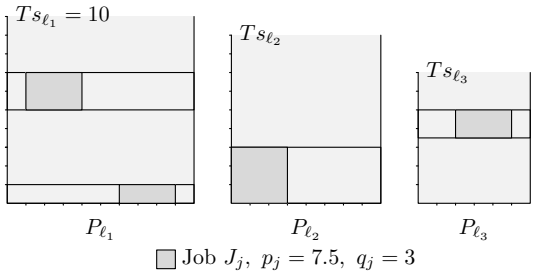

Figure 8.1: Relaxed schedule.

The relaxed schedule can be formulated via the linear program below: For each platform in P_ℓ, $1 \leq \ell \leq N$ we introduce configurations C^ℓ. A configuration C^ℓ is a function $C^\ell : [n] \longrightarrow \{0,1\}$, so that $\sum_{\{j \in [n] | C^\ell(j)=1\}} q_j \leq m_\ell$. It tells us which jobs can be scheduled in parallel in platform P_ℓ. By definition, the number $q(\ell)$ of different configurations for P_ℓ is bounded by 2^n. Let $\mathcal{C}^\ell = \{C^\ell_1, \ldots, C^\ell_{q(\ell)}\}$ denote the set of all configurations for a platform P_ℓ. In the linear program (8.1) below the variable $x_{C^\ell_k}$ indicates the length of configuration C^ℓ_k. That means that the jobs in $\{j \in \mathcal{J} | C^\ell_k(j) = 1\}$ are executed in platform P_ℓ during $x_{C^\ell_k}$ operation steps.

$$
\begin{aligned}
\sum_{k=1}^{q(\ell)} x_{C^\ell_k} &= s_\ell T \quad \ell \in [N] \\
\sum_{\ell=1}^{N} \sum_{\{k \in [q(\ell)] | C^\ell_k(j)=1\}} x_{C^\ell_k} &\geq p_j \quad j \in [n] \\
x_{C^\ell_k} &\geq 0 \quad k \in [q(\ell)],\ \ell \in [N]
\end{aligned}
\tag{8.1}
$$

The first N constraints ensure that the makespan $C^\ell_{\max}$ in each platform P_ℓ does not exceed T. The next n constraints are covering constraints for the n jobs. They make sure that every job is executed sufficiently long.

Lemma 8.3. *For $T = \mathrm{OPT}_{SPPS}(\mathcal{J}, \mathcal{B})$ the linear program above (8.1) is a relaxation of SPPS with input $(\mathcal{J}, \mathcal{B})$.*

Proof. Consider an optimum solution of SPPS with input $\mathcal{J}$ with $T = \mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}, \mathcal{B})$. Then we construct a solution for (8.1) in the following way: For each platform P_ℓ we consider the finishing and starting times $t_1, \ldots, t_{2\widetilde{n}(\ell)} \in [0, T]$ of the $\widetilde{n}(\ell) \leq n$ jobs assigned to this platform. They give a partition of $[0, T]$ (not minding empty intervals). We can assume w.l.o.g. $0 = t_1 \leq \ldots \leq t_{2\widetilde{n}(\ell)}$ and denote $t_{2\widetilde{n}(\ell)+1} = T$. For each non-empty interval $[t_i, t_{i+1}]$, $i \in [2\widetilde{n}(\ell)]$, we store a vector $v^\ell_i \in \{0,1\}^n$ and set $v^\ell_i(j) = 1$ if job j is scheduled in platform P_ℓ during this interval, else $v^\ell_i(j) = 0$. Furthermore we store a value $x^\ell_i := s_\ell(t_{i+1} - t_i)$ for v^ℓ_i. Note that if $t_{2\widetilde{n}(\ell)} < T$ the vector $v^\ell_{2\widetilde{n}(\ell)}$ is equal to $0_{\{0,1\}^n}$. Note that we store at most $2n$ vectors for every platform, since for $\widetilde{n}(\ell) = n$ we conclude $t_{2n} = T$. Let V^ℓ denote the set of stored vectors. As long as there are two identical vectors $v^\ell_i = v^\ell_j$ with $i \neq j$ (w.l.o.g. $i < j$) in V^ℓ we reset $x^\ell_i = x^\ell_i + x^\ell_j$ and discard v^ℓ_j. Finally we define for every $\ell \in \mathbb{N}$ and $1 \leq k \leq q(\ell)$

$$\bar{x}_{C^\ell_k} := \begin{cases} x^\ell_i \text{ if } C^\ell_k = v^\ell_i \text{ for } i \in 1, \ldots, 2\widetilde{n}(\ell) \\ 0 \textit{ else.} \end{cases}$$

We claim that $\bar{x} := (\bar{x}_{C^\ell_k})_{k \in [q(\ell)], \ell \in [N]}$ is a solution of (8.1). Clearly, we have that $\bar{x}_{C^\ell_k} \geq 0$ and $\sum_{k=1}^{q(\ell)} \bar{x}_{C^\ell_k} = s_\ell T$ for $\ell \in [N]$, since it is the sum of the values $s_\ell(t_{i+1} - t_i)$, $i \in [2\widetilde{n}(\ell)]$. Since we started with a feasible schedule, by definition of $\bar{x}$ we already have

$$\sum_{\{k \in [q(\ell)] \mid C^\ell_k(j) = 1\}} \bar{x}_{C^\ell_k} = \sum_{\{v^\ell_i \in V^\ell \mid v^\ell_i(j) = 1\}} x^\ell_i = p_j$$

for all jobs assigned to platform P_ℓ, so the covering constraint in (8.1) is fulfilled. □

8.2 Solving the LP

The linear program (8.1) above is a feasibility problem with $\mathcal{O}(2^n)$ variables and $N + n$ constraints. The encoding length of the constraint matrix and also the facet complexity of the corresponding polyhedron cannot be bounded by a ploynomial in n. Thus, solving (8.1) with the ellipsoid algorithm or via the separation problem could lead to exponential running times. Let $PS(\mathcal{J}, \mathcal{B})$ denote the makespan of an optimal relaxed schedule with the properties as described above. Clearly, $PS(\mathcal{J}, \mathcal{B}) \leq \mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}, \mathcal{B})$. It is sufficient to solve the LP approximately for a value T with $PS(\mathcal{J}, \mathcal{B}) \leq T \leq (1+\varepsilon)\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}, \mathcal{B})$. Let $d_{\min} := \frac{p_{\max}}{\max_\ell s_\ell}$. Since $\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}, \mathcal{B}) \geq d_{\min}$ we are only interested in fractional solutions for $T \geq d_{\min}$. We define a lower bound $L := d_{\min}$ and an upper bound $U := nt_{\max}$. Now there must be a value $L + j\varepsilon L \in [L, U]$ for $j \in \{0, \ldots \lfloor \frac{U-L}{\varepsilon L} \rfloor\}$, so that $PS(\mathcal{J}, \mathcal{B}) \leq L + j\varepsilon L \leq (1+\varepsilon)\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}, \mathcal{B})$. Consequently, using binary search to find a proper value T takes time in $\mathcal{O}(\log(\frac{U-L}{\varepsilon L})) = \mathcal{O}(\log(nt_{\max}(\varepsilon d_{\min})^{-1}))$. Since $t_{\max} \leq \frac{\max p_j}{\min s_\ell}$, we have $t_{\max}/d_{\min} \leq \max s_\ell$ (remember that $\min s_\ell = 1$) and therefore we find T in time

$$\mathcal{O}(\log(n\varepsilon^{-1} \max s_\ell)).$$

We may assume $n \geq \varepsilon^{-1}$, for $n < \varepsilon^{-1}$ the problem simplifies at certain points: The length of the schedule is at most $t_{\max}\varepsilon^{-1}$ and we find a proper value T in time $\mathcal{O}(\log(\varepsilon^{-2} \max s_\ell)) = \mathcal{O}(\log(\max\{\varepsilon^{-1}, \max s_\ell\}))$. The number of configurations and thus the number of variables in the LP is bounded by $2^{\varepsilon^{-1}}$. Furthermore, we need at most ε^{-1} platforms to schedule all jobs.

We can now formulate the problem described in (8.1) as a Max-Min Resource Sharing problem and solve it by using binary search on T and testing in each step the feasibility of a system of (in-)equalities for a given $T \in [L, U]$.

Let $(x_{C_k^\ell}) := (x_{C_k^\ell})_{k \in [q(\ell)], \ell \in [N]}$. For $x \in B(T)$ the system of inequalities is given as

$$\sum_{\ell=1}^{N} \frac{1}{p_j} \sum_{\{k \in [q(\ell)] \mid C_k^\ell(j)=1\}} x_{C_k^\ell} \geq 1 \quad j \in [n] \tag{8.2}$$

where

$$B(T) := \left\{ (x_{C_k^\ell}) | x_{C_k^\ell} \geq 0, \sum_{k=1}^{q(\ell)} x_{C_k^\ell} = s_\ell T,\ \ell \in [N] \right\}.$$

We represent the system of n inequalities as $Ax \geq e$, where e is the vector of all ones, and test the feasibility of the system by computing an approximate solution for

$$\lambda^\star = \max \{\lambda | Ax \geq \lambda e,\ x \in B(T)\}. \tag{8.3}$$

This problem can be considered as a fractional covering problem with convex set $B(T)$ and n covering constraints given by n concave functions. According to [24] we can compute a $(1-\delta)$-approximate solution for (8.3) in $\mathcal{O}(n(1/\delta^2 + \log n))$ iterations, where an iteration includes roughly summarized the following steps:
We start with an initial solution $\bar{x} \in B(T)$ and compute a certain price vector $y = y(\bar{x}) \in \mathbb{R}_+^n$ depending on $\bar{x}$. Then we consider the so called block problem, Max $\{y^T Ax | x \in B(T)\}$ and compute a $(1-\delta')$-approximate solution $\widetilde{x} \in B(T)$ for it with $\delta' = \delta/6$ (see the next section for details). Set $\bar{x} := (1-\tau)\bar{x} + \tau\widetilde{x}$ for a certain step width $\tau \in (0,1)$. The algorithm stops after $\mathcal{O}(n(1/\delta^2 + \log n))$ iterations for a given value $T \in [L, U]$ with a vector $\bar{x} \in B(T)$ so that

$$\sum_{\ell=1}^{N} \frac{1}{p_j} \sum_{\{k \in [q(\ell)] | C_k^\ell(j)=1\}} \bar{x}_{C_k^\ell} \geq (1-\delta)\lambda^\star \quad j \in [n].$$

For $x \in B(T)$ we define $\lambda(x) := \min_j \sum_{\ell=1}^{N} \frac{1}{p_j} \sum_{\{k \in [q(\ell)] | C_k^\ell(j)=1\}} x_{C_k^\ell}$. If $\lambda(\bar{x}) \geq (1-\delta)$ we have $\lambda^\star \geq 1$ and thus $A\bar{x} \geq (1-\delta)e$ and we may try a smaller value for T. If $\lambda(\bar{x}) < 1-\delta$ we have $\lambda^\star < 1$ and conclude that there is no solution $x \in B(T)$ satisfying (8.2) and have to increase T.

Using an idea from [16] the binary search can be avoided and thereby the running time can be improved: We compute a solution only for $T = 1$

and scale the solution $\bar{x}' := \lambda(\bar{x})^{-1}\bar{x}$. Then we have for $j \in [n]$

$$\begin{aligned}\sum_{\ell=1}^{N} \frac{1}{p_j} \sum_{\{k\in[q(\ell)]|C_k^\ell(j)=1\}} \bar{x}'_{C_k^\ell} &= \sum_{\ell=1}^{N} \frac{1}{p_j} \sum_{\{k\in[q(\ell)]|C_k^\ell(j)=1\}} \lambda(\bar{x})^{-1}\bar{x}_{C_k^\ell} \\ &\geq \frac{(1-\delta)\lambda^\star}{\lambda(\bar{x})} \\ &\overset{\lambda^\star\geq\lambda(\bar{x})}{\geq} 1-\delta\end{aligned}$$

and with $T := \lambda(\bar{x})^{-1}$

$$\sum_{k=1}^{q(\ell)} \bar{x}'_{C_k^\ell} = \sum_{k=1}^{q(\ell)} \lambda(\bar{x})^{-1}\bar{x}_{C_k^\ell} = \lambda(\bar{x})^{-1}s_\ell = Ts_\ell.$$

8.2.1 Solving the Block Problem.

Consider the block problem $\text{Max}\{y^T Ax | x \in B(T)\}$. The set $B(T)$ can be written as a Cartesian product of N convex sets

$$B_\ell(T) := \left\{ (x_{C_k^\ell})_{k\in[q(\ell)]} | \sum_{k=1}^{q(\ell)} x_{C_k^\ell} = s_\ell T,\ x_{C_k^\ell} \geq 0 \right\}.$$

Each set $B_\ell(T)$ is a simplex. So the block problem can be re-written as

$$\begin{aligned}\text{Max} \sum_{\ell=1}^{N}\sum_{j=1}^{n} \frac{y_j}{p_j} \sum_{\{k\in[q(\ell)]|C_k^\ell(j)=1\}} x_{C_k^\ell}, \\ x \in \bigtimes_{\ell=1}^{N} B_\ell(T).\end{aligned} \tag{8.4}$$

Thus, it is sufficient to solve N independent smaller block problems of the form

$$\begin{aligned}\text{Max} \sum_{j=1}^{n} \frac{y_j}{p_j} \sum_{\{k\in[q(\ell)]|C_k^\ell(j)=1\}} x_{C_k^\ell}, \\ (x_{C_k^\ell})_{k\in[q(\ell)]} \in B_\ell(T).\end{aligned} \tag{8.5}$$

For each $\ell \in [N]$ we find the optimum of (8.5) at a vertex $\widetilde{x}$ of B_ℓ(T). Such a vertex corresponds to a configuration $C_{\widetilde{k}}^\ell$ with $\widetilde{x}_{C_{\widetilde{k}}^\ell} = s_\ell T$ and $\widetilde{x}_{C_k^\ell} = 0$ for $C_k^\ell \neq C_{\widetilde{k}}^\ell$. Thus, we have to find a configuration $C_{\widetilde{k}}^\ell$ of jobs that can be

executed in parallel in P_ℓ with largest profit, where the profit value of a configuration C_k^ℓ is given as $\sum_{\{j\in[n]|C_k^\ell(j)=1\}} \frac{y_j}{p_j}$. This results in the following knapsack problem:

$$\begin{aligned} \text{Max } & \sum_{j=1}^{n} \frac{y_j}{p_j} x_j \\ \text{s.t } & \sum_{j=1}^{n} q_j x_j \leq m_\ell \\ & x_j \in \{0,1\} \end{aligned}$$

Kellerer and Pferschy showed in [43] that this knapsack problem can be solved in time

$$\mathcal{O}(n \min\{\log n, \log(1/\delta')\} + 1/\delta'^2 \log(1/\delta') \min\{n, 1/\delta' \log(1/\delta')\}),$$

which is $\mathcal{O}(n \log(1/\delta') + 1/\delta'^3 \log^2(1/\delta'))$ for $n \geq 1/\delta' \log(1/\delta')$. Therefore, we can solve the block problem in time $\mathcal{O}(N(n \log(1/\delta') + 1/\delta'^3 \log^2(1/\delta')))$. Summing up all near optimal solutions for (8.5) for $\ell = 1, \ldots, N$ gives a $(1-\delta')$-approximate solution for (8.4).

Here an open question is, wether it is possible to solve the Max-Min Resource Sharing problem using in each iteration only the solution of one of the N smaller block problems (8.5) and optimize cyclically over the platforms. In [64] such a method is described for the related Min-Max Resource Sharing problem.

8.2.2 Constructing a Schedule.

Now we construct a schedule based on an approximate solution of (8.3) as in [32]. Choose $\delta := \frac{3\varepsilon}{20}$ and assume $\varepsilon \leq 1$. By binary search on T or as described in the end of Section 8.2 we can achieve a solution $\widetilde{x}$ so that for every $\ell \in [N]$ we have $\sum_{k=1}^{q(\ell)} \widetilde{x}_{C_k^\ell} = Ts_\ell \leq (1+\varepsilon) s_\ell \text{OPT}_{\text{SPPS}}(\mathcal{J}, \mathcal{B})$ and $\sum_{\ell=1}^{N} \frac{1}{p_j} \sum_{\{k\in[q(\ell)]|C_k^\ell(j)=1\}} \widetilde{x}_{C^\ell} \geq (1-\delta)$ for all $j \in [n]$. We slightly extend the

length of each configuration setting $x_{C_k^\ell} := \widetilde{x}_{C_k^\ell}(1+4\delta)$ and conclude

$$\begin{aligned}\sum_{\ell=1}^{N} \frac{1}{p_j} \sum_{\{k\in[q(\ell)]\,|\,C_k^\ell(j)=1\}} x_{C_k^\ell} &\geq (1-\delta)(1+4\delta)\\ &= (1+3\delta-4\delta^2)\\ &\geq 1,\end{aligned}$$

since $\delta \leq \frac{3}{4}$. Consequently, in each platform the length of our generated schedule is also extended to

$$\begin{aligned}\sum_{k=1}^{q(\ell)} x_{C_k^\ell} &= (1+4\delta)\sum_{k=1}^{q(\ell)} \widetilde{x}_{C_k^\ell}\\ &\leq (1+4\delta)(1+\varepsilon)s_\ell \mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J},\mathcal{B})\\ &\leq (1+3\varepsilon)s_\ell \mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J},\mathcal{B}).\end{aligned} \tag{8.6}$$

8.3 Rounding the Fractional Solution.

In this section we round the jobs in order to get a unique assignment of every job to a subset of processors of a platform. Consider an approximate solution $(x_{C_k^\ell})$ of the LP-relaxation. We introduce a new variable $x_j^\ell \in [0, p_j]$ that indicates the length of the fraction of job j that is scheduled on P_ℓ. Formally this is $x_j^\ell = \sum_{\{k\in[q(\ell)]\,|\,C_k^\ell(j)=1\}} x_{C_k^\ell}$, the sum of the length of all configurations in P_ℓ in which j appears. We can assume for all jobs j the equality $\sum_{\ell=1}^{N} x_j^\ell = p_j$, if not we simply delete job j from appropriate configurations or replace a configuration by two "shorter" configurations (one with job j and one without, their total length is the same as the one of the original configuration). For all fractions x_j^ℓ of a platform P_ℓ we build rectangles (x_j^ℓ, q_j) of height x_j^ℓ and width q_j. Let $\mathcal{R}^\ell$ denote the set of rectangles for P_ℓ. We group the rectangles in $\mathcal{R}^\ell$ into $M+1$ groups for $M = \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$ as described in Algorithm 11 using parts of Algorithm 6. For every group G_i^ℓ, $i = 0,\ldots,M$, $\ell = 1,\ldots,N$, resulting from the rounding we denote with $y_{j,i}^\ell \in [0, p_j]$ the fraction of job j that is assigned to group G_i^ℓ. Furthermore let $z_{j,i}^\ell := y_{j,i}^\ell / p_j \in [0,1]$ (compare also to Algorithm 11).

If we were able to round the variables $z_{j,i}^\ell$ to integer values in $\{0,1\}$ (without losing too much), this would imply a unique assignment of every

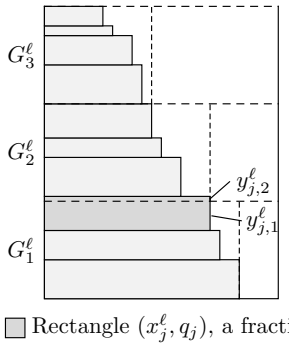

Figure 8.2: Constructing L'^{ℓ}_{wide}.

rectangle to exactly one group of a platform. Re-identifying the rectangles with jobs, where we identify the height of a rectangle with the length of a job and its width with the number of processors required, this would also imply a unique assignment of every job to a platform. We achieve such a rounding of the variables $z_{j,i}^\ell$ via the following general assignment problem, so that there remain at most $M+1$ fractionally assigned rectangles per platform.

$$
\begin{aligned}
\sum_{j=1}^{n} z_{j,0}^\ell p_j q_j &\leq SIZE(\mathcal{R}_{narrow}^\ell) \quad \ell \in [N] \\
\sum_{j=1}^{n} z_{j,i}^\ell p_j &\leq H(G_i^\ell) \quad i \in [M],\ \ell \in [N] \\
\sum_{\ell=1}^{N} \sum_{i=0}^{M} z_{j,i}^\ell &\geq 1 \quad j \in [n] \\
z_{i,j}^\ell &\in [0,1]
\end{aligned}
\tag{8.7}
$$

The above formulation is related to the problem of scheduling jobs on unrelated machines with $(M+1)N$ machines. Each group G_i^ℓ corresponds to a machine. Lenstra et al. showed in [50] that a feasible solution $(z_{j,i}^\ell)$ of this problem can be rounded to a feasible solution $(\widetilde{z}_{j,i}^\ell)$ of the corresponding integer program formulation in polynomial time, so that there remains at most one fractional job $\widetilde{z}_{j,i}^\ell < 1$ per machine. Hence, we get a unique assignment of almost all rectangles to the platforms P_ℓ except at most $M+1$ fractionally assigned rectangles per platform. Let F^ℓ denote the set of rectangles with fractional variables $\widetilde{z}_{j,i}^\ell$ after the rounding. We

Algorithm 11 Grouping

Input: $\mathcal{R}^\ell$, $\varepsilon > 0$
Output: A grouping of $\mathcal{R}^\ell$ into $M+1$ groups

1: Choose $\delta := \varepsilon/(4+\varepsilon)$
2: Partition the rectangles into wide and narrow rectangles, $\mathcal{R}^\ell_{wide} := \{(x^\ell_j, q_j) | q_j > (\delta/2)m_\ell\}$ and $\mathcal{R}^\ell_{narrow} := \{(x^\ell_j, q_j) | q_j \leq (\delta/2)m_\ell\}$.
3: Group the rectangles in $\mathcal{R}^\ell_{wide}$ with Step 1 to 5 of Algorithm 6 (not proceeding the rounding) into $M \leq 5/\delta(\lfloor \log(2/\delta) \rfloor + 1) = \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$ groups G^ℓ_i.
4: Denote the resulting list of rectangles with $\mathcal{R}'^\ell_{wide}$ and let $y^\ell_{j,i} \in [0, p_j]$ denote the fraction of job j that is assigned to G^ℓ_i. Let $z^\ell_{j,i} = y^\ell_{j,i}/p_j \in [0,1]$ denote the scaled fraction.
5: Compute $SIZE(\mathcal{R}^\ell_{narrow}) = \sum_{(x^\ell_j, q_j) \in \mathcal{R}^\ell_{narrow}} x^\ell_j q_j$ and locate the corresponding rectangles on top of the stack as group G^ℓ_0. Let $y^\ell_{j,0} \in [0, p_j]$ denote the fraction of a narrow job j that is assigned to G^ℓ_0 and and let $z^\ell_{j,0} = y^\ell_{j,0}/p_j \in [0,1]$.

will execute the corresponding jobs at the end of the schedule; their total processing time is bounded by $(M+1)t_{\max}$. From now on we consider for each platform P_ℓ an instance of Strip Packing containing a set of wide rectangles $\widetilde{\mathcal{R}}^\ell_{wide} := \{(\widetilde{z}^\ell_{j,i} p_j, q_j) | \widetilde{z}^\ell_{j,i} = 1, i > 0\}$ and a set of narrow rectangles $\widetilde{\mathcal{R}}^\ell_{narrow} := \{(\widetilde{z}^\ell_{j,0} p_j, q_j) | \widetilde{z}^\ell_{j,0} = 1\}$. In every platform P_ℓ we repack the pre-assigned rectangles in $\widetilde{\mathcal{R}}^\ell_{wide} \cup \widetilde{\mathcal{R}}^\ell_{narrow}$ using Algorithm 7.

8.4 Analysis

In the end we re-identify the rectangles with jobs, i.e. their widths with q_j and their heights with p_j. Note that a packing of the rectangles of total height $h(\ell)$ in platform P_ℓ corresponds to a schedule with makespan $h(\ell)/s_\ell$. Then the fractional jobs in F^ℓ are scheduled on top. To directly apply a Strip Packing subroutine we scale the widths of all rectangles in $\widetilde{\mathcal{R}}^\ell_{wide} \cup \widetilde{\mathcal{R}}^\ell_{narrow}$ by $1/m_\ell$. Using this assumption, we can consider platform P_ℓ as a strip of width 1 and infinite height. As we consider each platform and the allocated jobs independently, this has no impact on the solution.

8.4.1 Analyzing the Output

Let $(x_{C_k^\ell})$ be an approximate solution of (8.1) and let $\widetilde{\mathcal{R}}^\ell_{wide} \cup \widetilde{\mathcal{R}}^\ell_{narrow}$ contain the rectangles that have to be repacked in Step 8 of Algorithm 10 with Algorithm 7.

Let $\mathcal{R}^\ell_{round}$, $\mathcal{R}'^\ell_{round}$ and $\widetilde{\mathcal{R}}^\ell_{round}$ denote the sets of rectangles obtained from $\mathcal{R}^\ell_{wide}$, $\mathcal{R}'^\ell_{wide}$ and $\widetilde{\mathcal{R}}^\ell_{wide}$ via the rounding Algorithm 6. Remember that the difference between $\mathcal{R}^\ell_{wide}$ and $\mathcal{R}'^\ell_{wide}$ is that in $\mathcal{R}'^\ell_{wide}$ every rectangle is uniquely assigned to one of the M groups G^ℓ_i (since we introduced two new rectangles for border rectangles). As in Chapter 5, for a list of rectangles $\mathcal{R}$ we denote with $FSP(\mathcal{R})$ the height of an optimal solution of the linear program (5.1) corresponding to a FRACTIONAL STRIP PACKING with input $\mathcal{R}$ and height $FSP(\mathcal{R})$. Thus $FSP(\mathcal{R}^\ell_{round}) = FSP(\mathcal{R}'^\ell_{round})$. Moreover, we can show the following:

Lemma 8.4. *For all $\ell \in [N]$ we have*

a) $FSP(\widetilde{\mathcal{R}}^\ell_{round}) \leq (1+\delta)^2 FSP(\mathcal{R}^\ell_{wide})$

b) $SIZE(\widetilde{\mathcal{R}}^\ell_{round}) \leq (1+\delta)^2 SIZE(\mathcal{R}^\ell_{wide})$.

Proof. First notice that $FSP(\mathcal{R}^\ell_{wide}) = FSP(\mathcal{R}'^\ell_{wide})$ since we are comparing fractional solutions. By Lemma 5.4 we have $(1+\delta)FSP(\mathcal{R}^\ell_{wide}) \geq FSP(\mathcal{R}^\ell_{round}) = FSP(\mathcal{R}'^\ell_{round})$. During building the (indexed) sets $\widetilde{\mathcal{R}}^\ell_{wide}$ from $\mathcal{R}'^\ell_{wide}$ in Step 6 of Algorithm 10 via the general assignment problem we do not increase the total height of any group G^ℓ_i and we do not exceed the largest width of a rectangle that appears in it. Therefore we have $\mathcal{R}'^\ell_{round} \geq_g \widetilde{\mathcal{R}}^\ell_{wide}$ and thus $FSP(\mathcal{R}'^\ell_{round}) \geq FSP(\widetilde{\mathcal{R}}^\ell_{wide})$. Again applying Lemma 5.4 to $\widetilde{\mathcal{R}}^\ell_{wide}$ and $\widetilde{\mathcal{R}}^\ell_{round}$ gives a). In a similar way assertion b) follows. □

Let h^ℓ_{round} denote the height of the packing produced by converting the fractional solution of the linear program (5.1) for input $\widetilde{\mathcal{R}}^\ell_{round}$ into an integral one. We obtain the integral solution by adding additional space equal to the maximum height of a rectangle to each configuration appearing with height > 0 in the fractional solution. Each basic solution of (5.1) has at most M non-zero entries. Thus, there are effectively at most $2M$ different configurations after filling the configurations as in Lemma 6.3. Consequently we achieve a feasible integral packing of $\widetilde{\mathcal{R}}^\ell_{round}$ with height $h^\ell_{round} \leq FSP(\widetilde{\mathcal{R}}^\ell_{round}) + 2M \max\{p_j | (p_j, q_j) \in \widetilde{\mathcal{R}}^\ell_{round}\}$.

Now let $h(\ell)$ denote the height after adding the narrow rectangles in $\widetilde{\mathcal{R}}^{\ell}_{narrow}$ to our packing in platform P_ℓ, $\ell \in [N]$. We bound $h(\ell)$ in the following way:

Lemma 8.5. *For all $\ell \in [N]$ we have*

$$h(\ell) \leq (1+7\varepsilon)\mathrm{OPT}_{SPPS}(\mathcal{J},\mathcal{B})s_\ell + \mathcal{O}(1/\varepsilon \log(1/\varepsilon)) \max\{p_j|(p_j,q_j) \in \widetilde{\mathcal{R}}^{\ell}_{wide} \cup \widetilde{\mathcal{R}}^{\ell}_{narrow}\}.$$

Proof. First note that by construction we have that the height of an optimal fractional strip packing of the wide and narrow rectangles in $\mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow}$ into platform P_ℓ is less than the length of the schedule corresponding to the approximate solution of (8.1) given in (8.6), that is

$$FSP(\mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow}) \leq s_\ell(1+3\varepsilon)\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J},\mathcal{B}). \tag{8.8}$$

We consider two different cases:
If $h(\ell) > h^{\ell}_{round}$, Kenyon and Rémila [45] showed (for strip packing with M different rounded width) that

$$h(\ell) \leq \frac{SIZE(\widetilde{\mathcal{R}}^{\ell}_{round} \cup \widetilde{\mathcal{R}}^{\ell}_{narrow})}{(1-\delta/2)} + (4M+1)\max\{p_j|(p_j,q_j) \in \widetilde{\mathcal{R}}^{\ell}_{round} \cup \widetilde{\mathcal{R}}^{\ell}_{narrow}\}.$$

Since $SIZE(\widetilde{\mathcal{R}}^{\ell}_{narrow}) \leq SIZE(\mathcal{R}^{\ell}_{narrow})$ by construction Lemma 8.4 b) implies

$$SIZE(\widetilde{\mathcal{R}}^{\ell}_{round} \cup \widetilde{\mathcal{R}}^{\ell}_{narrow}) \leq (1+\delta)^2 SIZE(\mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow}). \tag{8.9}$$

With (8.8) and since we scaled the widths of the rectangles by $1/m_\ell$ we have $SIZE(\mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow}) \leq s_\ell(1+3\varepsilon)\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J},\mathcal{B})$. With $\delta = \frac{2\varepsilon}{7+\varepsilon}$ we

have

$$\begin{aligned}
h(\ell) &\overset{(8.9)}{\leq} SIZE(\mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow}) \frac{(1+\delta)^2}{(1-\delta/2)} \\
&\qquad + (4M+1) \max\{p_j | (p_j, q_j) \in \mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow}\} \\
&\leq \frac{(1+3\varepsilon)\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}, \mathcal{B}) s_\ell (1+\delta)^2}{(1-\delta/2)} \\
&\qquad + (4M+1) \max\{p_j | (p_j, q_j) \in \mathcal{R}^{\ell}_{round} \cup \mathcal{R}^{\ell}_{narrow}\} \\
&\leq (1+7\varepsilon)\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}, \mathcal{B}) s_\ell + (4M+1) \max\{p_j | (p_j, q_j) \in \mathcal{R}^{\ell}_{round} \cup \mathcal{R}^{\ell}_{narrow}\}
\end{aligned}$$

If $h(\ell) \leq h^{\ell}_{round}$ the converting process from fractional to integral gives

$$\begin{aligned}
h(\ell)_{round} &\leq FSP(\widetilde{\mathcal{R}}^{\ell}_{round}) + 2M \max\{p_j | (p_j, q_j) \in \widetilde{\mathcal{R}}^{\ell}_{wide}\} \\
&\overset{8.4\ a)}{\leq} FSP(\mathcal{R}^{\ell}_{wide})(1+\delta)^2 \\
&\qquad + 2M \max\{p_j | (p_j, q_j) \in \mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow}\} \\
&\leq FSP(\mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow})(1+\delta)^2 \\
&\qquad + 2M \max\{p_j | (p_j, q_j) \in \mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow}\} \\
&\overset{(8.8)}{\leq} s_\ell (1+3\varepsilon)\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}, \mathcal{B})(1+\delta)^2 \\
&\qquad + 2M \max\{p_j | (p_j, q_j) \in \mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow}\} \\
&\leq (1+6\varepsilon)\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}, \mathcal{B}) s_\ell + 2M \max\{p_j | (p_j, q_j) \in \mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow}\}.
\end{aligned}$$

According to Lemma 5.4 we have $M = \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$. □

The packing in each platform P_ℓ corresponds to a schedule with length (referring to p_j) at most

$$(1+7\varepsilon)\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}, \mathcal{B}) s_\ell + \mathcal{O}(1/\varepsilon \log(1/\varepsilon)) \max\{p_j | (p_j, q_j) \in \mathcal{R}^{\ell}_{wide} \cup \mathcal{R}^{\ell}_{narrow}\},$$

thus its completion time (referring to t^{ℓ}_j) is bounded by

$$(1+7\varepsilon)\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}, \mathcal{B}) + \mathcal{O}(1/\varepsilon \log(1/\varepsilon)) t_{\max}.$$

The remaining jobs in F^ℓ have total processing time bounded by

$$(M+1)t_{\max} = \mathcal{O}(1/\varepsilon \log(1/\varepsilon)) t_{\max} \leq \mathcal{O}(1/\varepsilon \log(1/\varepsilon)) p_{\max},$$

since $t_{\max} \leq p_{\max}$ as $\min s_\ell = 1$. Adding the remaining jobs in F^ℓ to the schedule does not change the magnitude of the additive factor. By rescaling ε we obtain that the makespan of the produced schedule in each platform P_ℓ is less than

$$C^\ell_{\max} \leq (1+\varepsilon)\mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J},\mathcal{B}) + \mathcal{O}(1/\varepsilon \log(1/\varepsilon))p_{\max}$$

and conclude Theorem 8.2. Since during the repacking process we considered jobs as rectangles, we assigned every job to a set of processors with consecutive addresses. Thus we also obtain an AFPTAS for a generalized Multiple Strip Packing where the strips may have different widths (in this case we have $s_\ell = 1$ for all $\ell \in [N]$).

8.4.2 Running Time of Algorithm 10

Since we can avoid the binary search on the makespan, the time needed for solving (8.1) approximately via Max-Min Resource Sharing in Step 1 is 'number of iterations' $\times N \times$ 'solving the knapsack' which is less than

$$\mathcal{O}(n(\varepsilon^{-2}+\log n)) \times N \times \mathcal{O}(n\log(1/\varepsilon) + 1/\varepsilon^3 \log^2(1/\varepsilon)) \\ \leq \mathcal{O}(Nn^2\varepsilon^{-5}\log n \log^2(1/\varepsilon)).$$

The number of non-zero configurations in the final solution in each platform P_ℓ is bounded by the number of iterations $\mathcal{O}(n(\varepsilon^{-2}+\log n))$ [24], since in each iteration there is at most one new configuration included.

As in [32] we can reduce the number of configurations per platform to $\mathcal{O}(n)$ by solving several systems of $n+1$ linear equalities with $n+1$ variables. For $\ell \in [N]$ let $\bar{x}^{(\ell)} = (\bar{x}_{C^\ell_k})$ be a solution of the following system:

$$\begin{aligned} \sum_{\{k\in[q(\ell)]\,|\,C^\ell_k(j)=1\}} \bar{x}_{C^\ell_k} &= b_j \geq p_j \qquad j = 1,\ldots,n \\ \sum_{k=1}^{q(\ell)} \bar{x}_{C^\ell_k} &= b_{n+\ell} \leq (1+\delta)s_\ell \mathrm{OPT}_{\mathrm{SPPS}}(\mathcal{J}). \end{aligned} \tag{8.10}$$

We may assume $x_{C^\ell_k} > 0$ for $k = 1,\ldots,q'(\ell)$ and $x_{C^\ell_k} = 0$ for $k = q'(\ell)+1,\ldots,q(\ell)$. We take a submatrix B with n rows and $n+1$ columns and solve the system $Bz = 0$. Since $rank(B) \leq n$ there is a non-trivial solution

$\bar{z}$ with $B\bar{z} = 0$. Then for any λ we have that $\bar{x} + \lambda\bar{z}$ is a solution of (8.10) where we extend $\bar{z}$ appropriately by adding zeros. For λ chosen appropriately we eliminate one positive variable $\bar{x}_{C_k^\ell}$ by setting $\bar{x} = \bar{x} + \lambda\bar{z}$. After $\mathcal{O}(n(\varepsilon^{-2} + \log n))$ times solving such a linear system we obtain a solution with only $\mathcal{O}(n)$ non-zero configurations per platform. The crucial point here is that it requires time in $\Omega(n^2)$ to solve each linear system. Thus, we compute the running time of Algorithm 10 without reducing the number of configurations.

Therefore, in Step 4 of the algorithm there are at most $n^2(\varepsilon^{-2} + \log n)$ rectangles in each platform that have to be sorted. So Step 4 takes time

$$\mathcal{O}(Nn^2(\varepsilon^{-2} + \log n)\log(n^2(\varepsilon^{-2} + \log n)) = \mathcal{O}(Nn^2\varepsilon^{-2}\log^2(n)\log(1/\varepsilon)).$$

We represent the assignment problem in Step 6 as a weighted bipartite graph $G = (V_1, V_2, E)$, where V_1 corresponds to the $N(M+1)$ machines (groups), V_2 to the jobs. There is an edge between the node representing group G_i^ℓ, $i = 0, \ldots, M$ for P_ℓ and the node representing job j if $z_{j,i}^\ell > 0$. This assignment problem can be converted in time $\mathcal{O}(|E||V_1|) = \mathcal{O}(|V_1|^2|V_2|) = \mathcal{O}(\varepsilon^{-2}\log^2(1/\varepsilon)N^2n)$ into another assignment, whose corresponding graph is a forest [55]. Applying the rounding technique in [50] to the new assignment takes time in $\mathcal{O}(|V_1| + |V_2|) = \mathcal{O}(\varepsilon^{-1}\log(1/\varepsilon)N + n)$. So Step 6 takes time

$$\mathcal{O}(\varepsilon^{-2}\log^2(1/\varepsilon)N^2n).$$

In Step 8 we solve the corresponding linear program (5.1) approximatively with accuracy $\Theta(\varepsilon)$ also via a Max-Min Resource Sharing problem. This can be done in time $\mathcal{O}(M(\varepsilon^{-2} + \log M)\max\{M + \varepsilon^{-3}, M\log\log(M\varepsilon^{-1})\})$ for every platform [32]. Since $M = \mathcal{O}(1/\varepsilon\log(1/\varepsilon))$ this gives for Step 8 a total running time in

$$\mathcal{O}(N\varepsilon^{-6}\log(1/\varepsilon)).$$

The overall running time sums up to

$$\mathcal{O}(\varepsilon^{-6}\log(1/\varepsilon)N^2n^2\log^2(n)).$$

8.5 Malleable Jobs

One can also obtain an AFPTAS for scheduling malleable jobs non-preemptively by only adding a few modifications to the algorithm. To get a better overview we do not consider the platform speeds here. But remember that one can easily add speeds here by considering bins of height $s_\ell T$ instead of T, where T denotes an optimum value for the makespan for Scheduling malleable Jobs in Platforms (SMP). In the following we give a short instruction how to adjust the algorithm:
In malleable scheduling a job j is described by a function $p_j : [m_N] \longrightarrow \mathcal{Q}^+ \cup \infty$, where $p_j(k)$ is the length of job j running on k parallel processors of a platform. We introduce a configuration as a map $f_\ell : [m_\ell] \longrightarrow \{0\} \cup [n]$ that assigns a processor to a job (0 for idle time). Instead of solving (8.1) we can solve the following linear program similar as in [31]:

$$\begin{aligned} \sum_{f_\ell \in \mathcal{F}^\ell} x_{f_\ell} &= T \quad \ell \in [N] \\ \sum_{\ell=1}^{N} \sum_{k=1}^{m_\ell} \frac{1}{p_j(k)} \sum_{f_\ell \in \mathcal{F}^\ell, |f^{-1}(j)=k|} x_{f_\ell} &\geq 1 \quad j \in [n] \\ x_{f_\ell} &\geq 0. \end{aligned} \tag{8.11}$$

Here the upper and lower bounds for binary search on T, U and L, respectively, are $U := nd_{\min}$ and $L := d_{\min}$ where $d_{\min} := \max_\ell d^\ell$ and
$d^\ell := \max_j \min_{1 \leq k \leq m_\ell} \{p_j(k) | p_j$ can be scheduled on $P_\ell\}$. As before we can avoid the binary search on the makespan and solve the corresponding Max-Min Recource Sharing problem for $T = 1$. The block problem also splits into N smaller block problems, where each of them corresponds to a multiple choice knapsack. It was shown in [44, 47] that this kind of knapsack problem can be solved approximately in time $\mathcal{O}(n \log n + \frac{mn}{\varepsilon})$ for n items and m equivalence classes. In our case m is equal to $m_{\max} = \max_\ell m_\ell$ the largest number of processors contained in a platform. From Section 8.4.2 we can conclude that the time needed for solving the linear program ist in

$$\mathcal{O}(Nn^2 \log^2 n \varepsilon^{-3} m_{max}).$$

To guarantee that we have chosen the right number of processors for

a job we replace the grouping Algorithm 11 by Algorithm 12 in Step 4 of Algorithm 10.

For including different speed values we define the processing time of job $j \in \mathcal{J}$ in platform P_ℓ as $t_j^\ell(k) = \frac{p_j(k)}{s_\ell}$. Note that $t_j^\ell(k) = \infty$ is possible. We define $p_{\max} := \max_{j,k}\{p_j(k) | p_j(k) < \infty\}$ and $t_{\max} := \max_{j,k,\ell}\{t_j^\ell(k) | t_j^\ell(k) < \infty\}$. To include speed values in the linear program we change the first N constraints of LP (8.11) into $\sum_{f_\ell \in \mathcal{F}^\ell} x_{f_\ell} = s_\ell T$, since different speeds can be considered as providing length $s_\ell T$ instead of T for the schedule. The block problem also splits into N multiple choice knapsack problems with bin sizes $s_\ell T$ and can be solved as before. The rounding step and the repacking process remain the same and finally we achieve the following theorem.

Theorem 8.6. *There is an algorithm that for a set $\mathcal{J}$ of n malleable jobs, a set $\mathcal{B}$ of N heterogeneous platforms of different speeds and any accuracy $\varepsilon > 0$ generates a schedule for $\mathcal{J}$ into the platforms in $\mathcal{B}$ with makespan at most*

$$(1+\varepsilon)\mathrm{OPT}_{SMP}(\mathcal{J},\mathcal{B}) + \mathcal{O}(1/\varepsilon \log(1/\varepsilon))p_{\max}$$

The running time of the algorithm is in

$$\mathcal{O}(\varepsilon^{-6}\log(1/\varepsilon)N^2n^2\log^2(n)m_{\max}).$$

8.6 Release Times

For a better overview we describe here the idea for the proof for the case that all platforms run at the same speed, i.e. $s_\ell = 1$ for all $\ell \in [N]$. The general case can be derived from it.

Theorem 8.7. *There is an algorithm that for a set $\mathcal{J}$ of n parallel jobs with release times, a set $\mathcal{B}$ of N heterogeneous platforms of different speeds and any accuracy $\varepsilon > 0$ generates a schedule for $\mathcal{J}$ into the platforms in $\mathcal{B}$ with makespan at most*

$$(1+\varepsilon)\mathrm{OPT}_{SPPR}(\mathcal{J},\mathcal{B}) + \mathcal{O}(1/\varepsilon^2 \log(1/\varepsilon))p_{\max}$$

where $\mathrm{OPT}_{SPPR}(\mathcal{J},,\mathcal{B})$ denotes the length of an optimum schedule for the problem. The running time of the algorithm is in

$$\mathcal{O}(\varepsilon^{-8}\log(1/\varepsilon)N^2n^2\log^2(n)\log(np_{\max})).$$

Proof. For a given set of n jobs $\mathcal{J}$ with release times $r_1, \ldots, r_n$ let T denote the optimum value of the makespan. Let $\Phi := \max_j r_j$, $\varepsilon > 0$ and assume that $1/\varepsilon \in \mathbb{N}$ or round it up to the next integer. Clearly we have $\Phi \leq T \leq \Phi + n \max_j p_j$ so we can find an approximate value for T with binary search in time $\mathcal{O}(\log(np_{\max}\varepsilon^{-1})) \leq \mathcal{O}(\log(np_{\max}))$ (for $n \geq \varepsilon^{-1}$.) If $\Phi \leq \varepsilon T$, we can apply Algorithm 10 to the instance ignoring the release times and shift the constructed schedule in the end by εT. So in this case we achieve for every accuracy ε an algorithm with output less than $(1+\varepsilon)\mathrm{OPT}_{\mathrm{SPPR}}(\mathcal{J}) + \mathcal{O}(1/\varepsilon \log(1/\varepsilon))p_{\max}$. Thus, we assume $\Phi > \varepsilon T$. As in [25] we round down the release time to the next multiples of $i\varepsilon T$ $i \in \{0, 1, \ldots, 1/\varepsilon\}$ and obtain new release times $\widetilde{r}_1 \ldots, \widetilde{r}_n$ with at most $R := 1/\varepsilon + 1 \in \mathcal{O}(1/\varepsilon)$ different values $\rho_1, \ldots, \rho_R$. To recover the error we made by rounding down we will shift the final schedule by εT in the end. For every platform P_ℓ we consider R new platforms $\widetilde{P}_{\ell,i}$, $i \in [R]$, with m_ℓ processors and create an instance $\widetilde{\mathcal{J}}_R$ of *SPP* (without release times) with RN platforms and n jobs. A job j can now be scheduled in platform $\widetilde{P}_{\ell,i}$ if it fits and if it is already released, i.e. $q_j \leq m_\ell$ and $\widetilde{r}_j \leq \rho_i$. For each of the new platforms $\widetilde{P}_{\ell,i}$ the value of an optimal fractional schedule is at most εT. Now we slightly modify the concept of a configuration: A configuration for a platform $\widetilde{P}_{\ell,i}$ is a function $C^{\ell,i} : [n] \longrightarrow \{0,1\}$, so that

- $\widetilde{r}_j \leq \rho_i$ for all $j \in [n]$ with $C^{\ell,i}(j) = 1$
- and $\sum_{\{j \in [n] | C^\ell(j)=1\}} q_j \leq m_\ell$.

We can apply Algorithm 10 (using the new concept of configurations in the LP-relaxation) to $\widetilde{\mathcal{J}}_R$ obtaining in each platform $\widetilde{P}_{\ell,i}$ a feasible schedule with length $(1+\varepsilon)\varepsilon T + \mathcal{O}(1/\varepsilon \log(1/\varepsilon))p_{\max}$. Summing up the schedules for $\widetilde{P}_{\ell,1}, \ldots, \widetilde{P}_{\ell,R}$ (in the right order of course) we create a schedule for the original platform P_ℓ with length $1/\varepsilon[(1+\varepsilon)\varepsilon T + \mathcal{O}(1/\varepsilon \log(1/\varepsilon))p_{\max}] = (1+\varepsilon)T + \mathcal{O}(1/\varepsilon^2 \log(1/\varepsilon))p_{\max}$. Correcting the mistake we made in the beginning and shifting the whole schedule by εT we obtain in every platform a feasible schedule with length

$$(1+2\varepsilon)T + \mathcal{O}(1/\varepsilon^2 \log(1/\varepsilon))p_{\max}.$$

The running time of the algorithm is in

$$\mathcal{O}(\varepsilon^{-8} \log(1/\varepsilon) N^2 n^2 \log^2(n) \log(np_{\max})).$$

as we apply Algorithm 10 to $N := N/\varepsilon$ platforms and $s_\ell = 1$ for all $\ell \in [N]$. □

It is an open question wether the additive term in this case can be decreased to $\mathcal{O}(1/\varepsilon \log(1/\varepsilon))$.

Algorithm 12 Grouping for malleable jobs

Output: $\mathcal{R}^\ell$, $\varepsilon > 0$
Input: A grouping of $\mathcal{R}^\ell$ into $M+1$ groups

1: Choose $\delta := {}^{2\varepsilon}/_{(7+\varepsilon)}$.
2: Partition the rectangles into wide and narrow rectangles, $\mathcal{R}^\ell_{wide} := \{(x^\ell_j, q_j) | q_j > (\delta/2)m_\ell\}$ and $\mathcal{R}^\ell_{narrow} := \{(x^\ell_j, q_j) | q_j \leq (\delta/2)m_\ell\}$.
3: Group the rectangles in $\mathcal{R}^\ell_{wide}$ with Step 1 to 5 of Algorithm 6 into $M = \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$ groups G^ℓ_i. Denote the resulting set of rectangles with $\mathcal{R}'^\ell_{wide}$. Let a^ℓ_i, b^ℓ_i be the smallest and the largest width of a rectangle in group G^ℓ_i and let $W^\ell_{i,j}$ be the set of widths job J_j adopts in G^ℓ_i.
4: For $i \in [M]$ and $w \in W^\ell_{i,j}$ let $y^\ell_{j,i}(w)$ denote the fraction of job j of width w that is assigned to G^ℓ_i. Let $z^\ell_{j,i} = \sum_{w \in W_{i,j}} y^\ell_{j,i}(w)$ be the complete fraction of job j in G^ℓ_i.
5: For each group $i \in [M]$ and job j with $|W^\ell_{j,i}| \geq 2$ compute $k^\ell_{j,i} := \arg\min_{k \in [a^\ell_i, b^\ell_i]} p^\ell_j(k)$ and replace the rectangles corresponding to job j in G^ℓ_i by $(z^\ell_{j,i} p_j(k^\ell_{j,i}), k^\ell_{j,i})$. Note that $p_j(k^\ell_{j,i})$ is the smallest processing time among all processor numbers $k \in [a^\ell_i, b^\ell_i]$.
6: For each job j with $|W^\ell_{j,0}| \geq 2$ compute $k^\ell_{j,0} := \arg\min_{k \in [0, \delta m_\ell]} p^\ell_j(k)k$ and replace all rectangles corresponding to job j in G^ℓ_0 by $(z^\ell_{j,0} p_j(k^\ell_{j,0}), k^\ell_{j,0})$.

9. $(2+\varepsilon)$-Approximation for SPP

In this chapter we present a $(2+\varepsilon)$-approximation for SPP improving the currently best known absolute ratio for an approximation algorithm for SPP. Note that on the one hand the 2-approximation for MSP given in Chapter 4 cannot be applied to SPP with heterogeneous platforms. On the other hand it does also not apply to SPP with identical platforms as the approximation ratio is not preserved: In case of a constant number of platforms the subroutine for rectangle packing with area maximization cannot be applied to select jobs.

Theorem 9.1. *There is an algorithm that for a set of n parallel jobs $\mathcal{J}$ and for any accuracy $\varepsilon > 0$ generates a schedule for $\mathcal{J}$ into a set $\mathcal{B}$ of N heterogeneous platforms with length at most*

$$(2+\varepsilon)\mathrm{OPT}_{SPP}(\mathcal{J},\mathcal{B}).$$

The algorithm has running time $g(1/\varepsilon)\cdot n^{\mathcal{O}(f(1/\varepsilon))}$ for some function g and $f(1/\varepsilon) = 2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon))}$.

Our algorithm considers two different scenarios for the shape of the platforms given by the input: Throughout this chapter we may assume $m_1 \geq m_2 \geq \ldots \geq m_N$ and use $i = 1,\ldots,N$ as running index for the platforms P_i. For $\varepsilon > 0$ with $3/18 \geq \varepsilon$ and $\gamma = \frac{8}{3}N_1$, where $N_1 = \mathcal{O}(1/\varepsilon^3 \log(1/\varepsilon))$ (specified exactly in Lemma 9.7), we distinguish two cases:

1. For all $i \in [N]$ we have $\frac{m_1}{m_i} \leq \gamma$.

2. There is a number $K \in [N]$ so that $\frac{m_1}{m_i} \leq \gamma$ holds for all $i \leq K$ and $\frac{m_1}{m_i} > \gamma$ holds for all $i > K$.

In Section 9.1 we describe our algorithm for the first case while the algorithm for the second case is given in Section 9.2. Here we distinguish wether $K \geq N_0$ or $K < N_0$.

For both cases the main idea obtaining a $(2+\varepsilon)$-approximation is the following. Via binary search in the interval $\left[\frac{SIZE(\mathcal{J})}{\sum_{i=1}^{N} m_i}, np_{\max}\right]$ we find a candidate T for the makespan. By scaling we may assume $T = 1$. We partition the platforms into a set $\mathcal{B}_0 = \{P_1, \ldots, P_{|\mathcal{B}_0|}\}$ containing a constant number $\mathcal{O}(N_1)$ of similar platforms, that is $m_1/m_{|\mathcal{B}_0|} \leq \gamma$, and a set $\mathcal{B}_1$ containing an arbitrary number of platforms. Then the platforms in $\mathcal{B}_1$ are grouped and rounded obtaining the set $\widetilde{\mathcal{B}}_1$ that contains groups of N_1 similar platforms.

Using gap creation [39] we modify an optimum solution so that the structure of jobs with large processing times in $\mathcal{B}_0$ could be approximately enumerated via a dynamic programming approach. Then we use a linear program to allocate the remaining jobs fractionally. Therefore the processing times of the remaining jobs are rounded harmonically, but only processing times of large jobs are increased. Then a linear program (based on a configuration LP) fractionally assigns some small jobs to layers of free space in $\mathcal{B}_0$ and all the rest to $\widetilde{\mathcal{B}}_1$. This gives a fractional schedule of length $T_\infty = 1.69..$ (harmonic constant) for any platform in $\widetilde{\mathcal{B}}_1$ and a schedule of length $(1+\mathcal{O}(\varepsilon))$ for any platform in $\mathcal{B}_0$. Using a rounding technique by [50] we convert the fractional assignment into an almost integral one. The jobs assigned to $\widetilde{\mathcal{B}}_1$ are rescheduled using a 2D-Bin Packing subroutine that is based on a Strip Packing algorithm and harmonic rounding [4]. This increases the schedule in $\widetilde{\mathcal{B}}_1$ to 2. As final step we rearrange some jobs to obtain a schedule for $\mathcal{B}_0 \cup \mathcal{B}_1$. An overview about the procedures in both cases is given by Algorithm 13.

9.1 Case 1: Similar Platforms

9.1.1 Platform Rounding

For $N_0 = 2(2N_1+1)$ we partition the set of platforms $\mathcal{B}$ into $L+1$ groups $B_0, B_1, \ldots, B_L$ by $L-$times collecting the N_1 smallest platforms where $L :=$

$\max\left\{0, \lfloor \frac{N-N_0}{N_1} \rfloor\right\}$. Let

$$\mathcal{B}_0 = B_0 := \{P_1, \ldots, P_{N-LN_1}\}$$

and for $\ell \in [L]$ define

$$B_\ell := \{P_{N-(L-(\ell-1))N_1+1}, \ldots, P_{N-(L-\ell)N_1}\}$$

and $\mathcal{B}_1 = \bigcup_{\ell=1}^{L} B_\ell$. Therefore, group $\mathcal{B}_1$ is further partitioned into several groups B_ℓ of equal constant cardinality. Each group $B_\ell \subseteq \mathcal{B}_1$ contains exactly N_1 platforms. Group $\mathcal{B}_0$ contains a constant number of platforms, moreover we have $5N_1 + 2 = N_0 + N_1 > |\mathcal{B}_0| \geq N_0$.

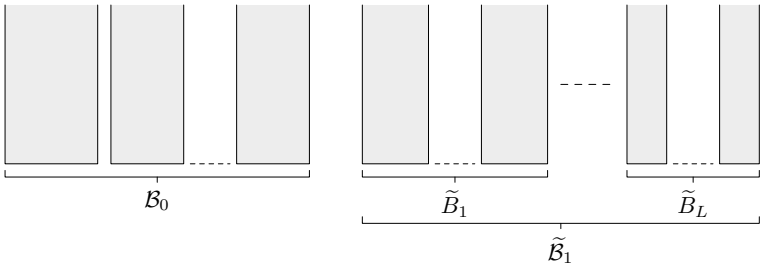

Figure 9.1: Grouping and rounding platforms in case 1.

In each group B_ℓ, $\ell \in [L]$ we round the number of processors of each platform up to the number of processors $\widetilde{m}_\ell := m_{N-(L-(\ell-1))N_1+1}$ of the largest platform $P_{N-(L-(\ell-1))N_1+1}$ contained in this group and denote with $\widetilde{B}_\ell$ the group of rounded platforms, see also Figure 9.1. We will compute a schedule for $\mathcal{B}_0 \cup \widetilde{\mathcal{B}}_1$, where $\widetilde{\mathcal{B}}_1 = \bigcup_\ell \widetilde{B}_\ell$, and convert this solution into a solution for $\mathcal{B}_0 \cup \mathcal{B}_1$.

It might be possible that the number of platforms is bounded by a constant $N < N_0 + N_1$. In this case we have only one group $\mathcal{B}_0$ and do not round the platforms. The algorithm simplifies at some points as it only performs the steps concerning $\mathcal{B}_0$.

9.1.2 Simplifying the Structure of an Optimum Solution in $\mathcal{B}_0$

Consider an optimum solution with makespan equal to 1 and denote with $\mathcal{J}^\star(\mathcal{B}_0)$ the set of jobs that are scheduled in $\mathcal{B}_0$ by the optimum solu-

Algorithm 13 $(2+\varepsilon)$-Algorithm

Input: $\mathcal{J}$, $\varepsilon > 0$

Output: A schedule of length $(2+\mathcal{O}(\varepsilon))\mathrm{OPT}_{\mathrm{SPP}}(\mathcal{J})$

1: For a certain constant $N_1 = \mathcal{O}(1/\varepsilon^3 \log(1/\varepsilon))$ partition the set of platforms into $L+1$ groups $\mathcal{B}_0, B_1 \ldots, B_L$ and let $\mathcal{B}_1 := \bigcup_{\ell=1}^{L} B_\ell$.
2: Round the number of processors of the platforms in each group B_ℓ and obtain $\widetilde{\mathcal{B}}_1$ containing groups $\widetilde{B}_\ell$ of N_1 similar platforms
3: **for** a candidate value for the makespan $T \in \left[\frac{SIZE(\mathcal{J})}{\sum_{i=1}^{N} m_i}, np_{\max}\right]$ **do**
4: **for** $k \in \{1, \ldots, \frac{|\mathcal{B}_0|}{\varepsilon}\}$ **do**
5: Let $\delta := \sigma_{k-1}$ where $\sigma_0 = \varepsilon/20$, $\sigma_{k+1} = \sigma_k^5$ for $k \geq 1$.
6: For δ distinguish small, medium, and large jobs
7: Round the processing times and possible starting times of large jobs to integral multiples δ^2.
8: For $\alpha = \delta^4/16$ distinguish wide and narrow large jobs.
9: Enumerate an assignment vector V of large wide jobs to $\mathcal{B}_0$ and let $\mathcal{J}_{la-wi}(\mathcal{B}_0)$ denote the selected jobs.
10: **for** an assignment vector V of large wide jobs **do**
11: Approximately guess the total load Π of large narrow jobs for each starting time and height in every platform of $\mathcal{B}_0$ and block gaps corresponding to Π.
12: **for** a guess Π **do**
13: Compute free layers of height δ^2 in $\mathcal{B}_0$.
14: Round the processing times p_j of the jobs $\mathcal{J}' = \mathcal{J} \setminus \mathcal{J}_{la-wi}(\mathcal{B}_0)$ harmonically.
15: Compute a solution of LP (9.1)
16: **if** There is no feasible solution for (9.1) **then**
17: Discard the guess Π and take another one and go back to Step 13. If all guesses have failed discard V, take another and go back to Step 11. If all pairs (V, Π) have failed, increase k and go to Step 5.
18: **end if**
19: Round the solution of (9.1) using a result of Lenstra et al. [50] and obtain an almost integral assignment of

- a subset of the small jobs to the free layers in $\mathcal{B}_0$
- a subset of the large narrow jobs to the gaps Π in $\mathcal{B}_0$
- the remaining jobs to the groups $\widetilde{B}_\ell$ in $\widetilde{\mathcal{B}}_1$.

20: Pack small jobs with Strip Packing subroutine into the layers.
21: **for** $\ell = 1, \ldots, L$ **do**
22: Pack the jobs assigned to $\widetilde{B}_\ell$ with 2D-Bin Packing subroutine into at most $2N_1$ bins of size $[0, \widetilde{m}_\ell] \times [0, 1]$.
23: **end for**
24: **end for**
25: **end for**
26: **end for**
27: **end for**
28: Convert the schedule for $\mathcal{B}_0 \cup \widetilde{\mathcal{B}}_1$ into a schedule for $\mathcal{B}_0 \cup \mathcal{B}_1$
29: Schedule medium jobs in $\mathcal{J}_\tau(\mathcal{B}_0)$ in P_1.

tion. Using the gap creation technique [39] we find a subset of jobs with medium processing time

$$\mathcal{J}^{\star}_{medium}(\mathcal{B}_0) \subseteq \mathcal{J}^{\star}(\mathcal{B}_0)$$

and small total load. We can remove these medium jobs from the instance and schedule them later on top of the solution constructed for the reduced instance only slightly increasing the makespan.

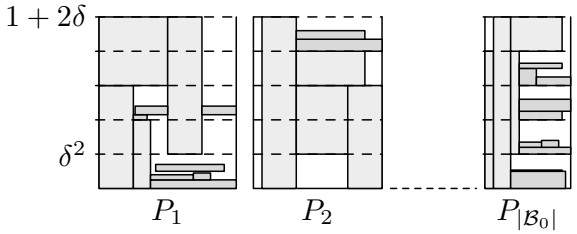

Figure 9.2: Simplified structure of an optimum solution in $\mathcal{B}_0$.

Define $\sigma_0 = \frac{\varepsilon}{9}$ and $\sigma_{k+1} = \sigma_k^5$ and sets $\mathcal{J}_k = \{j \in \mathcal{J} | p_j \in (\sigma_k, \sigma_{k-1}]\}$ for $k \geq 1$. Let $\mathcal{J}_k^{\star}(\mathcal{B}_0)$ and $\mathcal{J}_k^{\star}(\mathcal{B}_1)$ denote those jobs in $\mathcal{J}_k$ that are scheduled by an optimum algorithm in $\mathcal{B}_0$ and $\mathcal{B}_1$, respectively. Using $T = 1$ we have

$$\sum_{k\geq 1} \sum_{j \in \mathcal{J}_k(\mathcal{B}_0)} p_j q_j \leq \sum_{i=1}^{|\mathcal{B}_0|} m_i \leq |\mathcal{B}_0| m_1.$$

Using the pigeonhole principle we proof the existence of a set $\mathcal{J}_\tau^{\star}(\mathcal{B}_0)$ with $\tau \in \{1, \ldots, \frac{|\mathcal{B}_0|}{\varepsilon}\}$ so that $\sum_{j \in \mathcal{J}_\tau(\mathcal{B}_0)} p_j q_j \leq \varepsilon m_1$: If not, we have

$$\sum_{k\geq 1}^{|\mathcal{B}_0|/\varepsilon} \sum_{j \in \mathcal{J}_k(\mathcal{B}_0)} p_j q_j > |\mathcal{B}_0| m_1$$

which is a contradiction. Then we choose $\delta = \sigma_{\tau-1}$ and may assume that ε^{-1} is integral and thus

$$\delta^{-1} = \left(\frac{\varepsilon}{20}\right)^{-\left(5^{\tau-1}\right)} = \left(\frac{20}{\varepsilon}\right)^{5^{\tau-1}}$$

is integral (if not we choose the next smaller value for ε). Furthermore,

note that in the worst case

$$\delta^{-1} = \left(\frac{20}{\varepsilon}\right)^{\frac{|\mathcal{B}_0|}{\varepsilon}-1}.$$

We partition the jobs into

- small jobs $\mathcal{J}_{small} := \{j \in \mathcal{J} | p_j \leq \delta^5\}$,
- medium jobs $\mathcal{J}_{medium} := \mathcal{J}_\tau = \{j \in \mathcal{J} | p_j \in (\delta^5, \delta]\}$,
- large jobs with $\mathcal{J}_{large} := \{j \in \mathcal{J} | p_j \in (\delta, 1]\}$.

Scheduling the medium jobs in $\mathcal{J}^\star_\tau(\mathcal{B}_0)$ in the end on top of the largest platform P_1 using List Schedule [21] increases the makespan by at most

$$2\max\left\{(1/m_1)\sum_{j\in\mathcal{J}_\tau(\mathcal{B}_0)} p_j q_j, \max_{j\in\mathcal{J}_\tau} p_j\right\}$$
$$= 2\max\left\{\frac{\varepsilon m_1}{m_1}, \delta\right\} \leq 2\max\{\varepsilon, \delta\} \leq 2\varepsilon.$$

For $\mathcal{B}_0$ we can now simplify the structure of the starting times and different processing times of large jobs. We round up the processing time of each job with processing time $p_j > \delta$ to $\bar{p}_j = h\delta^2$, the next integral multiple of δ^2 with $(h-1)\delta^2 < p_j \leq h\delta^2 = \bar{p}_j$, for $h \in \{\frac{1}{\delta}+1, \ldots, \frac{1}{\delta^2}\}$. Since there can be at most $1/\delta$ jobs with height $> \delta$ on each processor within each platform this increases the makespan in $\mathcal{B}_0$ by only $\delta^2/\delta = \delta$. The number of different large jobs sizes H is bounded by

$$H = \frac{1}{\delta^2} - (\frac{1}{\delta}+1) + 1 \leq \frac{1}{\delta^2}.$$

In a similar way we round the starting time of each large job in $\mathcal{B}_0$ to $a\delta^2$, the next integral multiple of δ^2. This increases the makespan again by at most δ to $1+2\delta$. Therefore the large jobs have starting times $a\delta^2$ with $a \in \{0, 1, \ldots, \frac{1+2\delta}{\delta^2} - 1\}$ and the number of different starting times is $A = \frac{1+2\delta}{\delta^2}$.

An optimum schedule for $\mathcal{J}^\star(\mathcal{B}_0) \setminus \mathcal{J}^\star_\tau(\mathcal{B}_0)$ in $\mathcal{B}_0$ with rounded processing times $\bar{p}_j$ and rounded starting times for the large jobs has length at most $1+2\delta$. The schedule with simplified structure in $\mathcal{B}_0$ is illustrated in Figure 9.2.

Let $\tau \in \{1, \ldots, \frac{|\mathcal{B}_0|}{\varepsilon}\}$ be the current iteration step for finding $\mathcal{J}_\tau$ with the desired properties and $\delta = \sigma_{\tau-1}$. We enumerate the set of large wide jobs and approximately guess the structure of large narrow jobs in $\mathcal{B}_0$ that correspond to a good solution for the jobs with rounded processing times $\bar{p}_j$. We distinguish between wide and narrow large jobs and as follows. We may assume that $m_N \geq 32/\delta^4$. Otherwise the number of processors in P_1 and (since $m_1 \geq \ldots \geq m_N$) in every platforms is bounded by a constant

$$m_1 \leq m_N \gamma \leq \frac{32\gamma}{\delta^4}.$$

Thus, in every platform also the number of jobs that fit next to each other is bounded by $\frac{32\gamma}{\delta^4}$, moreover, the total number of large jobs in each platform is bounded by $\frac{32\gamma}{\delta^4} \cdot A$. Then we do not distinguish between large and narrow jobs and can enumerate the entire subset of large jobs that has to be scheduled in $\mathcal{B}_0$ as it is done for the large wide jobs in Section 9.1.2.1.

We choose $\alpha = \delta^4/16$. Then α satisfies $\alpha m_N \geq 2$, implying $\lfloor \alpha m_N \rfloor \geq \alpha m_N - 1 \geq \alpha m_N/2$. A job $j \in \mathcal{J}$ is called *wide* if $q_j \geq \lfloor \alpha m_N \rfloor$ and *narrow* otherwise. Furthermore distinguish

- large narrow jobs $\mathcal{J}_{la-na} := \{j \in \mathcal{J}_{large} | q_j \leq \lfloor \alpha m_N \rfloor\}$,
- large wide jobs $\mathcal{J}_{la-wi} := \{j \in \mathcal{J}_{large} | q_j > \lfloor \alpha m_N \rfloor\}$.

9.1.2.1 Assignment of Large Wide Jobs in $\mathcal{B}_0$

The number of large wide jobs that fit next to each other within one platform is bounded by $\frac{m_1}{\lfloor \alpha m_N \rfloor} \leq \frac{m_1}{\alpha m_N - 1} \leq \frac{m_1}{(\alpha m_N)/2} \leq (2\gamma)/\alpha$. Since large jobs have processing times $> \delta$, at most $\frac{1+2\delta}{\delta}$ rounded large jobs can be finished on one processor before time $1 + 2\delta$. Therefore, the number of large wide jobs that have to be placed in one platform in $\mathcal{B}_0$ is bounded by $\frac{2\gamma}{\alpha} \cdot \frac{1+2\delta}{\delta}$. Furthermore, in every platform a large jobs can have A different starting times. Each possible assignment of large wide jobs to platform and starting time can be represented by a tuple of vectors $V = (v_1, \ldots, v_{|\mathcal{B}_0|}) \in \left(([n] \cup \{0\})^{A \cdot \frac{2\gamma}{\alpha} \cdot \frac{1+2\delta}{\delta}}\right)^{|\mathcal{B}_0|}$. The running time of a dynamic program to compute such an assignment is equal to the number of possible vectors which is bounded by $(n+1)^{|\mathcal{B}_0| \cdot A \cdot \frac{2\gamma}{\alpha} \cdot \frac{1+2\delta}{\delta}}$. Let $\mathcal{J}_{la-wi}(\mathcal{B}_0)$ denote the set of large wide jobs selected and let $\mathcal{J}' := \mathcal{J} \setminus \mathcal{J}_{la-wi}(\mathcal{B}_0)$.

9.1.2.2 Gaps for Large Narrow Jobs in $\mathcal{B}_0$

In every platform $P_i \in \mathcal{B}_0$ we approximately guess the total load $\Pi^\star_{i,a,h}$ of jobs with height $h\delta^2$ starting at time $a\delta^2$. Note that we only need to consider those triples (i,a,h) with $h\delta^2 + a\delta^2 \leq (1+2\delta)$. Therefore we compute a vector $\Pi = (\Pi_{i,a,h})$ with $\Pi_{i,a,h} = b \cdot \lfloor \alpha m_N \rfloor$, $b \in \{0,1,\ldots,\frac{2\gamma}{\alpha}\}$ and $\Pi_{i,a,h} \leq \Pi^\star_{i,a,h} \leq \Pi_{i,a,h} + \lfloor \alpha m_N \rfloor$. Here the condition $\alpha m_N - 1 \geq \alpha m_N/2$ guarantees that $\frac{2\gamma}{\alpha} \cdot \lfloor \alpha m_N \rfloor \geq \frac{2\gamma}{\alpha} \cdot (\alpha m_N - 1) \geq m_1$. There is only a constant number $(1+\frac{2\gamma}{\alpha})^{|\mathcal{B}_0|\cdot A\cdot H}$ of different vectors Π. For every triple (i,a,h) we block a gap of $\Pi_{i,a,h} + \lfloor \alpha m_N \rfloor$ (not necessary contiguous) processors for large narrow jobs with $\bar{p}_j = h\delta^2$. Later we will place large narrow jobs with $\bar{p}_j = h\delta^2$ total width $\geq \Pi^\star_{i,a,h}$ into them. This will be done using linear programming and the subsequent rounding. Let G denote the total number of gaps, clearly $G \leq |\mathcal{B}_0| \cdot A \cdot H$.

Since $\gamma, |\mathcal{B}_0| = \mathcal{O}(N_1) = \mathcal{O}(1/\varepsilon^3 \log(1/\varepsilon))$ we have

$$\delta^{-1} \leq \mathcal{O}(\frac{1}{\varepsilon}^{|\mathcal{B}_0|/\varepsilon}) = 2^{\mathcal{O}(\varepsilon^{-4}\log^2(1/\varepsilon))}.$$

With $\alpha = \Theta(\delta^4)$ and $A = \mathcal{O}(\delta^{-2})$ we have

$$|\mathcal{B}_0| \cdot A \cdot \frac{2\gamma}{\alpha} \cdot \frac{1+2\delta}{\delta} = \mathcal{O}(\delta^{-7}\varepsilon^{-6}\log^2(1/\varepsilon)) = 2^{\mathcal{O}(\varepsilon^{-4}\log^2(1/\varepsilon))}.$$

Therefore, the steps described above take time $g(1/\varepsilon) \cdot n^{\mathcal{O}(f(1/\varepsilon))}$ for some function g and $f(1/\varepsilon) = 2^{\mathcal{O}(\varepsilon^{-4}\log^2(1/\varepsilon))}$.

9.1.2.3 Layers for Small Jobs

We can now compute the free layers of height δ^2 between the large wide jobs allocated by the dynamic program and the gaps designated for the large narrow jobs. Let $L_1,\ldots,L_F$ denote the free layers, each having m_f processors for $f \in [F]$. In Figure 9.3 an allocation of the enumerated large wide jobs and a guess Π for the gaps reserved for the large narrow jobs in $\mathcal{B}_0$ is illustrated. The empty spaces of height δ^2 between and next to the large narrow and large wide jobs represent the layers for small jobs.

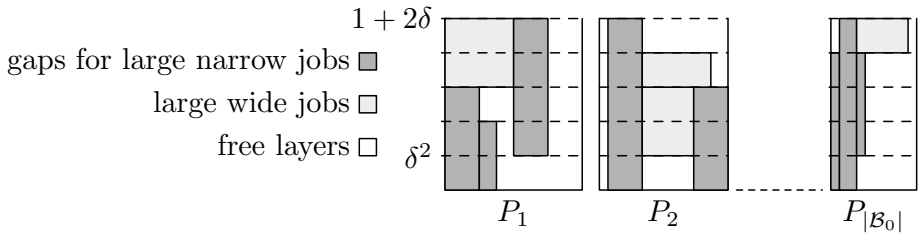

Figure 9.3: Simplified structure of large jobs in $\mathcal{B}_0$.

9.1.3 Linear Program for the Remaining Jobs $\mathcal{J}'$

In this section we give a linear programming relaxation for the following problem:

- place a set of small jobs $\mathcal{J}_{small}(\mathcal{B}_0) \subset \mathcal{J}_{small}$ into the layers $L_1, \ldots, L_F$
- select large narrow jobs $\mathcal{J}_{la-na}(\mathcal{B}_0) \subset \mathcal{J}_{la-na}$ to be placed into the gaps Π,
- fractionally place the remaining jobs into $\widetilde{\mathcal{B}}_1$.

We then round the solution of the LP and obtain an approximate and almost integral solution for the problem. For integrally scheduling the large jobs in the platforms of $\widetilde{\mathcal{B}}_1$ we round their processing times in advance harmonically as described in the next section.

9.1.3.1 Harmonic Rounding

The harmonic transformation was first introduced by Lee and Lee [48]. For $k \in \mathbb{N}$ it is described by a function $f_k : [0,1] \longrightarrow [0,1]$ with $f_k(x) = 1/q$ for $x \in (1/(q+1), 1/q]$ for $q = 1, \ldots, k-1$ and $f_k(x) = kx/(k-1)$ if $x \in (0, 1/k]$. The harmonic transformation has the following property:

Lemma 9.2. *[48] For a sequence $x_1, \ldots, x_n$ with $x_i \in (0,1]$ for $i \in [n]$ and $\sum_{i=1}^n x_i \leq 1$ we have $\sum_{i=1}^n f_k(x_i) \leq T_k$, where $T_k \leq T_\infty + 1/(k-1)$ and $T_\infty \approx 1.691$ is the Harmonic constant.*

We use a slightly modified function $h_k : [0,1] \longrightarrow [0,1]$, $h_k(x) = f_k(x)$ for $x > 1/k$ and $h_k(x) = x$ for $x \leq 1/k$. According to [4] for a sequence of numbers $x_1, \ldots, x_n$ with values in $(0,1]$ and $\sum_{i=1}^n x_i \leq 1$ we have $\sum_{i=1}^n h_k(x_i) \leq T_\infty$. For $k = \lfloor \frac{20}{\varepsilon} \rfloor$ we round the processing times of the jobs in $\mathcal{J}'$ via h_k. Since $\varepsilon \leq 3/18$, we have $k = \frac{20}{\varepsilon} \geq 120$.

For each job $j \in \mathcal{J}'$ let $\widetilde{p}_j := h_k(p_j) \in (0,1]$ denote its harmonically rounded processing time. In fact, we only modify the processing times of large jobs in $\mathcal{J}'$, because the small and medium jobs have processing times $p_j \leq \delta \leq \varepsilon/20 = 1/k$. Consequently, for all small and medium jobs we have $\widetilde{p}_j = p_j$. It might also be possible that there are large jobs with processing time $1/k \geq p_j > \delta$ for which we have $p_j = \widetilde{p}_j$.

Lemma 9.3. *Assume that the choice of $\mathcal{J}_{la-wi}(\mathcal{B}_0)$ was correct and let $\mathcal{J}^\star(\mathcal{B}_1) \subset \mathcal{J}'$ be the subset of jobs scheduled in $\mathcal{B}_1$ in an optimum solution. If the processing times of the jobs in $\mathcal{J}^\star(\mathcal{B}_1)$ are rounded harmonically, an optimum schedule of the rounded jobs into $\mathcal{B}_1$ (and therefore in $\widetilde{B}_1$) has makespan at most T_∞.*

Proof. The Lemma follows from the fact that for a sequence of numbers $x_1, \ldots, x_n$ with values in $(0,1]$ and $\sum_{i=1}^n x_i \leq 1$ we have $\sum_{i=1}^n h_k(x_i) \leq T_\infty$ [4, 48] and using a result of Seiden and van Stee [61]. □

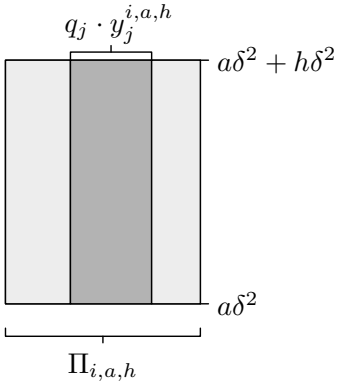

Figure 9.4: Slice of job j with $\bar{p}_j = h\delta^2$ in gap $\Pi_{i,a,h}$ in $P_i \in \mathcal{B}_0$.

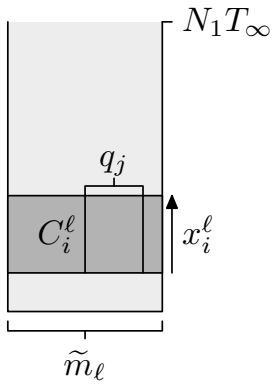

Figure 9.5: Job j in configuration C_i^ℓ in $\widetilde{B}_\ell \subseteq \widetilde{\mathcal{B}}_1$.

9.1.3.2 LP-Formulation.

For every group $\widetilde{B}_\ell$ we introduce a set $\mathcal{C}^\ell$ of feasible configurations $C^\ell : \mathcal{J}' \to \{0,1\}$ representing a subset of jobs in $\mathcal{J}'$ that fit next to each other in a platform with $\widetilde{m}_\ell$ processors, i.e. $\sum_{\{j \in \mathcal{J}' | C^\ell(j)=1\}} q_j \leq \widetilde{m}_\ell$. Let $q(\ell)$ denote the number of different configurations for $\widetilde{B}_\ell$. In the LP below the variable x_i^ℓ indicates the length of configuration C_i^ℓ for $i \in [q(\ell)]$ (compare to Figure 9.5). That means each job in $\{j \in \mathcal{J}' | C^\ell(j) = 1\}$ is executed in $\widetilde{B}_\ell$ during time x_i^ℓ. In a similar way, for each layer L_f in $\mathcal{B}_0$ we introduce

a set $\mathcal{C}^f$ of feasible configurations $C^f : \mathcal{J}_{small} \to \{0,1\}$ of small jobs that fit next to each other on m_f processors and denote with $q(f)$ the number of different configurations for L_f. The variable x_i^f indicates the length of configuration C_i^f for $i \in [q(f)]$. Furthermore, for every job $j \in \mathcal{J}_{la-na}$ we introduce variables $y_j^{i,a,h} \in [0,1]$, each $y_j^{i,a,h}$ indicates the vertical slice of job j (with $\bar{p}_j = h\delta^2$) that is assigned to a gap $\Pi_{i,a,h}$ in $\mathcal{B}_0$ (compare to Figure 9.4) .

(9.1)

$$\sum_{i=1}^{q(\ell)} x_i^\ell \leq N_1 T_\infty \quad \ell \in [L]$$

$$\sum_{i=1}^{q(f)} x_i^f \leq \delta^2 \quad f \in [F]$$

$$\sum_{\{j \in \mathcal{J}_{la-na} | \bar{p}_j = h\delta^2\}} y_j^{i,a,h} \cdot q_j \leq \Pi_{i,a,h} + \lfloor \alpha m_N \rfloor \quad i \in [|\mathcal{B}_0|], a \in [A], h \in [H]$$

$$\sum_{f=1}^{F} \sum_{\{i \in [q(f)] | C_i^f(j)=1\}} x_i^f + \sum_{\ell=1}^{L} \sum_{\{i \in [q(\ell)] | C_i^\ell(j)=1\}} x_i^\ell \geq \widetilde{p}_j = p_j \quad j \in \mathcal{J}_{small}$$

$$\sum_{\ell=1}^{L} \sum_{\{i \in [q(\ell)] | C_i^\ell(j)=1\}} x_i^\ell \geq \widetilde{p}_j \quad j \in \mathcal{J}_{la-wi} \setminus \mathcal{J}_{la-wi}(\mathcal{B}_0)$$

$$\sum_{\ell=1}^{L} \sum_{\{i \in [q(\ell)] | C_i^\ell(j)=1\}} x_i^\ell \cdot q_j + \tilde{p}_j \cdot \sum_{\substack{i,a,h \\ \bar{p}_j = h\delta^2}} y_j^{i,a,h} \cdot q_j \geq \tilde{p}_j \cdot q_j \quad j \in \mathcal{J}_{la-na}$$

$$\sum_{j \in \mathcal{J}_\tau} p_j q_j - \sum_{j \in \mathcal{J}_\tau} \sum_{\ell=1}^{L} \sum_{\{i \in [q(\ell)] | C_i^\ell(j)=1\}} x_i^\ell q_j \leq \varepsilon m_1$$

$$x_i^f, x_i^\ell \geq 0$$

$$y_j^{i,a,h} \in [0,1]$$

In the LP above we have one constraint for every group $\tilde{B}_\ell$ that guarantees that the makespan of the fractional schedule corresponding to a feasible LP-solution does not exceed length T_∞ for any $P_i \in \tilde{B}_\ell$. In a similar way we have one constraint for every layer L_f. For each gap $\Pi_{i,a,h}$ a constraint guarantees that the total load of large narrow jobs (fractionally) assigned to the gap does not exceed $\Pi_{i,a,h} + \lfloor \alpha m_N \rfloor$. For every small job we have a covering constraint combined from heights of configurations in

L_f and in B_ℓ. Furthermore, we have a covering constraint for each large wide job that is not placed in $\mathcal{B}_0$, i.e. $j \in \mathcal{J}_{la-wi} \setminus \mathcal{J}_{la-wi}(\mathcal{B}_0)$. Every large narrow job $j \in \mathcal{J}_{la-na}$ is covered by an area constraint: The total width of job j assigned to $\mathcal{B}_0$ multiplied with its height $\tilde{p}_j$ in $\widetilde{\mathcal{B}}_1$ plus the area covered by configurations in $\widetilde{\mathcal{B}}_1$ should be at least $\tilde{p}_j q_j$. For the medium jobs $\mathcal{J}_\tau$ the last constraint ensures that the total area of uncovered medium jobs is small, i.e. less than εm_1. If the LP has no feasible solution either the enumerated set of large wide jobs was not correct, the choice of Π does not fit or the choice of δ, moreover the choice of τ, was not correct.

9.1.3.3 Solving the LP

We can compute an approximate solution of the linear program above by solving approximately a Max-Min Resource Sharing problem.

For $n' := |\mathcal{J}'|$ and $n_\tau = |\mathcal{J}_\tau|$ the linear program (9.1) is a feasibility problem with an exponential number of variables and $L + F + G + n' - n_\tau + 1 \leq L + F + G + n + 1$ constraints (not counting non-negativity constraints). We can formulate the problem as a fractional covering problem with convex set K and at most $n' - n_\tau + 1$ covering constraints. The system of inequalities (defined by concave functions) is given as

(9.2)

$$\frac{1}{\widetilde{p}_j}\left(\sum_{f=1}^{F}\sum_{\{i\in[q(f)]|C_i^f(j)=1\}} x_i^f + \sum_{\ell=1}^{L}\sum_{\{i\in[q(\ell)]|C_i^\ell(j)=1\}} x_i^\ell\right) \geq 1 \quad j \in \mathcal{J}_{small}$$

$$\frac{1}{\widetilde{p}_j}\sum_{\ell=1}^{L}\sum_{\{i\in[q(\ell)]|C_i^\ell(j)=1\}} x_i^\ell \geq 1 \quad j \in \mathcal{J}_{la-wi} \setminus \mathcal{J}_{la-wi}(\mathcal{B}_0)$$

$$\frac{1}{\widetilde{p}_j}\left(\sum_{\ell=1}^{L}\sum_{\{i\in[q(\ell)]|C_i^\ell(j)=1\}} x_i^\ell + \sum_{\substack{i,a,h \\ \tilde{p}_j = h\delta^2}} \tilde{p}_j y_j^{i,a,h}\right) \geq 1 \quad j \in \mathcal{J}_{la-na}$$

$$\frac{1}{\sum_{j\in\mathcal{J}_\tau}\tilde{p}_j q_j - \varepsilon m_1}\left(\sum_{j\in\mathcal{J}_\tau}\sum_{\ell=1}^{L}\sum_{\{i\in[q(\ell)]|C_i^\ell(j)=1\}} x_i^\ell q_j\right) \geq 1$$

for a column vector $(x,y) \in K$ where

$$K := \left(\bigtimes_{\ell=1}^{L} K^{\ell} \times \bigtimes_{f=1}^{F} K^{f} \right) \times \bigtimes_{i,a,h} K^{i,a,h}$$

is given by the cartesian product of $L+F$ simplices

$$K^{\ell} := \left\{ x^{\ell} = (x_i^{\ell})_{i\in[q(\ell)]} | \sum_{i=1}^{q(\ell)} x_i^{\ell} = N_1 T_{\infty},\ x_i^{\ell} \geq 0 \right\},$$

$$K_f := \left\{ x^{f} = (x_i^{f})_{i\in[q(f)]} | \sum_{i=1}^{q(f)} x_i^{f} = \delta^2,\ x_i^{f} \geq 0 \right\}$$

and for $\mathcal{J}_{la-na}^{h} := \{j \in \mathcal{J}_{la-na} | \bar{p}_j = h\delta^2\}$ we have F sets of the form

$$K^{i,a,h} := \left\{ y^{i,a,h} = (y_j^{i,a,h})_{j\in\mathcal{J}_{la-na}^{h}} \mid \sum_{j=1}^{|\mathcal{J}_{la-na}^{h}|} y_j^{i,a,h} q_j = \Pi_{i,a,h} + \lfloor \alpha m_N \rfloor,\ y_j^{i,a,h} \in [0,1] \right\}.$$

We represent the system of inequalities (9.2) as $A(x,y) \geq e$ where e is the vector of all ones and A consists of the row vectors $a_1, \ldots, a_{n'-n_\tau+1}$. We can get a feasible (approximate) solution of the covering problem computing an approximate solution of the following MAX-MIN RESOURCE SHARING problem

$$\lambda^{\star} = \max\{\lambda | A(x,y) \geq \lambda e, (x,y) \in K\}. \tag{9.3}$$

According to [24] we can compute a $(1-\rho)$-approximate solution for (9.3) in $\mathcal{O}(n(\rho^{-2} + \log n))$ iterations: We start the iteration with an initial solution $(\bar{x},\bar{y}) \in K$ and compute a price vector $b = (b(\bar{x},\bar{y})) \in \mathbb{Q}^{n'-n_\tau+1}$ depending on $(\bar{x},\bar{y})$. Then we compute a $(1-\rho')$-approximate solution $(\hat{x},\hat{y}) \in K$ for the block problem $\max\{b^{\top} A(x,y) | (x,y) \in K\}$ for $\rho' = \rho/6$ and reset $(\bar{x},\bar{y}) := (1-\tau)(\bar{x},\bar{y}) + \tau(\hat{x},\hat{y})$ for a certain step width $\tau \in (0,1)$. After $\mathcal{O}(n(\rho^{-2} + \log n))$ the algorithm stops at a vector $(\bar{x},\bar{y}) \in K$ so that $A(\bar{x},\bar{y}) \geq (1-\rho)\lambda^{\star} e$. For $(x,y) \in K$ we define $\lambda(x,y) := \min\{a_j \cdot (x,y) | 1 \leq j \leq n' - n_\tau + 1\}$ and proceed by case distinction.

If $\lambda(\bar{x},\bar{y}) < 1-\rho$ we conclude that there is no solution $(x,y) \in K$ satisfying $A(x,y) \geq e$ and so for (9.1). We discard the vector Π and the

assignment V of large wide jobs and compute another pair (V, Π). If all possible pairs (V, Π) fail we increase the value for τ.

In case of $\lambda(\bar{x}, \bar{y}) \geq (1-\rho)$ we have $\lambda^\star \geq 1$ and can therefore compute a fractional schedule from $(\bar{x}, \bar{y})$ as follows.

We slightly extend the length of each configuration setting $(x, y) := (1 + \frac{\rho}{1-\rho})(\bar{x}, \bar{y})$ to achieve

$$a_j(x, y) = (1 + \frac{\rho}{1-\rho}) a_j(\bar{x}, \bar{y}) \geq (1 + \frac{\rho}{1-\rho})(1-\rho) \geq 1$$

for $j \in [n' - n_\tau + 1]$. Consequently, for every $\ell \in [L]$ we have $\sum_{i=1}^{q(\ell)} x_i^\ell = (1 + \frac{\rho}{1-\rho}) \sum_{i=1}^{q(\ell)} \bar{x}_i^\ell = (1 + \frac{\rho}{1-\rho}) N_1 T_\infty$. The fractional schedule in each Layer L_f and therefore in each of the platforms in group $\mathcal{B}_0$ is also increased by factor $(1 + \frac{\rho}{1-\rho})$. The same holds for the total width of large narrow jobs assigned to a gap in $\mathcal{B}_0$. So the width of a gap is also increased by the same factor . For ρ sufficient small, namely $\frac{\rho}{1-\rho} \leq \frac{\alpha}{2\gamma} \leq 1/2$, using $\Pi_{i,a,h} \leq m_1$ and $\frac{m_1 \alpha}{2\gamma} \leq \lfloor \alpha m_N \rfloor$ the increased width can be bounded by

$$(1 + \frac{\alpha}{2\gamma})(\Pi_{i,a,h} + \lfloor \alpha m_N \rfloor) \leq \Pi_{i,a,h} + 3\lfloor \alpha m_N \rfloor. \tag{9.4}$$

For $\frac{\rho}{1-\rho} = \frac{\alpha}{2\gamma}$ we have

$$\frac{\rho}{1-\rho} = \frac{\alpha}{2\gamma} \overset{\alpha = \delta^4/16, \delta \leq \varepsilon, \gamma \geq 1}{\leq} \delta^4/16 \leq \varepsilon^4.$$

and $\rho = \frac{\alpha}{2\gamma + \alpha} = \Theta(\alpha/\gamma) = \Theta(\delta^4 \varepsilon^3 \log^{-1}(1/\varepsilon))$ if $N_1 = \Theta(1/\varepsilon^3 \log(1/\varepsilon))$ since $\gamma = \frac{8}{3} N_1$.

Solving the Block Problem. The block problem $\max\{b^\top A(x, y) | (x, y) \in K\}$ can be decomposed into $L + F + G$ independent smaller block problems. We introduce a weighting $w \in \mathbb{Q}^{n'}$ with

$$w_j := \begin{cases} \frac{b_j}{\tilde{p}_j} \text{ if } j \in \mathcal{J}' \setminus \mathcal{J}_\tau \\ \frac{q_j b_{n'-n_\tau+1}}{\sum_{k \in \mathcal{J}_\tau} p_k q_k - \varepsilon m_1} \text{ if } j \in \mathcal{J}_\tau. \end{cases}$$

Then for $(x,y) \in K$ we obtain after some rearrangement

$$b^\top A(x,y) = \sum_{\ell=1}^{L} \sum_{j\in\mathcal{J}'} \sum_{\{i\in[q(\ell)]|C_i^\ell(j)=1\}} w_j x_i^\ell + \sum_{f=1}^{F} \sum_{j\in\mathcal{J}_{small}} \sum_{\{i\in[q(f)]|C_i^f(j)=1\}} w_j x_i^f + \sum_{i,a,h} \sum_{j\in\mathcal{J}_{la-na}^h} w_j \tilde{p}_j y_j^{i,a,h}.$$

So the block problem can be rewritten as

$$\begin{aligned} \max & \sum_{\ell=1}^{L} \sum_{j\in\mathcal{J}'} \sum_{\{i\in[q(\ell)]|C_i^\ell(j)=1\}} w_j x_i^\ell \\ & + \sum_{f=1}^{F} \sum_{j\in\mathcal{J}_{small}} \sum_{\{i\in[q(f)]|C_i^f(j)=1\}} w_j x_i^f \\ & + \sum_{i,a,h} \sum_{j\in\mathcal{J}_{la-na}^h} b_j y_j^{i,a,h} \\ x^\ell & \in K^\ell \quad \text{for } \ell \in [L] \\ x^f & \in K^f \quad \text{for} f \in [F] \\ y^{i,a,h} & \in K^{i,a,h} \quad \text{for all } (i,a,h). \end{aligned} \tag{9.5}$$

For solving (9.5) it is sufficient to compute a solution for L problems of the kind

$$\begin{aligned} \max & \sum_{j\in\mathcal{J}'} \sum_{\{i\in[q(\ell)]|C_i^\ell(j)=1\}} w_j x_i^\ell \\ & x^\ell \in K^\ell \end{aligned} \tag{9.6}$$

and F problems

$$\begin{aligned} \max & \sum_{j\in\mathcal{J}_{small}} \sum_{\{i\in[q(f)]|C_i^f(j)=1\}} w_j x_i^f \\ & x^f \in K^f \end{aligned} \tag{9.7}$$

and G problems of the form

$$\begin{aligned} \max & \sum_{j\in\mathcal{J}_{la-na}^h} b_j y_j^{i,a,h} \\ & y^{i,a,h} \in K^{i,a,h} \end{aligned} \tag{9.8}$$

Each of the last G problems corresponds to a fractional 0-1 knapsack problem of the following form

$$\begin{aligned} &\max \sum_j b_j x_j \\ &\text{s.t. } \sum_j q_j x_j \leq \Pi_{i,a,h} + \lfloor \alpha m_N \rfloor \\ &\qquad x_j \in [0,1]\ j \in \mathcal{J}^h_{la-na}. \end{aligned} \tag{9.9}$$

Which can be solved in time $\mathcal{O}(n \log n)$. As K^ℓ and K^f are simplices we find the optimum of (9.6) and (9.7) at a vertex $\widetilde{x}^\ell \in K^\ell$ and $\bar{x}^f \in K^f$, respectively. For K^ℓ such a vertex corresponds to a configuration $C^\ell_{\tilde{i}}$ with $\widetilde{x}^\ell_{\tilde{i}} = N_1 T_\infty$ and $\widetilde{x}^\ell_i = 0$ for $C^\ell_i \neq C^\ell_{\tilde{i}}$. That means for solving (9.6) we have to find a configuration $C^\ell_{\tilde{i}}$ with maximum weight $\sum_{\{j \in \mathcal{J}' | C^\ell_i(j)=1\}} w_j$. This can be formulated as a knapsack problem:

$$\begin{aligned} &\max \sum_{j \in \mathcal{J}'} w_j x_j \\ &\text{s.t. } \sum_{j \in \mathcal{J}'} q_j x_j \leq \tilde{m}_\ell \\ &\qquad x_j \in \{0,1\} \end{aligned} \tag{9.10}$$

Finding an optimum solution of (9.7) corresponds to

$$\begin{aligned} &\max \sum_{j \in \mathcal{J}_{small}} w_j x_j \\ &\text{s.t. } \sum_{j \in \mathcal{J}_{small}} q_j x_j \leq m_f \\ &\qquad x_j \in \{0,1\}, \end{aligned} \tag{9.11}$$

where the value m_f denotes the number of processors of the free layer L^f of height δ^2 in $\mathcal{B}_0$. Using the algorithm by Lawler [47] we compute a $(1-\rho')$ approximate solution for each of the knapsack problems in time $\mathcal{O}(n \log(1/\rho') + 1/\rho'^4)$. Summing up all near optimal solutions for (9.6) and (9.7) gives a $(1-\rho')$-approximate solution for (9.5).

This implies that the Max-Min Resource Sharing problem can be solved approximately with accuracy $\rho \leq \frac{\alpha}{2\gamma+\alpha}$ in time $poly(n, 1/\rho)$ where $\rho = \mathcal{O}(\alpha\gamma^{-1}) = \mathcal{O}(\delta^4\gamma^{-1})$.

9.1.4 Rounding the LP-solution

A solution $((x^f),(x^\ell),(y_j^{i,a,h}))$ of (9.1) can be transformed into a fractional solution of a general assignment problem. The fractional assignment can be rounded using a result of Lenstra et al. [50] for scheduling unrelated machines similar as in Section 8.3. The main difficulty and difference here is to handle the large narrow narrow jobs as they are placed fractionally in $\mathcal{B}_0$ and $\widetilde{B}_0$.

For all jobs $j \in \mathcal{J}'$ we introduce new variables $x^\ell(j), x^f(j) \geq 0$ that indicate the length of the fraction of job j that is scheduled in $\widetilde{B}_\ell$, $\ell \geq 1$, and in L_f, respectively. Formally this is

$$x^\ell(j) = \sum_{\{i \in [q(\ell)] | C_i^\ell(j)=1\}} x_i^\ell$$

the sum of the length of all configurations in $\widetilde{B}_\ell$ in which job j appears, and

$$x^f(j) = \sum_{\{i \in [q(f)] | C_i^f(j)=1\}} x_i^f.$$

Additionally, for medium jobs $j \in \mathcal{J}_\tau$ (here again $\tilde{p}_j = p_j$) we define

$$x^0(j) := p_j - \sum_{\ell=1}^{L} x^\ell(j) \in [0, \tilde{p}_j],$$

the fraction of the job that should be placed in $\mathcal{B}_0 = B_0$. For the medium jobs assigned to $\mathcal{B}_0$ we therefore have the inequality

$$\varepsilon m_1 \geq \sum_{j \in \mathcal{J}_\tau} x^0(j) q_j.$$

Then the following covering constraints hold for the jobs:

$$\begin{aligned}
\sum_{\ell=1}^{L} x^{\ell}(j) + \sum_{f=1}^{F} x^{f}(j) &\geq \widetilde{p}_j = p_j,\ j \in \mathcal{J}_{small} \\
x^{0}(j) + \sum_{\ell=1}^{L} x^{\ell}(j) &= \tilde{p}_j = p_j\ j \in \mathcal{J}_{\tau} \\
\sum_{\ell=1}^{L} x^{\ell}(j) &\geq \widetilde{p}_j,\ j \in \mathcal{J}_{la-wi} \setminus \mathcal{J}_{la-wi}(\mathcal{B}_0) \\
\sum_{\ell=1}^{L} x^{\ell}(j) q_j + \tilde{p}_j \cdot \sum_{\substack{i,a,h \\ \tilde{p}_j = h\delta^2}} y_j^{i,a,h} \cdot q_j &\geq \tilde{p}_j \cdot q_j\ j \in \mathcal{J}_{la-na}
\end{aligned} \tag{9.12}$$

By deleting job j from appropriate configurations or replace a configuration by two "shorter" configurations (one with job j and one without, their total length is the same as the one of the original configuration) we may assume equality in each of the inequalities above. For the same reason we may also assume that $x^f(j), x^\ell(j) \in [0, \tilde{p}_j]$ now. For all fractions $x^\ell(j)$,$x^f(j)$,$x^0(j)$ we build rectangles of width q_j and height $x^\ell(j)$, $x^f(j)$ and $x^0(j)$, respectively.

The rectangles belonging to fractions of medium jobs for B_0 we simply collect in a set

$$\mathcal{R}^0 := \{(x^0(j), q_j) | j \in \mathcal{J}_{\tau}\}.$$

For $\varepsilon' := \varepsilon/9$ we partition the rectangles of every group $\widetilde{B}_\ell$, $\ell \geq 1$, and Layer L_f into wide rectangles

$$\begin{aligned}
\mathcal{R}^{\ell}_{wide} &:= \{(x^{\ell}(j), q_j) | q_j > (\varepsilon'/2)\widetilde{m}_\ell\} \\
\mathcal{R}^{f}_{wide} &:= \{(x^{f}(j), q_j) | q_j > (\varepsilon'/2) m_f\}
\end{aligned}$$

and narrow rectangles

$$\begin{aligned}
\mathcal{R}^{\ell}_{narrow} &:= \{(x^{\ell}(j), q_j) | q_j \leq (\varepsilon'/2)\widetilde{m}_\ell\} \\
\mathcal{R}^{f}_{narrow} &:= \{(x^{f}(j), q_j) | q_j \leq (\varepsilon'/2) m_f\}.
\end{aligned}$$

The wide rectangles in $\mathcal{R}^{\ell}_{wide}$, $\mathcal{R}^{f}_{wide}$ are partitioned further (by width) into $M = \mathcal{O}(1/\varepsilon' \log(1/\varepsilon'))$ groups G^{ℓ}_k using Step 1 to 5 of the rounding technique described in Algorithm 6. For each set $\mathcal{R}^{\ell}_{wide}$ and group

$k = 1, \ldots, M$ of $\mathcal{R}^{\ell}_{wide}$ we introduce a variable $x^{\ell}_k(j) \in [0, \tilde{p}_j]$ that indicates the fraction of Job $j = 1, \ldots, n$ that is contained in this group. In a similar way we introduce variables $x^f_k(j)$. For the fraction of job j in $\mathcal{R}^{\ell}_{narrow}$ and $\mathcal{R}^f_{narrow}$ we introduce variables $x^{\ell}_0(j),\ x^f_0(j) \in [0,1]$, respectively. Note that $\sum_{k=0}^{M} x^f_k(j) = x^f(j)$ and $\sum_{k=0}^{M} x^{\ell}_k(j) = x^{\ell}(j)$ holds by construction. We scale all variables $x^f_k(j)$, $x^{\ell}_k(j)$, $x^0(j)$ by $1/\tilde{p}_j$ and introduce the corresponding variables $z^f_k(j)$, $z^{\ell}_k(j)$, $z^0(j) \in [0,1]$. Then we can rewrite (9.12) (dividing every equation by its right hand side):

$$
\begin{aligned}
\sum_{\ell=1}^{L}\sum_{k=0}^{M} z^{\ell}_k(j) + \sum_{f=1}^{F}\sum_{k=0}^{M} z^f_k(j) &= 1,\ j \in \mathcal{J}_{small} \\
x^0(j) + \sum_{\ell=1}^{L}\sum_{k=0}^{M} z^{\ell}_k(j) &= 1\ j \in \mathcal{J}_{\tau} \\
\sum_{\ell=1}^{L}\sum_{k=0}^{M} z^{\ell}_k(j) &= 1,\ j \in \mathcal{J}_{la-wi} \setminus \mathcal{J}_{la-wi}(\mathcal{B}_0) \\
\sum_{\ell=1}^{L}\sum_{k=0}^{M} z^{\ell}_k(j) + \sum_{\substack{i,a,h \\ \bar{p}_j = h\delta^2}} y^{i,a,h}_j &= 1\ j \in \mathcal{J}_{la-na}
\end{aligned}
\tag{9.13}
$$

We compute $SIZE(\mathcal{R}^0)(\leq \varepsilon m_1)$, $SIZE(\mathcal{R}^{\ell}_{narrow})$ and $SIZE(\mathcal{R}^f_{narrow})$ and observe that the following capacity constraints hold:

$$
\begin{aligned}
\sum_{j \in \mathcal{J}_{small}} z^f_0(j) \cdot \tilde{p}_j \cdot q_j &\leq SIZE(\mathcal{R}^f_{narrow})\ \ f \in [F] \\
\sum_{j \in \mathcal{J}_{small}} z^f_k(j) \cdot \tilde{p}_j &\leq H(G^f_k)\ \ k \in [M], f \in [F] \\
\sum_{j \in \mathcal{J}_{\tau}} z^0(j) \cdot \tilde{p}_j \cdot q_j &\leq SIZE(\mathcal{R}^0) \\
\sum_{j=1}^{n} z^{\ell}_0(j) \cdot \tilde{p}_j \cdot q_j &\leq SIZE(\mathcal{R}^{\ell}_{narrow})\ \ \ell \in [L] \\
\sum_{j=1}^{n} z^{\ell}_k(j) \cdot \tilde{p}_j &\leq H(G^{\ell}_k)\ \ k \in [M], \ell \in [L] \\
\sum_{\{j \in \mathcal{J}_{la-na} \mid \bar{p}_j = h\delta^2\}} y^{i,a,h}_j \cdot q_j &\leq \Pi_{i,a,h} + \lfloor \alpha m_N \rfloor\ \ i \in [|\mathcal{B}_0|], a \in [A], h \in [H]
\end{aligned}
\tag{9.14}
$$

Now we observe that $(z^f_k(j),\ z^{\ell}_k(j),\ z^0(j), y^{i,a,h}_j)$ is a fractional solution of

a general assignment problem formulated by (9.13) and (9.14). This assignment problem corresponds to scheduling n jobs on $|\mathcal{B}_0| \cdot A \cdot H + (F + L)(M+1) + 1$ unrelated machines. Using a result by Lenstra et al. [50] a fractional solution of this problem can be rounded to an almost integral one with only one fractionally assigned job per machine.

Let $(\tilde{z}_k^f(j), \tilde{z}_k^\ell(j), \tilde{z}^0(j), \tilde{y}_j^{i,a,h})$ be such a rounded solution. Define

$$\begin{aligned}
\widetilde{\mathcal{R}}^\ell_{wide} &:= \{(\tilde{z}_k^\ell(j)\tilde{p}_j, q_j) | \tilde{z}_k^\ell(j) = 1, k > 0\},\\
\widetilde{\mathcal{R}}^\ell_{narrow} &:= \{(\tilde{z}_k^\ell(j)\tilde{p}_j, q_j) | \tilde{z}_k^\ell(j) = 1, k = 0\},\\
Frac^\ell &:= \{(\tilde{z}_k^\ell(j)\tilde{p}_j, q_j) | \tilde{z}_k^\ell(j) < 1\},\\
\widetilde{\mathcal{R}}^f_{wide} &:= \{(\tilde{z}_k^f(j)\tilde{p}_j, q_j) | \tilde{z}_k^f(j) = 1, k > 0\},\\
\widetilde{\mathcal{R}}^f_{narrow} &:= \{(\tilde{z}_k^\ell(j)\tilde{p}_j, q_j) | \tilde{z}_k^f(j) = 1, k = 0\},\\
Frac^f &:= \{(\tilde{z}_k^f(j)\tilde{p}_j, q_j) | \tilde{z}_k^f(j) < 1\},\\
\widetilde{\mathcal{R}}^0 &:= \{(z^0(j)\tilde{p}_j, q_j) | j \in \mathcal{J}_\tau, z^0(j) = 1\},\\
frac^0 &:= (z^0(j)\tilde{p}_j, q_j) \text{ with } j \in \mathcal{J}_\tau \text{ and } z^0(j) < 1.
\end{aligned}$$

For every group $\widetilde{B}_\ell$ we obtain a set of integrally assigned wide $\widetilde{\mathcal{R}}^\ell_{wide}$ and narrow rectangles $\widetilde{\mathcal{R}}^\ell_{narrow}$ and M fractionally assigned wide rectangles plus one fractional narrow job which we collect in $Frac^\ell$. For every layer L_f we obtain in a similar way sets of integral rectangles corresponding to small jobs $\widetilde{\mathcal{R}}^f_{wide}$, $\widetilde{\mathcal{R}}^f_{narrow}$ and $M+1$ fractional small jobs (M wide and one narrow rectangles) collected in $Frac^f$.

In addition, we have a set of integral rectangles $\widetilde{\mathcal{R}}^0$ corresponding to a set of medium jobs to be scheduled in $\mathcal{B}_0$ with total load $SIZE(\widetilde{\mathcal{R}}^0) \leq SIZE(\mathcal{R}^0) \leq \varepsilon m_1$ plus one fractional medium job $frac^0$ with processing time $< \delta$. We schedule the jobs in $\widetilde{\mathcal{R}}^0 \cup \{frac^0\} (= \mathcal{J}_\tau(\mathcal{B}_0))$ using list schedule on top of the largest platform P_1 in the end. This will increase the length of the schedule in P_1 by at most $2\varepsilon + \delta \leq 3\varepsilon$.

9.1.5 Packing into the Gaps

For every gap with width $\Pi_{i,a,h}$ we have rounded variables $\tilde{y}_j^{i,a,h}$. Except for one value with $\tilde{y}_j^{i,a,h} < 1$ all of them are integral $\tilde{y}_j^{i,a,h} = 1$ and the widths of the corresponding jobs (completely including the fractional one)

sum up to at most $\Pi_{i,a,h} + 2\lfloor \alpha m_N \rfloor$. Since $\Pi_{i,a,h} \leq \Pi^{\star}_{i,a,h} \leq \Pi_{i,a,h} + \lfloor \alpha m_N \rfloor$ we need to remove large narrow jobs of total width at most $3\lfloor \alpha m_N \rfloor$ for every gap that cannot be finished before $1 + 2\delta$ and schedule them on top of the solution. For all gaps their total width sums up to

$$|\mathcal{B}_0| \cdot 3\lfloor \alpha m_N \rfloor \cdot A \cdot H \leq |\mathcal{B}_0| \frac{3(1+2\delta)\lfloor \alpha m_N \rfloor}{\delta^4} \leq |\mathcal{B}_0| \cdot \frac{4\lfloor \alpha m_N \rfloor}{\delta^4}.$$

As those additional large narrow jobs have small width, placing some of them next to each other in a platform $P_i \in \mathcal{B}_0$, we use at least $m_i - \lfloor \alpha m_N \rfloor \geq m_N - \alpha m_N = (1-\alpha) m_N$ processors of this platform. Since $\frac{|\mathcal{B}_0| \cdot 4\lfloor \alpha m_N \rfloor}{\delta^4 (1-\alpha) m_N} \overset{\alpha \leq 1/2}{\leq} \frac{|\mathcal{B}_0| \cdot 8\alpha}{\delta^4}$ we need at most $\left\lceil \frac{|\mathcal{B}_0| \cdot 8\alpha}{\delta^4} \right\rceil$ platforms to schedule those jobs. In those platforms the schedule is increased by 1.

9.1.6 Packing into the Layers

We pack the rectangles in $\widetilde{\mathcal{R}}^f_{wide}$, $\widetilde{\mathcal{R}}^f_{narrow}$ with the Strip Packing subroutine Algorithm 7 into the layers L^f and add the rectangles in $Frac^f$ greedily at the end. According to Theorem 5.2 for $M = \mathcal{O}(1/\varepsilon' \log(1/\varepsilon'))$ in every layer we obtain a packing of height

$$\begin{aligned}(1+\varepsilon')\delta^2 + (1 + 4M + |Frac^f|) \max\{p_j | (p_j, q_j) \in \widetilde{\mathcal{R}}^f_{wide} \cup \widetilde{\mathcal{R}}^f_{narrow} \cup Frac^f\} \\ \overset{|Frac^f| \leq M+1}{\leq} (1+\varepsilon')\delta^2 + (2+5M)\delta^5.\end{aligned}$$

More precisely Lemma 5.4 gives $M \leq 5/\varepsilon'(\lfloor \log(2/\varepsilon') \rfloor + 1)$. We conclude with $\varepsilon' = \varepsilon/9 \overset{\varepsilon \leq 3/18}{\leq} 1/54$ that $\log(1/\varepsilon') \geq 5$ and thus

$$M \leq 5/\varepsilon' \log(2/\varepsilon) + 5/\varepsilon' = 5/\varepsilon' \log(1/\varepsilon') + 10/\varepsilon' \leq 7/\varepsilon' \log(1/\varepsilon').$$

Then we have

$$\begin{aligned}(1+\varepsilon')\delta^2 + (2+5M)\delta^5 &\leq \delta^2 + \delta^2 \left(\varepsilon' + 2\delta^3 + \frac{35\delta^3}{\varepsilon'} \log(1/\varepsilon') \right) \\ &\overset{\delta \leq \varepsilon/20 \leq \varepsilon'/2, \varepsilon \leq 1/2}{\leq} \delta^2 + \delta^2 \left(\frac{\varepsilon}{9} + \frac{\varepsilon}{2 \cdot 20^3} + \frac{35\varepsilon'^3}{8\varepsilon'} \log(1/\varepsilon') \right) \\ &\overset{\varepsilon' \leq \varepsilon/9}{\leq} \delta^2 + \delta^2 \left(\frac{\varepsilon}{9} + \frac{\varepsilon}{2 \cdot 20^3} + \frac{\varepsilon}{2} \right) \leq \delta^2 + \delta^2 \cdot \frac{2\varepsilon}{3}\end{aligned}$$

That means, we exceed the height of every layer by at most $\frac{2\varepsilon\delta^2}{3}$. Since there are at most $\frac{1+2\delta}{\delta^2} \overset{\delta\leq 1/20}{\leq} \frac{21}{20\delta^2}$ layers, we increase the length of the schedule in every platform in $\mathcal{B}_0$ at most by $\frac{21}{20\delta^2} \cdot \frac{2\varepsilon\delta^2}{3} = \frac{7\varepsilon}{10} \leq \varepsilon$.

So far we have scheduled a subset of the job into $\mathcal{B}_0$ so that almost all jobs can be finished before $1 + 2\delta + 4\varepsilon \leq 1 + 5\varepsilon$ except for some large narrow jobs that can be distributed within $\left\lceil \frac{|\mathcal{B}_0| \cdot 8\alpha}{\delta^4} \right\rceil$ extra platforms in $\mathcal{B}_0$. In those platforms the length of the schedule is at most $2 + 5\varepsilon$.

9.1.7 2D-Bin Packing Subroutine for $\widetilde{\mathcal{B}}_1$

We introduce here a subroutine for 2D-Bin Packing that packs the rectangles in $\widetilde{\mathcal{R}}^{\ell}_{wide} \cup \widetilde{\mathcal{R}}^{\ell}_{narrow} \cup Frac^{\ell}$ into $\widetilde{B}_{\ell}$, $\ell \in [L]$. A *two-dimensional bin* of width x and height y will be denoted with $b(x, y)$. In this context a *strip* is a bin of width 1 and infinite height $b(1, \infty)$. We also use the notation $b(x, \infty)$ for a strip of width x. If $x \in \mathbb{N}$ a strip $b(x, \infty)$ corresponds to a platform with x processors. Furthermore, remember that we assume an optimum makepsan equal to 1 and $p_{\max} \leq 1$. We define similar as in [4] the following property for integral packings.

Definition 9.4. A packing of a set of rectangles with heights $\in [0, 1]$ and width at most w into a strip $b(w, \infty)$ has the *tall not sliced property for* ε, if when drawing horizontal lines through the packing at heights $i = 1, 2, 3, \ldots$ no rectangle with height $> \varepsilon$ intersects with its interior such a horizontal line.

For packings having the tall not sliced property one can show "good cutting properties":

Lemma 9.5. *Let $\mathcal{R}$ be a set of rectangles with heights bounded by 1 and widths at most w that can be integrally packed into a strip $b(w, \infty)$ with height at most h. If the packing has the tall not sliced property for some ε, it can be converted into a 2-dimensional bin packing using at most $(h + 1)(1 + \varepsilon)$ bins $b(w, 1)$.*

Proof. By drawing horizontal lines at height $i = 1, \ldots, \lceil h \rceil$ through the packing we cut the packing into $\lceil h \rceil$ slices. The rectangles that lie completely with their interior between two consecutive horizontal lines can be packed into a bin $b(w, 1)$. This gives at most $h + 1$ bins. For every horizontal line we take out the rectangles intersecting it and get a slice of height at most ε of cut rectangles (as the packing has the tall not sliced property

for ε). We can pack $\frac{1}{\varepsilon}$ of such slices together into a bin $b(w,1)$. Since we have at most $\lfloor h \rfloor$ ε-slices we get another $\varepsilon(h+1)$ bins. □

The following lemma is derived by a result of Bansal et al. [4]

Lemma 9.6. *The rectangles in $\widetilde{\mathcal{R}}^{\ell}_{wide} \cup \widetilde{\mathcal{R}}^{\ell}_{narrow}$ can be packed integrally into a strip $b(\tilde{m}_\ell, \infty)$ obtaining a packing of height at most*

$$(1+\varepsilon')^2 T_\infty N_1 + (4M + (M+1)k)$$

having the tall not sliced property for $1/k$.

Proof. According to the solution of the configuration LP (9.1), a fractional strip packing for $\widetilde{\mathcal{R}}^{\ell}_{wide} \cup \widetilde{\mathcal{R}}^{\ell}_{narrow}$ into $b(\tilde{m}_\ell, \infty)$ has height at most

$$FSP(\widetilde{\mathcal{R}}^{\ell}_{wide} \cup \widetilde{\mathcal{R}}^{\ell}_{narrow}) \leq N_1 T_\infty.$$

As in Section 5.2 for a list of rectangles $\mathcal{R}$ with $FSP(\mathcal{R})$ we denote the height of an optimal solution for FRACTIONAL STRIP PACKING , i.e. the the height of the solution given by the linear program (5.1) for input $\mathcal{R}$. We group and round $\widetilde{\mathcal{R}}^{\ell}_{wide}$ using Algorithm 6 and obtain $\widetilde{\mathcal{R}}^{\ell}_{round}$ with only $M = \mathcal{O}(1/\varepsilon' \log(1/\varepsilon'))$ (remember $\varepsilon' = \varepsilon/9$) different width. Then we compute an optimum solution of (5.1) for $\widetilde{\mathcal{R}}^{\ell}_{round}$ (modulo scaling of rectangle width by $1/\tilde{m}_\ell$). From Lemma 6.3 we know that there are at most $2M$ non-zero configurations in the solution. With Lemma 8.4 we conclude

$$\begin{aligned} FSP(\widetilde{\mathcal{R}}^{\ell}_{round}) &\overset{L.8.4}{\leq} (1+\varepsilon')^2 FSP(\mathcal{R}^{\ell}_{wide}) \\ &\leq (1+\varepsilon')^2 N_1 T_\infty. \end{aligned}$$

Thus, we obtain a fractional packing of $\widetilde{\mathcal{R}}^{\ell}_{round}$ into $b(\tilde{m}_\ell, \infty)$ with at most $2M$ non-zero configurations and height at most $(1+\varepsilon')^2 N_1 T_\infty$. We can produce an integral packing as follows:
The rectangles in $\widetilde{\mathcal{R}}^{\ell}_{round} \cup \widetilde{\mathcal{R}}^{\ell}_{narrow} \cup Frac^{\ell}$ correspond to jobs that have harmonically rounded processing times. Thus, rectangles that correspond to jobs with processing times $> 1/k$ have heights in $\left\{ \frac{1}{q} \middle| \; q = 1, \ldots, k-1 \right\}$. Using a technique by Bansal et al. [4] the fractional packing of the wide rectangles $\widetilde{\mathcal{R}}^{\ell}_{round}$ can be converted into an integral packing with height $(1+\varepsilon')^2 T_\infty N_1 \; + (2M + Mk) p_{\max}$ having the tall not sliced property for

$1/k$: We first add an extra height of $p_{\max}(\leq 1)$ to each configuration. For each non-zero configuration we generate columns of different width according to the configuration, i.e. reserved space for the rectangles of the corresponding width. The height of each column is equal to the height of the corresponding configuration. This increases the height of the packing to $(1+\varepsilon')^2 T_\infty N_1 + 2Mp_{\max}$ (compare also to the proof of Lemma 6.4.) For each width w_i in $\widetilde{\mathcal{R}}^\ell_{round}$ we order the rectangles of width w_i and height $> 1/k$ by height and fill the columns of width w_i greedily starting at height 0. Whenever the height changes we shift the (vertical) starting position of the upcoming rectangle to the next integral. So it is guaranteed that rectangles of height $1/q$ always start at integral multiples of $1/q$. Since there are M different width and k different heights for the rectangles, this may happen at most Mk times. We do not care about the starting position of rectangles with height $\leq 1/k$. In total this increases the height of the packing again by Mk.

We use NFDH to place the narrow rectangles in $\widetilde{\mathcal{R}}^\ell_{narrow}$ in the empty space next to the configurations in a similar way. If the height changes we open a new level with baseline at the next integral. This increases the total height again by at most $(2M+k)$ since the configuration changes at most $2M$ times and the height changes k times. The final packing has height less than

$$(1+\varepsilon')^2 T_\infty N_1 + (4M + (M+1)k). \quad \square$$

Adding now the $M+1$ fractional rectangles in $Frac^\ell$ preserving the tall not sliced property for $1/k$ we get a strip packing for $\widetilde{\mathcal{R}}^\ell_{round} \cup \widetilde{\mathcal{R}}^\ell_{narrow} \cup Frac^\ell$ into $b(\tilde{m}_\ell, \infty)$ with height at most

$$\begin{aligned}(1+\varepsilon')^2 N_1 T_\infty + (M+1)k + 5M + 2 &\overset{k\geq 6, \varepsilon' \leq \varepsilon/9, M \geq 2}{\leq} (1+\varepsilon) N_1 T_\infty + 2Mk + Mk \\ &= (1+\varepsilon) N_1 T_\infty + 3Mk.\end{aligned}$$

For the appropriate choice of N_1 we can now convert the strip packing into a 2-dimensional bin packing using at most $2N_1$ bins $b(\tilde{m}_\ell, 1)$. Stacking now any two bins on top of each other gives packing into N_1 strips $b(\tilde{m}_\ell, \infty)$ of height 2 that corresponds to a schedule of length 2 for the platforms in $\widetilde{B}_\ell$.

Lemma 9.7. *For* $N_1 = \frac{(3M(k+1)+2)k}{2k-(k+1)(1+\varepsilon)T_\infty} = \mathcal{O}(Mk^2)$ *we can convert the strip*

packing for $\widetilde{\mathcal{R}}^{\ell}_{round} \cup \widetilde{\mathcal{R}}^{\ell}_{narrow} \cup Frac^{\ell}$ *into* $b(\tilde{m}_\ell, \infty)$ *into a 2-dimensional bin packing using at most* $2N_1$ *bins* $b(\tilde{m}_\ell, 1)$.

Proof. Using the above Lemma 9.5 for $h := (1+\varepsilon)N_1 T_\infty + 3Mk$ and $1/k$ we can convert the strip packing into $b(\tilde{m}_\ell, \infty)$ into a 2-dimensional bin packing using at most $(h+1)(1+1/k)$ bins $b(\tilde{m}_\ell, 1)$. To proof our claim we show that $(h+1)(1+1/k) \leq 2N_1$.

First we show that $k > \frac{(1+\varepsilon)T_\infty}{2-(1+\varepsilon)T_\infty}$: Since $\varepsilon \leq \frac{3}{18}$ we have

$$2 - (1+\varepsilon)T_\infty \geq 2 - (1+\varepsilon)1.7 \overset{\varepsilon \leq 3/18}{\geq} 1/60 > 0$$

and therefore $\frac{(1+\varepsilon)T_\infty}{2-(1+\varepsilon)T_\infty} \leq \frac{1.7(1+\frac{3}{18})}{2-1.7(1+\frac{3}{18})} = 119 \leq k$ since $k = \frac{20}{\varepsilon} \geq 120$. Furthermore we have

$$\begin{aligned} N_1 &= \frac{(3M(k+1)+2)k}{2k-(k+1)(1+\varepsilon)T_\infty} \\ &= \frac{C \cdot k^2 M}{k(2-(1+\varepsilon)T_\infty) - (1+\varepsilon)T_\infty} \text{ for a constant } C > 0 \\ &\leq \frac{C \cdot k^2 M}{k/60 - (1+3/18)1.7} \\ &\overset{k \geq 120}{\leq} C \cdot 60 \cdot k^2 M. \end{aligned}$$

And finally we can prove

$$\begin{aligned} (h+1)(1+1/k) &= \frac{(k+1)[(1+\varepsilon)N_1 T_\infty + 3Mk]}{k} + \frac{k+1}{k} \\ &\leq \frac{(k+1)(1+\varepsilon)N_1 T_\infty}{k} + \frac{k(3M(k+1)+2)}{k} \\ &\leq \frac{(k+1)(1+\varepsilon)N_1 T_\infty}{k} + \frac{N_1(2k-(k+1)(1+\varepsilon)T_\infty)}{k} \\ &\leq 2N_1. \end{aligned}$$

□

9.1.8 Converting Process

So far we constructed a schedule for the rounded platforms $\mathcal{B}_0 \cup \widetilde{\mathcal{B}}_1$. It remains to convert the schedule into one for $\mathcal{B}_0 \cup \mathcal{B}_1$.

Lemma 9.8. *The schedule can be converted into a schedule of length* $2 + \mathcal{O}(\varepsilon)$ *for* $\mathcal{B}_1 \cup \mathcal{B}_0$.

Proof. Remember from Section 9.1.6 that for $\mathcal{B}_0$ the schedule produced so far has length $\leq 1+5\varepsilon$ except for $\left\lceil \frac{|\mathcal{B}_0|\cdot 8\alpha}{\delta^4} \right\rceil$ platforms in which the schedule has length $\leq 2+5\varepsilon$. If it is possible to distribute the jobs scheduled in group $\widetilde{B}_1 \subseteq \widetilde{\mathcal{B}}_1$ among the platforms in $\mathcal{B}_0$ we can apply a shifting argument (see Figure 9.6) and obtain a schedule for $\mathcal{B}_0 \cup \mathcal{B}_1$. Recall that the

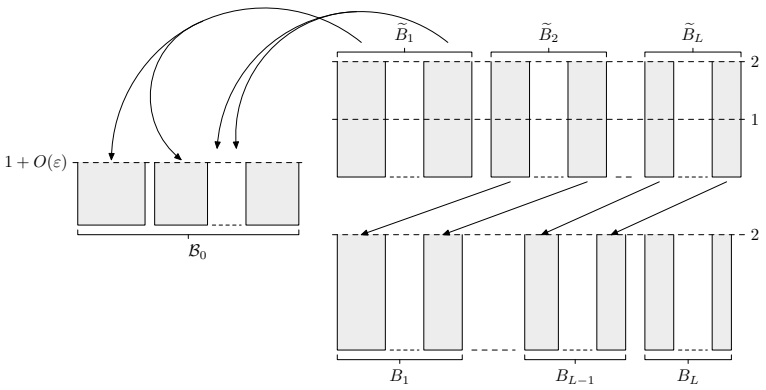

Figure 9.6: Shifting technique.

schedule in every platform in $\widetilde{B}_1$ is composed by stacking at most two 2-dimensional bins $b(\tilde{m}_1, 1)$ (where $\tilde{m}_1 \leq m_{|\mathcal{B}_0|}$) on top of each other. Thus, we need to distribute $2N_1$ bins $b(\tilde{m}_1, 1)$ among the platforms in $\mathcal{B}_0$.

In total, if $|\mathcal{B}_0|$ satisfies the inequality

$$|\mathcal{B}_0| \geq 2N_1 + \left\lceil \frac{|\mathcal{B}_0| \cdot 8\alpha}{\delta^4} \right\rceil \tag{9.15}$$

we can convert the schedule. Since $|\mathcal{B}_0| \geq N_0$ equation (9.15) holds, if

$$|\mathcal{B}_0|(1 - \frac{8\alpha}{\delta^4}) \geq 2N_1 + 1 \tag{9.16}$$

Since $\alpha = \delta^4/16$ this is fulfilled for $|\mathcal{B}_0| \geq N_0 = 2(2N_1+1)$.

□

9.2 Case 2: Using the Gap γ

We may now assume that there is a number $K \in [N]$, so that $m_1/m_K \leq \gamma$ and $m_1/m_{K+1} > \gamma$, where $\gamma = \frac{8}{3}N_1$.

If $K \geq N_0 = \mathcal{O}(1/\varepsilon^3 \log(1/\varepsilon))$ a variant of the algorithm for the first scenario can be applied achieving a $(2+\mathcal{O}(\varepsilon))$-approximation. In this

variant we partition the platforms in the same way as in Case 1 and consider jobs as wide if they satisfy $q_j \geq \lfloor \alpha m_{|\mathcal{B}_0|} \rfloor$ where $\alpha = \delta^4/16$ as before. The rest of the algorithm can be directly applied. Thus, throughout this section we assume $K < N_0$.

9.2.1 Structural Simplifications

We define $\mathcal{B}_0 := \{P_1, \ldots, P_K\}$ and $\mathcal{B}_1 := \{P_{K+1}, \ldots, P_N\}$. For $N_1 = \mathcal{O}(1/\varepsilon^3 \log(1/\varepsilon))$ as in Lemma 9.7 we partition $\mathcal{B}_1$ into $L := \left\lceil \frac{N-K}{N_1} \right\rceil$ groups

$$B_\ell = \{P_{K+(\ell-1)N_1+1}, \ldots, P_{K+\ell N_1}\} \text{ for } \ell \in [L-1]$$

containing exactly N_1 platforms and $B_L := \{P_{K+(L-1)N_1+1,\ldots,P_N}\}$ containing maybe less that N_1 platforms. In each group B_ℓ, $\ell \in [L]$ we round the number of processors of each platform up to the number of processors $\widetilde{m}_\ell := m_{K+(\ell-1)N_1+1}$ of the largest platform $P_{K+(\ell-1)N_1+1}$ contained in this group. In group B_L we add $N_1 - |B_L|$ dummy platforms with $\widetilde{m}_L$ processors, so that every modified group, denoted with $\widetilde{B}_\ell$, $\ell \in [L]$, contains exactly N_1 platforms of the same kind.

We first compute a schedule for $\mathcal{B}_0 \cup \widetilde{\mathcal{B}}_1$, where $\widetilde{\mathcal{B}}_1 = \bigcup_\ell \widetilde{B}_\ell$, and convert this solution into a solution for $\mathcal{B}_0 \cup \mathcal{B}_1$. With a similar argument as we assumed $m_N \geq 32/\delta^4$ in the first case we may assume here that $m_K \geq 32/\delta^4$. (Otherwise the number of processors in platform P_1 and therefore in every platform is bounded by a constant. This implies that also the number of jobs that fit next to each other is bounded by a constant and we do not distinguish between wide and narrow jobs to enumerate them.) We choose $\alpha = \delta^4/16$. Then we have $\alpha m_K \geq 2$ implying $\alpha m_K - 1 \geq \alpha m_K/2$ We call a job (p_j, q_j) *wide* if $q_j \geq \lfloor \alpha m_K \rfloor$ and *narrow* otherwise.

9.2.2 Algorithm for Case 2

As in the first case we enumerate and assignment of large wide for $\mathcal{B}_0$ and approximately guess the vector $\Pi^\star = (\Pi^\star_{i,a,h})$ of loads of the large narrow jobs in $\mathcal{B}_0$. We compute the free layers of height δ^2 in $\mathcal{B}_0$ and use the techniques as described in Sections 9.1.3-9.1.7.

It remains now to convert the schedule for $\mathcal{B}_0 \cup \widetilde{\mathcal{B}}_1$ into a schedule for $\mathcal{B}_0 \cup \mathcal{B}_1$. As in Case 1 we need to distribute the jobs scheduled in $\widetilde{B}_1$ and some extra large narrow jobs among the platforms in $\mathcal{B}_0$.

9.2.3 Converting Process and Choice of γ

In the worst case $K = 1$ and we have to place the additional $2N_1$ 2D-bins $b(\tilde{m}_1, 1)$ of width at most $m_{K+1} \geq \tilde{m}_1$ plus additional large narrow jobs of total width

$$K \cdot \frac{4\lfloor \alpha m_K \rfloor}{\delta^4} = \frac{4\lfloor \alpha m_1 \rfloor}{\delta^4}$$

next to each other on P_1. Thus, in the worst case the number of processors at least needed in P_1 can be bounded by

$$2N_1 \cdot m_{k+1} + \frac{4\lfloor \alpha m_1 \rfloor}{\delta^4} \overset{\alpha=\delta^4/16}{\leq} 2N_1 \cdot m_{K+1} + \frac{m_1}{4} \tag{9.17}$$

If we choose $\gamma = \frac{8}{3}N_1$ we have $m_1 > 2N_1 \cdot m_{K+1} + \frac{m_1}{4}$.

Finally, we obtained a $(2 + \mathcal{O}(\varepsilon))$-approximation in both cases and proved Theorem 9.1.

III Scheduling on Uniform Processors

10. Introduction

10.1 Definitions

One of the classical problems in optimization theory is Scheduling Jobs on Identical or Uniform Processors with the objective to minimize the makespan. Here we are given a set $\mathcal{J}$ of n jobs with processing times $p_j \in \mathbb{Q}_{\geq 0}$ and a set $\mathcal{P}$ of m processors P_i, each of them running with a certain speed $s_i \in \mathbb{Q}_{>0}$. A job j needs p_j/s_i time units to be finished, if it is executed on P_i. The goal is to find an assignment $a : \mathcal{J} \longrightarrow \mathcal{P}$ of the jobs to the processors, i.e. a schedule, minimizing the makespan $C_{\max} := \max_i C_i$, where $C_i := (1/s_i) \sum_{\{j|a(j)=P_i\}} p_j$. In Scheduling on Identical Processors ($P||C_{\max}$) all the processors run at the same speed. If the speed values may differ we call the problem Scheduling on Uniform Processors ($Q||C_{\max}$). Consequently, the problem $P||C_{\max}$ is a special case of $Q||C_{\max}$. Variants of both problems are $Pm||C_{\max}$ and $Qm||C_{\max}$, respectively. Here, we may additionally assume that the number of machines m is bounded by a constant.

10.2 Related Work

The problem $P2||C_{\max}$ is already NP-hard and $P||C_{max}$ is even known to be strongly NP-hard [22, 23]. In the 1970's Gonzales et al. [22] proposed the list algorithm LPT using longest processing time first policy that for an instance $\mathcal{I}$ of $Q||C_{\max}$ has output in $[(1.5\mathrm{OPT}(\mathcal{I}), 2\mathrm{OPT}(\mathcal{I})]$ and running-time $\mathcal{O}(nm\log(n))$ (w.l.o.g. $m \leq n$). Moreover, it was shown that $LPT(\mathcal{I}) \leq 4/3 - 1/(3m)$ for m identical processors and $LPT(\mathcal{I}) \leq 2 - \frac{2}{m+1}$ for m uniform processors [22].

Coffman et al. [13] presented the MULTIFIT algorithm for $P||C_{\max}$. It uses a bin packing subroutine with first fit decreasing policy (FFD)

in a binary search over possible schedule lengths and has running-time $\mathcal{O}(n \log^2(n))$. Yue [66] proved that MULTIFIT has exact ratio 13/11. Before, it was already known that the approximation ratio of MULTIFIT for $P||C_{\max}$ is not worse than 1.22 [13], 1.2 [19], respectively. For uniform processors there are variants of MULTIFIT with ratio 3/2 [46], 1.4 [20] and 1.38 [12].

In the 1980's Hochbaum and Shmoys found a PTAS for $Q||C_{\max}$ with running-time $(n/\varepsilon)^{\mathcal{O}(1/\varepsilon^2)}$ [27, 28]. For identical processors the complexity was improved to $(n/\varepsilon)^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon))}$ by Leung [52]. Earlier Horowitz and Sahni have already proved the existence of a PTAS for $Qm||C_{\max}$ [30].

Since $P||C_{\max}$ and $Q||C_{\max}$ are strongly NP-hard an FPTAS is ruled out, but Hochbaum [29] and Alon et al. [1] developed an EPTAS for $P||C_{\max}$ with running-time $f(1/\varepsilon) + \mathcal{O}(n)$, where f is doubly exponential in $1/\varepsilon$. In [33] Jansen gave an EPTAS for scheduling jobs on uniform processors using an MILP relaxation with running time $2^{\mathcal{O}(1/\varepsilon^2 \log^3(1/\varepsilon))} + poly(n)$. Sanders et al. [57] obtained a robust online algorithm for scheduling on identical machines with competitive ratio $(1+\varepsilon)$ and migration factor $\beta(\varepsilon) = 2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))}$ so that the running time for incorporating a newly arrived job is constant. It maintains and updates a data structure in time doubly exponential in $1/\varepsilon$, namely $2^{2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))}}$, in each iteration. This is done by comparing the distance between solutions for ILPs with different right hand sides. The general case for uniform processors is not considered.

10.3 New Results

We present an EPTAS for scheduling on uniform machines avoiding the use of an MILP or ILP solver. This result has been partly published as extended abstract in [38]. In our new approach instead of solving (M)ILPs we solve the LP-relaxation and use structural information about the "closest" ILP solution. For a given LP-solution x we consider the distance to the closest ILP solution y in the infinity norm, i.e. $\|x - y\|_\infty$. For the constraint

matrix $\widetilde{A}_\delta$ of the considered LP we call this distance

$$\max\text{-gap}(\widetilde{A}_\delta) := \max\{\min\{\|y^\star - x^\star\|_\infty : y^\star\text{solution of ILP}\} : x^\star\text{solution of LP}\}.$$

Let $C(\widetilde{A}_\delta)$ denote an upper bound for $\max\text{-gap}(\widetilde{A}_\delta)$. Our main theorem is the following.

Theorem 10.1. *There is an EPTAS for scheduling jobs on uniform machines with running time*

$$2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon) \log(C(\widetilde{A}_\delta)))} + poly(n) = 2^{\mathcal{O}(1/\varepsilon^2 \log^3(1/\varepsilon))} + poly(n).$$

If $C(\widetilde{A}_\delta) = poly(1/\varepsilon)$*, the running time improves to* $2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))} + poly(n)$.

Using a result of Cook et al. [15] we show that $C(\widetilde{A}_\delta) \leq 2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))}$. Consequently, our algorithm has running time at most $2^{\mathcal{O}(1/\varepsilon^2 \log^3(1/\varepsilon))} + poly(n)$, the same as in [33]. But, to our best knowledge, no instance is known to take on the value $2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))}$ for $\max\text{-}\,\text{gap}(\widetilde{A}_\delta)$. We conjecture $C(\widetilde{A}_\delta) \leq poly(1/\varepsilon)$. If that holds, the running time of the algorithm would be $2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))} + poly(n)$ and thus improve the result in [33].

In the next chapter we generally describe the method used. From this we derive an approach for identical machines. In Chapter 12 we describe the Algorithm for $Q||C_{\max}$. This chapter is mainly divided into three sections where in each we consider instances with different properties and give the algorithms for these instances.

11. Method Description

Assume that we are given an instance $\bar{\mathcal{I}}$ of m identical processors and n jobs with only d different processing times p_j, such that there are n_j jobs of each size. We use the dual approximation method by Hochbaum and Shmoys [28] to find a value T for the optimum makespan and transform the scheduling problem into a BIN PACKING problem with bin size T. Then the problem can be described via the following configuration ILP for d different item sizes:

$$\begin{aligned} \sum_i x_i &\leq m \\ \sum_i a(j,i)x_i &\geq n_j \text{ for } j = 1,\ldots,d \\ x_i &\in \mathbb{Z}_{\geq 0}. \end{aligned} \tag{11.1}$$

A configuration C_i is a multiset of processing times p_j so that their total sum is bounded by T. In (11.1) the variable x_i indicates the number of bins in which jobs are packed according to configuration C_i. The integer $a(j,i)$ denotes the number of jobs of processing time p_j in C_i.

Solving an ILP is always difficult [41, 49], so we consider the solution of its LP-relaxation of (11.1) and try to get information about the structure of a related ILP-solution.

For the constraint matrix $A := (a(j,i))_{ji}$ of the above (11.1) we consider

$$\begin{aligned} &\max\text{-gap}(A) := \\ &\max\{\min\{\|y^\star - x^\star\|_\infty : y^\star \text{solution of ILP (11.1)}\} : x^\star \text{solution of LP (11.1)}\}. \end{aligned}$$

Having an upper bound $C(A)$ for $\max\text{-gap}(A)$ and having an optimum fractional solution $x^\star$ we conclude that there exists an optimum solution $y^\star$ of (11.1) so that $y_i^\star \geq \lceil x_i^\star - C(A) \rceil$ for $x_i^\star \geq C(A)$. So we know how a

subset of the bins $\mathcal{B}' \subset \mathcal{B}$ has to be filled with jobs in the optimum solution $y^\star$. We can reduce the instance to an instance $\bar{\mathcal{I}}_{red}$ by taking out the bins in $\mathcal{B}'$ and those jobs that are packed in $\mathcal{B}'$:

$$\begin{aligned} \widetilde{m} &:= m - \sum_{x_i^\star > C(A)} \lceil x_i^\star - C(A) \rceil \text{ processors} \\ \widetilde{n}_j &:= n_j - \sum_{x_i^\star > C(A)} a(j,i) \lceil x_i^\star - C(A) \rceil \text{ for all processing times } p_j. \end{aligned} \tag{11.2}$$

In Figure 11.1 for example we have $C(A) = 3$. Given an optimum fractional solution $x^\star$ we conclude that there exists an optimum solution $y^\star$ of the ILP with $\|x^\star - y^\star\|_\infty \leq 3$. Thus, if $x_i^\star = 7.5$ we have $y_i^\star \geq 5$. Therefore we know that there is an integral solution of ILP (11.2) where at least 5 bins are occupied with configuration C_i. We take out these 5 bins and the corresponding jobs.

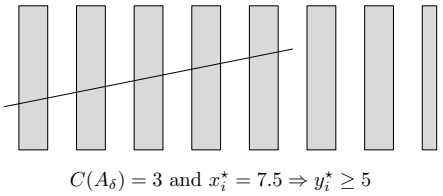

Figure 11.1: Reducing the instance.

Lemma 11.1. *If the number of different job sizes and the number of jobs per bin are both bounded by a constant, the total number of remaining jobs in $\bar{\mathcal{I}}_{red}$ can be bounded by a function in the value $C(A)$, namely*

$$\#(\textit{different job sizes}) \cdot \#(\textit{jobs per bin}) \cdot C(A).$$

Proof. Assume that the number of job sizes d is constant and that there can be at most k jobs placed in each bin. Therefore we have $\sum_{j=1}^{n} a(j,i) \leq k$ for every configuration C_i. Furthermore, in a basic solution of LP (11.1) the number of non-zero variables is at most $rank(A) \leq d$. W.l.o.g. let $x_1^\star, \ldots, x_d^\star$ be the non-zero variables. After the reduction for each non-zero configuration C_i there remain at most $a(j,i)(x_i^\star - \lceil x_i^\star - C(A) \rceil) \leq a(j,i)C(A)$ jobs of size j unassigned. So the total number of remaining

jobs in $\bar{\mathcal{I}}_{red}$ is bounded by

$$\sum_i \sum_{j=1}^{n} a(j,i)(x_i^\star - \lceil x_i^\star - C(A) \rceil) \leq \sum_{i=1}^{d} \sum_{j=1}^{n} a(j,i) C(A)$$
$$\leq \sum_{i=1}^{d} k \cdot C(A) = d \cdot k \cdot C(A).$$

□

Consequently, if we can bound the number of jobs sizes d from above, the size of the smallest job from below and obtain an upper bound $C(A)$ for max-gap(A) the number of remaining jobs in $\bar{\mathcal{I}}_{red}$ is also bounded by a constant. If this is the case we can use a dynamic programming approach as in [51] and compute an assignment of the remaining jobs in time $\mathcal{O}(|\bar{\mathcal{I}}_{red}|^d)$.

11.1 An Approach for Identical Processors

In the following we show how to modify a given instance $\mathcal{I}$ of $P||\mathrm{C}_{\max}$ and partition the jobs into large and small ones so that the technique described above can be applied to find an $(1+\varepsilon)$-approximate schedule for the large jobs. Small jobs will be added greedily in the end so that the resulting schedule has length at most $(1+\mathcal{O}(\varepsilon))\mathrm{OPT}(\mathcal{I})$. A summary of the approach is given in Algorithm 14.

11.1.1 Computing the Length of an Approximate Schedule

We first compute a near optimum value T for the makespan using the dual approximation method by Hochbaum and Shmoys [27].

Let $LPT(\mathcal{I})$ be the length of a schedule generated by the list scheduling algorithm with $LPT(\mathcal{I}) \leq 2\mathrm{OPT}(\mathcal{I})$ [22]. This implies that $LPT(\mathcal{I})/2 \leq OPT(\mathcal{I}) \leq LPT(\mathcal{I})$. For $\varepsilon > 0$ we choose a value $\delta \leq \min(\varepsilon/4, 1/2)$. Using binary search over the interval $[LPT(\mathcal{I})/2, LPT(\mathcal{I})]$ we compute a value T such that $OPT(\mathcal{I}) \leq T \leq (1+\delta)OPT(\mathcal{I})$. Notice that the interval $[LPT(\mathcal{I})/2, LPT(\mathcal{I})]$ can be divided into $1/\delta$ subintervals of length $(\delta/2)LPT(\mathcal{I}) \leq \delta OPT(\mathcal{I})$.
The dual approximation method either for a given value T computes an

approximate schedule of length at most $T(1+\delta)$ or shows that there is no schedule of length T.

Using binary search over the interval, we compute a value $T \leq (1+\delta)OPT(\mathcal{I})$ and an approximate schedule below of length at most $(1+2\delta)T \leq (1+2\delta)(1+\delta)OPT(\mathcal{I}) \leq (1+4\delta)OPT(\mathcal{I}) \leq (1+\varepsilon)OPT(\mathcal{I})$ for $\delta \leq 1/2$ and $\delta \leq \varepsilon/4$. Notice that $\mathcal{O}(\log(1/\varepsilon))$ steps are sufficient in the binary search.

Algorithm 14 Algorithm for identical machines

Input: Instance $\mathcal{I}$ of $P||C_{max}$, $\varepsilon > 0$
Output: A schedule of length $(1+\varepsilon)\text{OPT}(\mathcal{I})$

1: Obtain with list scheduling algorithm a schedule of length $LS(\mathcal{I}) \geq 2\text{OPT}(\mathcal{I})$.
2: **for** a candidate value for the makespan $T \in [LS(\mathcal{I})/2, LS(\mathcal{I})]$ **do**
3: Transform the problem into an instance of Bin Packing.
4: **if** the total size of all jobs is not bounded by Tm or $\max \bar{p}_j > T$ **then**
5: increase T and go to Step 3.
6: **end if**
7: Round the processing times of the jobs and obtain $\bar{\mathcal{I}}$.
8: Distinguish small and large jobs.
9: Compute a solution of LP (11.1) that allocates large jobs.
10: **if** LP (11.1) has a feasible solution **then**
11: Find a schedule for a subset of the large jobs and reduce the instance to $\bar{\mathcal{I}}_{red}$.
12: **else**
13: increase T and go to Step 3.
14: **end if**
15: Compute an assignment of large jobs in $\bar{\mathcal{I}}_{red}$ using dynamic programming.
16: **if** the dynamic program for large jobs in $\bar{\mathcal{I}}_{red}$ does find a feasible solution for m machines **then**
17: increase T and go to Step 3.
18: **end if**
19: Greedily add the small jobs.
20: **end for**

11.1.2 Rounding the Jobs

As described before, the scheduling problem can be transformed into a Bin Packing problem with bin capacities equal to T [13]. By scaling we may assume that $T = 1$ and round the processing times p_j to values $\bar{p}_j =$

$\delta(1+\delta)^{k_j}$ with $k_j \in \mathbb{Z}$ such that $p_j \leq \bar{p}_j \leq (1+\delta)p_j$. Let $\bar{\mathcal{I}}$ be the instance with the rounded processing times. Clearly, for any set A of jobs with $\sum_{j\in A} p_j \leq 1$ we have for the total rounded processing time $\sum_{j\in A} \bar{p}_j \leq (1+\delta)$. Consequently, we have to enlarge the bin capacities slightly to $(1+\delta)T = (1+\delta)$. We partition the jobs now into large and small jobs. A job j is *large*, if $\bar{p}_j \geq \delta$. Other jobs with $\bar{p}_j < \delta$ are called *small*. Note that the number of large jobs per bin is bounded by $(1+\delta)/\delta = \mathcal{O}(1/\delta)$. With the following Lemma 11.2 we can also bound the number of different large job sizes in $\mathcal{I}$.

Lemma 11.2. *Let A be a set $\{a(1+\delta)^x, \ldots, a(1+\delta)^y\}$ with $x, y \in \mathbb{Z}^+$, $x < y$ and $a \in \mathbb{R}^+$. Then $|A| \geq \log(\max(A)/\min(A))/\delta + 1$ and $|A| \leq 2\log(\max(A)/\min(A))/\delta + 1$ for any $\delta \in (0, 1/2]$.*

Proof. Using the assumption on A, $\max(A)/\min(A) = (1+\delta)^{y-x}$. Therefore, the number of elements $a(1+\delta)^i$ in A is equal to

$$y - x + 1 = \log(\max(A)/\min(A))/\log(1+\delta) + 1.$$

Using $\log(1+\delta) \geq \delta - \delta^2 \geq \delta/2$ for $\delta \in (0, 1/2]$ and $\log(1+\delta) \leq \delta$, the cardinality of A is at least $\log(\max(A)/\min(A))/\delta + 1$ and at most $2\log(\max(A)/\min(A))/\delta + 1$. □

We conclude that the number of different large rounded processing times is bounded by

$$d := 2\log((1+\delta)/\delta)/\delta + 1 = \mathcal{O}(1/\delta \log(1/\delta)).$$

With n_j we denote the number of jobs in $\bar{\mathcal{I}}$ with processing time $\delta(1+\delta)^j$ for $j = 0, \ldots, d-1$.

11.1.3 (I)LP-formulation for Large Jobs

We formulate the assignment of the large jobs to the bins as a configuration ILP as in (11.1) with an exponential number of variables. Here, a configuration is a multiset of large rounded processing times, i.e. values $\delta(1+\delta)^j \in [\delta, (1+\delta)]$ so that their total sum is bounded by $1+\delta$. Since there are at most $(1+\delta)/\delta$ many large jobs on a processor with makespan $(1+\delta)$, the number of multisets with large processing times is bounded

by

$$((1+\delta)/\delta)^d = ((1+\delta)/\delta)^{\mathcal{O}(1/\delta \log(1/\delta))} = 2^{\mathcal{O}(1/\delta \log^2(1/\delta))}.$$

Let $a(j,i)$ be the number of occurrences of value $\delta(1+\delta)^j$ in C_i and let the matrix $A_\delta := (a(j,i))_{ji}$ denote the constraint matrix of the ILP (11.1). Notice that $a(j,i) \leq (1+\delta)/\delta$ holds.

To avoid now the algorithm by Lenstra or Kannan to compute an optimum solution of (11.1), we consider its LP-relaxation. We can solve the LP using Theorem (6.6.5) in [53]:

Theorem 11.3. *[53] There exists algorithm that, for any well-described polyhedron $(P; n, \varphi)$ specified by a strong separation oracle, and for any given vector $c \in \mathbb{Q}^n$,*

1. *solve the strong optimization problem* $\max\{c^t x | x \in P\}$, *and*
2. *find an optimum vertex solution of* $\max\{c^t x | x \in P\}$, *and*
3. *find a basic optimum standard dual solution if one exists.*

The number of calls on the separation oracle, and the number of elementary arithmetic operations executed by the algorithms are bounded by a polynomial in φ. All arithmetic operations are performed on numbers whose encoding length is bounded by a polynomial in $\varphi + < c >$.

The first constraint of (11.1) can be transformed into an objective function $\min cx$ with $c \equiv 1$. Then the dual of the relaxed (11.1) is a maximization problem of the form $\max\left\{\sum_{j=0}^{d-1} n_j y_j |\ y \in P\right\}$, where

$$P = \left\{ y \in \mathbb{R}^d | \sum_j a(j,i) y_j \leq 1,\ 1 \leq i \leq C,\ y \geq 0 \right\}.$$

The polyhedron P has facet complexity at most

$$\varphi := d(\lceil \log(\frac{1+\delta}{\delta}) \rceil + 1) + 2 = \mathcal{O}(1/\delta \log^2(1/\delta))$$

since each inequality describing P has encoding length at most φ. So the triple $(P; d, \varphi)$ is a well described polyhedron in the sense of [53]. The strong separation problem for P can be solved in polynomial time by checking wether a given vector $y \in \mathbb{Q}^d$ satisfies the inequalities of P or

not. According to Theorem 11.3 there is an algorithm that finds a basic optimum solution of the dual of the dual, i.e. the relaxed ILP (11.1). The running time is bounded by a polynomial in φ and in the encoding length of $(n_0, \ldots, n_{d-1})$, which is in $\mathcal{O}(1/\delta \log(1/\delta) \log(n))$. Thus, the running time is bounded by $poly(1/\delta, \log(n))$.

11.1.4 Reducing the Instance of Large Jobs

Let $x^\star$ be an optimum basic LP-solution of (11.1) and let $C(A_\delta)$ be an upper bound for max-gap(A_δ). Let $\bar{\mathcal{I}}_{red}$ be the corresponding reduced instance achieved as in (11.2). From $x^\star$ we obtain a packing of the large rectangles in $\bar{\mathcal{I}} \backslash \bar{\mathcal{I}}_{red}$ as described above. There are at most $d = \mathcal{O}(1/\delta \log(1/\delta))$ different rounded large processing times and in each bin at most $(1+\delta)/\delta = \mathcal{O}(1/\delta)$ large jobs can be placed. According Lemma 11.1 the number $\widetilde{n}$ of large jobs $\bar{\mathcal{I}}_{red}$ is bounded by

$$\widetilde{n} \leq d \cdot \frac{1+\delta}{\delta} \cdot C(A_\delta) = \mathcal{O}(1/\delta^2 \log(1/\delta) C(A_\delta)).$$

Furthermore, we may suppose that $\widetilde{m} \leq \widetilde{n}$; since more processors are not necessary (in the end adding small jobs only slightly increases the makespan).

11.1.5 Dynamic Program for Large Jobs in Reduced Instance.

According to [51] an assignment of the remaining large jobs in $\bar{\mathcal{I}}_{red}$ can be computed via dynamic programming in time

$$\mathcal{O}(\widetilde{n}^d) = \widetilde{n}^{\mathcal{O}(1/\delta \log(1/\delta))} \leq 2^{\mathcal{O}(1/\delta \log(1/\delta) \log(C(A_\delta)/\delta))}.$$

We simply go over the machines and store vectors $(x_1, \ldots, x_d)$ with numbers x_j of jobs with processing time $\bar{p}_j$ for $j = 1, \ldots, d$ used by the first k processors for $k = 1, \ldots, \widetilde{m}$. If the algorithm does not find a feasible assignment, we know that (11.1) has no feasible solution and have to increase T.

11.1.6 Small Jobs

Once we have an assignment of the large jobs, we can pack the small jobs. They can be added greedily onto the processors, if the total size of the

small and large jobs is bounded by m (the total sum of all bin capacities). Adding the small jobs increases the makespan by at most δ. Therefore, the overall makespan is at most $(1+2\delta)$. If the condition above does not hold or if $p_{max} = \max_{j=1,\dots,n} p_j > 1$, there is no schedule of length $T = 1$.

11.2 Bounds for $\max$-gap(A_δ) and the Running time

In the following we analyze the running time of the approach given by Algorithm 14. To obtain an upper bound $C(A_\delta)$ for $\max$-gap(A_δ) we use an interesting result by Cook et al. [15]. They proved that the maximum distance between an optimum solution of the LP and a closest optimum solution of the ILP (and vice versa) is bounded by a function in the dimension and the coefficients of the underlying matrix.

Theorem 11.4. *[15] Let A be an integral $(M \times N)$ matrix, such that each subdeterminant is at most Δ in absolute value, and let b and c be vectors. Suppose that both objective values*
(i) $\min\{c^T x | Ax \geq b\}$ and (ii) $\min\{c^T x | Ax \geq b;\ x \in \mathbb{Z}^N\}$ are finite. Then:

(a) for each optimum solution y of (i) there exists an optimum solution z of (ii) with $\|y - z\|_\infty \leq N\Delta$ and

(b) for each optimum solution z of (ii) there exists an optimum solution y of (i) with $\|y - z\|_\infty \leq N\Delta$.

Note that the theorem above also holds, if we have additional inequalities of the form $x_i \geq 0$. Furthermore, we can use $c^T x = \sum_i x_i$ as objective function instead of the inequality $\sum_i x_i \leq m$ in (11.1). For scheduling on identical processors the objective values of the ILP formulation for the rounded large jobs $\bar{\mathcal{I}}$ given in Section 11.1.3 and its LP relaxation both are finite. Consequently, $\max$-gap(A_δ) is bounded by $N\Delta$.

In the following we give bounds for the parameters N and Δ.

Lemma 11.5. *The number of variables N in the modified ILP* (11.1)*, the maximum absolute value Δ over all subdeterminants corresponding to the matrix A_δ and $\max$-gap(A_δ) are at most $2^{\mathcal{O}(1/\delta \log^2(1/\delta))}$.*

Proof. The number N of variables in the (modified) ILP (11.1) is equal to the number of configurations, so it is bounded by $2^{\mathcal{O}(1/\delta \log^2(1/\delta))}$. On the

other hand, the absolute value of the determinant of a quadratic $(M \times M)$ sub-matrix A of A_δ with column vectors $A_1, \ldots, A_M$ and $|\det(A)| = \Delta$ can be bounded using the Hadamard's inequality

$$|\det(A)| \leq \prod_{i=1}^{M} \|A_i\|_2. \tag{11.3}$$

Since the coefficients of a column A_i correspond to a configuration C_i, the sum of the entries is bounded by $(1+\delta)/\delta$. Therefore, using the inequality $\|A_i\|_2 \leq \|A_i\|_1$, we obtain

$$\|A_i\|_2 \leq \|A_i\|_1 \leq \sum_{j=1}^{d} a(j,i) \leq (1+\delta)/\delta. \tag{11.4}$$

This implies that Δ is at most

$$\Delta \overset{(11.3)}{\leq} \prod_{i=1}^{d} \|A_i\|_2 \overset{(11.4)}{\leq} \prod_{i=1}^{d} \frac{(1+\delta)}{\delta} = \left(\frac{(1+\delta)}{\delta}\right)^d = 2^{O(1/\delta \log^2(1/\delta))}. \tag{11.5}$$

We conclude that max-gap(A_δ) is also at most $2^{\mathcal{O}(1/\delta \log^2(1/\delta))}$. □

This implies the following theorem

Theorem 11.6. *For a list of n jobs $\mathcal{I}$ Algorithm 14 produces a schedule on m machines with makespan at most $(1+\varepsilon)\mathrm{OPT}(\mathcal{I})$ in time*

$$2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon) \log(C(A_\delta)))} + \mathcal{O}(n \log n) \leq 2^{\mathcal{O}(1/\varepsilon^2 \log^3(1/\varepsilon))} + \mathcal{O}(n \log n).$$

If $C(A_\delta) = poly(1/\varepsilon)$, the running time improves to

$$2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))} + \mathcal{O}(n \log n).$$

Proof. The running time is composed by operations as follows: "sorting the items by size"+"binary search on T"·"solving the LP"·"dynamic program" +"adding small jobs". This gives total running time in

$$\begin{aligned} &\mathcal{O}(n \log n) + \mathcal{O}(\log(1/\varepsilon)) poly(1/\varepsilon, \log n) 2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon) \log(C(A_\delta)))} + \mathcal{O}(n) \\ &\leq \mathcal{O}(n \log n) + poly(\log n) 2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon) \log(C(A_\delta)))} \\ &\leq 2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon) \log(C(A_\delta)))} + \mathcal{O}(n \log n). \end{aligned}$$

□

On the other hand, the value Δ can be quite large. But note that a lower bound for max-gap(A_δ) remains unknown.

Lemma 11.7. *The maximum value Δ over all subdeterminants of the coefficient matrix $A_\delta = (a(j,i))$ is at least $2^{\Omega(1/\delta \log^2(1/\delta))}$.*

Proof. The number of rounded processing times $\bar{p}_j$ in the interval $[\delta, \delta^{1/2}]$ is at least $\log(\delta^{1/2}/\delta)/\delta = 1/\delta \log(\delta^{-1/2})$. For each rounded processing time $\bar{p}_j = \delta(1+\delta)^j \in [\delta, \delta^{1/2}]$ for $j = 0, \ldots, 1/\delta \log(\delta^{-1/2}) - 1$, we take one configuration or multiset C_j which consists of only $\delta^{-1/2}$ numbers $\bar{p}_j$, i.e.

$$C_j = \{0 : \delta(1+\delta)^0, \ldots, 0 : \delta(1+\delta)^{j-1}, \delta^{-1/2} : \delta(1+\delta)^j, \\ 0 : \delta(1+\delta)^{j+1}, \ldots, 0 : \delta(1+\delta)^{1/\delta \log(\delta^{-1/2})-1}\}.$$

The determinant of the matrix corresponding to these configurations $C_0, \ldots, C_{1/\delta \log(1/\delta^{1/2})-1}$ is

$$(\delta^{-1/2})^{\Omega(1/\delta \log(\delta^{-1/2}))} = 2^{\Omega(1/\delta \log^2(\delta^{-1/2}))} = 2^{\Omega(1/\delta \log^2(1/\delta))}.$$

This implies that $\Delta \geq 2^{\Omega(1/\delta \log^2(1/\delta))}$. □

12. An EPTAS for Scheduling on Uniform Processors without solving MILPs

For uniform processors we can compute a 2 - approximation using the LPT algorithm studied by Gonzales et al. [22]. Here $LPT(\mathcal{I}) \leq 2OPT(\mathcal{I})$ where $LPT(\mathcal{I})$ is the schedule length generated by the LPT algorithm. Similar to the approach for identical processors, we can split the interval $[LPT(\mathcal{I})/2, LPT(\mathcal{I})]$ into $1/\delta$ subintervals of length $(\delta/2)LPT(\mathcal{I}) \leq \delta OPT(\mathcal{I})$ and transform the scheduling problem with makespan T into a bin packing problem with bin sizes $c_1 \geq \ldots \geq c_m$ where $c_i = T \cdot s_i$. By scaling we assume $c_m = 1$. As for identical machines we round the job sizes p_j to values $\bar{p}_j = \delta(1+\delta)^{k_j} \leq (1+\delta)p_j$. Additionally we round and increase slightly the bin capacities c_i to values $\bar{c}_i = (1+\delta)^{\ell_i} \leq c_i(1+\delta)^2$. Let the instance of rounded jobs and bin capacities be denoted with $\bar{\mathcal{I}}$. For a set of bins $\mathcal{B}$ let $c_{\min}(\mathcal{B}) := \min\{c_i | b_i \in \mathcal{B}\}$. Analogously we define $c_{\max}(\mathcal{B})$.

Lemma 12.1. *[33] If there is a feasible packing of n jobs with processing times p_j into m bins with capacities $c_1, \ldots, c_m$, then there is also a packing of the n jobs with rounded processing times $\bar{p}_j = \delta(1+\delta)^{k_j} \leq (1+\delta)p_j$ into the m bins with rounded bin capacities $\bar{c}_i = (1+\delta)^{\ell_i} \leq c_i(1+\delta)^2$.*

Let $g : \mathbb{N} \longrightarrow \mathbb{N}$ be a function so that

$$g(1/\delta) \geq 1/\delta \log^2(1/\delta) \text{ with } g = poly(1/\delta).$$

Furthermore, define constants $G := \frac{g(1/\delta)}{\delta}$,

$$D := 2/\delta \log(g(1/\delta)/\delta) + 1 \text{ and } L := 2/\delta \log(g(1/\delta)) + 1$$

Note that $D, L = \mathcal{O}(1/\delta \log(1/\delta))$ and $G = \mathcal{O}(1/\delta)$.

Lemma 12.2. *Let $\bar{\mathcal{I}}$ be a rounded instance of $Q||C_{\max}$ as above with $\frac{\bar{c}_1}{\bar{c}_m} \leq g(1/\delta)$. Then the following holds:*

1. *The number of different rounded bin sizes in $\bar{\mathcal{I}}$ is bounded by L.*
2. *The number of different rounded job sizes with $\bar{p}_j \geq \delta \bar{c}_m$ is bounded by D.*
3. *The number of jobs with $\bar{p}_j \geq \delta \bar{c}_m$ that can be placed into any bin is bounded by G*

Proof. Using Lemma 11.2, the number of different rounded bin sizes is at most

$$2/\delta \log(\bar{c}_1/\bar{c}_m) + 1 = 2/\delta \log(g(1/\delta)) + 1 = L$$

If the set of jobs with large processing times $\bar{p}_j \geq \delta \bar{c}_m$ is not empty, we define $\bar{p}_{\min} := \min\{p_j | p_j \geq \delta, j = 1, \ldots, n\}$. Clearly, we have $\bar{p}_{\min} \geq \delta$. Since $\bar{p}_{\max} = \max_{j=1,\ldots,n} \bar{p}_j \leq \bar{c}_1 \leq g(1/\delta)$, we obtain $\bar{p}_{\max}/\bar{p}_{\min} \leq g(1/\delta)/\delta$. Again using Lemma 11.2, the total number of different rounded large processing times can be bounded by

$$2/\delta \log(\bar{p}_{\max}/\bar{p}_{\min}) + 1 = 2/\delta \log(g(1/\delta)/\delta) + 1 = D.$$

The number of jobs with processing times $\geq \delta c_m$ that can be placed into a single bin is bounded by $\frac{\bar{c}_1}{\delta \bar{c}_m} \leq \frac{g(1/\delta)}{\delta} = G$. □

Furthermore, define $f : \mathbb{N} \longrightarrow \mathbb{N}$ so that $f(1/\delta) = \frac{3 \cdot D \cdot C(\widetilde{A}_\delta) \cdot G}{\delta^2}$. Here, $C(\widetilde{A}_\delta)$ is an upper bound for max-gap$(\widetilde{A}_\delta)$, where $\widetilde{A}_\delta$ is a matrix corresponding to a more general scheduling problem given by Definition 12.3 with L rows (different job sizes) and at most $L \cdot G^D = 2^{\mathcal{O}(1/\delta \log^2(1/\delta))}$ columns (configurations) with integral entries in $\{0, 1, \ldots, \lfloor G \rfloor\}$ and column sums bounded by $G + 1$, similar to the constraint matrix of the configuration ILP used for identical processors. The constant $C(\widetilde{A}_\delta)$ will be precised by Definition 12.13 at the beginning of Section 12.3.

We distinguish between three different scenarios depending on the structure of the set of bins in the instance.

Case 1: For all $i \in [m]$ we have $\bar{c}_1/\bar{c}_m \leq g(1/\delta)$.

Case 2: There exists an index $K < f(1/\delta)$ with

$$\bar{c}_1/\bar{c}_i < g(1/\delta) \text{ for } 1 \leq i \leq K \text{ and } \bar{c}_1/\bar{c}_i \geq g(1/\delta) \text{ for } K+1 \leq i \leq m.$$

Case 3: There exists an index $K \geq f(1/\delta)$ with
$\bar{c_1}/\bar{c_i} < g(1/\delta)$ for $1 \leq i \leq K$ and $\bar{c_1}/\bar{c_i} \geq g(1/\delta)$ for $K+1 \leq i \leq m$.

In the first scenario all bins have similar capacities. We have D different rounded jobs sizes for large jobs with $\bar{p}_j \geq \delta$ and a constant number L of different rounded bin sizes where each bin can hold at most G large jobs. Then, the scenario can be solved very similar to the problem with identical machines.

In the second scenario the quotient $\frac{c_1}{c_m}$ can be arbitrary large and for a lower bound on the large job sizes we cannot bound the number of different job sizes for jobs with $\bar{p}_j \geq \delta$ globally as before. Here, we consider two different bin groups

$$\mathcal{B}_0 := \{b_1, \ldots, b_K\} \text{ and } \mathcal{B}_1 := \{b_{K+1}, \ldots, b_m\}.$$

Then in $\mathcal{B}_0$ the number of bins is bounded and as $g(1/\delta) \geq \frac{c_1}{c_K}$ the number of different bin sizes is bounded by L. As in $\bar{\mathcal{I}}$ the number of different rounded job sizes of jobs with $\bar{p}_j \geq \delta c_K$ is bounded by D we can preprocess an assignment v of those jobs for $\mathcal{B}_0$ using dynamic programming. For a feasible assignment v we allocate the remaining jobs, except for tiny jobs with processing time $\leq \delta$, fractionally into $\mathcal{B}_1$ via a linear program. Then we round the solution of the LP with a novel rounding technique using a subroutine for Bin Packing with Different Bin Sizes that produces only few extra bins. The jobs in the extra bins can be distributed using among the bins in $\mathcal{B}_0$ using the gap $g(1/\delta)$. In the end the small jobs are scheduled behind the large ones. An overview gives Algorithm 15.

The third scenario is the most complicated case. Here we have three bin groups

$$\mathcal{B}_0 = \{b_1, \ldots, b_K\},\ \mathcal{B}_1 = \{b_i | i > K, \bar{c}_i \geq \delta c_{\min}(\mathcal{B}_0)\} \text{ and } \mathcal{B}_2 = \mathcal{B} \setminus (\mathcal{B}_0 \cup \mathcal{B}_1).$$

If $\mathcal{B}_1 \neq \emptyset$ we distinguish large, medium and small jobs, else we only have large and small jobs: A job is called *large* if $\bar{p}_j > \delta c_{\min}(\mathcal{B}_0)$ and *medium* if $\bar{p}_j \in (\delta c_{\min}(\mathcal{B}_1), \delta c_{\min}(\mathcal{B}_0)]$; other jobs are called *small*. We use an LP to

obtain a fractional assignment of all jobs with $\bar{p}_j < \delta$. From this solution we extract the part that represents the assignment of the large jobs to $\mathcal{B}_0 \cup \mathcal{B}_1$. This assignment describes a scenario where jobs with D different sizes have to be placed into bins having a constant number of different bin sizes so that each bin can hold at most G large jobs. Using a similar approach as in the first case and as for identical machines, we can allocate a subset of the large jobs into $\mathcal{B}_0$ and $\mathcal{B}_1$ and reduce the instance to an instance $\bar{\mathcal{I}}_{red}$ so that the number of remaining large jobs is bounded by a constant. Via dynamic programming we obtain an assignment of the remaining large jobs to $\mathcal{B}_0 \cup \mathcal{B}_1$. Here, it may happen that we produce some error. The medium (and some small jobs) jobs are packed using the same rounding technique for the LP solution with underlying subroutine for Bin Packing with Different Bin Sizes as in the second case. Caused by the error of the dynamic program we might have large jobs that block some area that is actually designated to hold medium jobs by the LP. Therefore we have to do some rearrangement of the affected medium jobs. Finally we add the jobs with processing times $< \delta$. An overview gives Algorithm 16.

12.1 Algorithm for Case 1

In this scenario a job is called *large*, if $\bar{p}_j = \delta(1+\delta)^{k_j} \geq \delta \bar{c}_m = \delta$, and *small* otherwise. According to Lemma 12.2 the number of different rounded bin sizes in $\bar{\mathcal{I}}$ is at most L, the number of different rounded large processing times in $\bar{\mathcal{I}}$ is at most D, and the number of large jobs that can be placed into a bin is at most G.

Let n_j be the number of jobs with processing time $\delta(1+\delta)^j$ for $j = 0, \ldots, D-1$ and let m_ℓ be the number of bins with capacity or bin size $\bar{c}_\ell = (1+\delta)^{\ell-1}$ for $\ell = 1, \ldots, L$. All bins with capacity $(1+\delta)^{\ell-1}$ form a block B_ℓ of identical bins. The overall algorithm works very similar to the one in the identical case, so we only describe the main part for the large jobs.

12.1.1 Allocating Large Jobs

For the assignment of the large jobs to the bins, we set up an integer linear program, which is block by block similar as (11.1) for identical machines. Here we introduce for each bin size $\ell \in \{1, \ldots, L\}$ configurations $C_i^\ell \in \mathcal{C}^\ell$

as multisets of numbers $\delta(1+\delta)^j \in [\delta, \bar{c}_\ell]$, where the total sum is bounded by $\bar{c}_\ell$ for $\ell = 1, \ldots, L$. Furthermore, let $a(j, i^{(\ell)})$ be the number of occurrences of processing time $\delta(1+\delta)^j$ in configuration $C_i^{(\ell)}$. In the ILP below we use an integral variable $x_i^{(\ell)}$ to indicate the number of configurations $C_i^{(\ell)}$ in block B_ℓ.

$$\begin{aligned} \sum_i x_i^{(\ell)} &\leq m_\ell \text{ for } \quad \ell = 1, \ldots, L \\ \sum_{i,\ell} a(j, i^{(\ell)}) x_i^{(\ell)} &\geq n_j \text{ for } \quad j = 0, \ldots, D-1 \\ x_i^{(\ell)} &\in \mathbb{Z}_{\geq 0} \text{ for } \quad i = 1, \ldots, |\mathcal{C}^{(\ell)}|, \ell = 1, \ldots, L \end{aligned} \tag{12.1}$$

The number of constraints in (12.1) (not counting the non-negativity constraints $x_i^{(\ell)} \geq 0$) is at most $D + L \leq \mathcal{O}(1/\delta \log(1/\delta))$. The number of variables is bounded by $L \cdot G^D \leq 2^{\mathcal{O}(1/\delta \log^2(1/\delta))}$ To apply Theorem 11.4, we multiply the first L inequalities of (12.1) by (-1) and obtain $\sum_i (-1) x_i^{(\ell)} \geq -m_\ell$ for $\ell = 1, \ldots, L$.

Definition 12.3. Let $\widetilde{A}_\delta$ be any constraint matrix corresponding to a configuration ILP that describes an assignment of jobs with D different job sizes to bins of L different sizes so that each bin can hold at most G of such jobs. Then $\widetilde{A}_\delta$ consists of at most $L \cdot G^D$ column vectors of the form

$$A_i^{(\ell)} = (0, \ldots, 0, -1, 0, \ldots, 0, a(1, i^{(\ell)}), \ldots, a(D, i^{(\ell)}))^T. \tag{12.2}$$

with $\sum_{j=1}^{D} a(1, i^{(\ell)}) \leq G$ for each column $A_i^{(\ell)}$.

Let $C(\widetilde{A}_\delta)$ to be an upper bound for $\max\text{-gap}(\widetilde{A}_\delta)$ for all constraint matrices describing the same scenario as $\widetilde{A}_\delta$. Note that the constraint matrix of (12.1) is such a matrix $\widetilde{A}_\delta$ and recall that

$$\begin{aligned} &\max\text{-gap}(\widetilde{A}_\delta) = \\ &\max\{\min\{\|y^\star - x^\star\|_\infty : y^\star \text{solution of ILP (12.1)}\} : x^\star \text{solution of LP (12.1)}\}. \end{aligned}$$

Suppose that there exists a feasible solution of (12.1), then there is one for the LP relaxation, too. Our algorithm first solves the LP relaxation similar as for identical machines: The polyhedron corresponding to the dual has facet complexity bounded by $\mathcal{O}(1/\delta \log^2(1/\delta))$ and the price vector of the dual has encoding length at most $\mathcal{O}(1/\delta \log(1/\delta) \log(n))$. Thus,

according to Theorem 11.3 we find a basic optimum solution $(x_i^{(\ell)})^\star$ of the LP relaxation in time $poly(1/\delta, \log(n))$.

Let $(y_i^{(\ell)})^\star$ be a feasible solution of the ILP (12.1) with distance (in the maximum norm) bounded by $C(\widetilde{A}_\delta)$, i.e. $|y_i^{(\ell)\star} - x_i^{(\ell)\star}| \leq C(\widetilde{A}_\delta)$ for all i. Such a solution $y^\star$ exists according to Theorem 11.4 since both, LP and ILP, are feasible. If $x_i^{\ell\star} > C(\widetilde{A}_\delta)$ we set $y_i^{(\ell)\star} = \lceil x_i^{(\ell)\star} - C(\widetilde{A}_\delta) \rceil$ and use $y_i^{(\ell)\star}$ configurations of type $C_i^{(\ell)}$ for block B_ℓ. Now we can reduce the instance as follows:

(12.3)

$$\widetilde{m_\ell} := m_\ell - \sum_{i: x_i^{(\ell)\star} > C(\widetilde{A}_\delta)} \lceil x_i^{(\ell)\star} - C(\widetilde{A}_\delta) \rceil \text{ bins in block } B_\ell \text{ for } \ell = 1, \ldots, L$$

$$\widetilde{n}_j := n_j - \sum_{i,\ell: x_i^{(\ell)\star} > C(\widetilde{A}_\delta)} a(j, i^{(\ell)}) \lceil x_i^{(\ell)\star} - C(\widetilde{A}_\delta) \rceil$$

jobs with rounded processing times $\delta(1+\delta)^j$ for $j = 0, \ldots, D-1$.

Let $\bar{\mathcal{I}}_{red}$ be the reduced instance. According to Lemma 12.2 the number of different large job sizes is bounded by D. Furthermore, in each bin at most G large jobs can be placed. According to Lemma 11.1 the number $\widetilde{n}$ of large jobs in $\bar{\mathcal{I}}_{red}$ can be bounded by

$$\widetilde{n} \leq D \cdot C(\widetilde{A}_\delta) \cdot G. \tag{12.4}$$

Since $G = \mathcal{O}(1/\delta)$ and $D = \mathcal{O}(1/\delta \log(1/\delta))$ and assuming $C(\widetilde{A}_\delta) \geq D, G$ an assignment of the large jobs $\bar{\mathcal{I}}_{red}$ in can be computed via a dynamic program in time

(12.5)

$$\widetilde{n}^{\mathcal{O}(1/\delta \log(1/\delta))} \leq 2^{\mathcal{O}(1/\delta \log(1/\delta) \log\left(D \cdot C(\widetilde{A}_\delta) \cdot G\right))} = 2^{\mathcal{O}(1/\delta \log(1/\delta) \log(C(\widetilde{A}_\delta)))}.$$

If the dynamic program does not find a feasible assignment, then we know that the ILP (12.1) has no feasible solution.

12.1.2 Upper Bound for $\max$-gap$(\widetilde{A}_\delta)$.

To achieve an upper bound for $\max$-gap$(\widetilde{A}_\delta)$ we use again Theorem 11.4.

Lemma 12.4. *The maximum absolute value Δ of a subdeterminant of matrix $\widetilde{A}_\delta$ and* max*-gap$(\widetilde{A}_\delta)$ are bounded by $2^{\mathcal{O}(1/\delta \log^2(1/\delta))}$. In particular,* max*-gap$(\widetilde{A}_\delta)$ is bounded by $L \cdot (G^2 + G)^D$.*

Proof. Using $\|A_i^{(\ell)}\|_1 = \sum_{j=1}^{D} a(j, i^{(\ell)}) + 1 \leq G + 1$ we obtain $\|A_i^{(\ell)}\|_2 \leq G + 1$ and with Hadamard-inequality

$$\Delta \leq (G+1)^D \leq 2^{\mathcal{O}(1/\delta \log^2(1/\delta))}.$$

We can bound the number N of variables in (12.1) by

$$N \leq L \cdot G^D \leq 2^{\mathcal{O}(1/\delta \log^2(1/\delta))}.$$

Then since by Theorem 11.4 the maximum distance max-gap$(\widetilde{A}_\delta)$ can be bounded by $N\Delta$, the assertion follows. □

Again we can show that $\Delta \geq 2^{\Omega(1/\delta \log^2(1/\delta))}$ and still a lower bound for max-gap$(\widetilde{A}_\delta)$ remains open. The upper bound implies that the running time of the dynamic program given in (12.5) is in $2^{\mathcal{O}(1/\delta^2 \log^3(1/\delta))}$, moreover this implies the following theorem

Theorem 12.5. *For Case 1 there is an algorithm that schedules n jobs on m uniform processors producing a schedule with makespan at most $(1+\varepsilon)OPT(\mathcal{I})$ in time*

$$2^{\mathcal{O}(1/\varepsilon^2 \log^3(1/\varepsilon))} poly(\log(n)) + \mathcal{O}(n \log(n)).$$

If $C(A) = poly(1/\varepsilon)$ the running time improves to

$$2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))} poly(\log(n)) + \mathcal{O}(n \log(n)).$$

12.2 Algorithm for Case 2

We partition the bins into two groups

$$\mathcal{B}_0 := \{b_1, \ldots, b_K\} \text{ and } \mathcal{B}_1 := \{b_{K+1}, \ldots, b_m\}.$$

Then, there is a gap $g(1/\delta)$ between the largest bin b_1 and bin b_{K+1}, compare to Figure 12.1. In particular we have $\bar{c}_1/\bar{c}_i < g(1/\delta)$ for $b_i \in \mathcal{B}_0$ and $\bar{c}_1/\bar{c}_i \geq g(1/\delta)$ for $b_i \in \mathcal{B}_1$.

12.2.1 Preprocessing Large Jobs for $\mathcal{B}_0$.

For the preprocessing a job is called *large*, if $\bar{p}_j \geq \delta c_{\min}(\mathcal{B}_0)$, other jobs are *small*. We compute the set of large jobs to be scheduled in $\mathcal{B}_0$ using a dynamic program. Note that according to Lemma 12.2 the number of different rounded large processing times in $\bar{\mathcal{I}}$ is bounded by D and the number of different rounded bin sizes in $\mathcal{B}_0$ is bounded by L and the number of large jobs that can be placed into a bin is bounded by G.

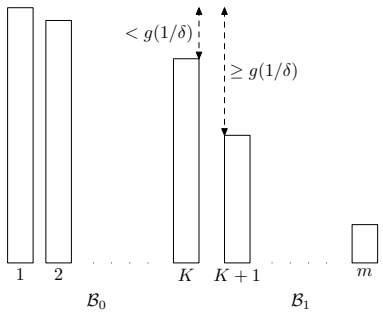

Figure 12.1: Structure of bins in Case 2.

Therefore, the total number of large jobs packed into $\mathcal{B}_0$ is bounded by $K \cdot G$. A possible choice of large jobs for the first K bins can be described by a vector $v = (v_1, \ldots, v_D)$ where $v_j \leq K \cdot G$ is the number of jobs with processing time r_j packed into $\mathcal{B}_0$. Since $D, L = \mathcal{O}(1/\delta \log(1/\delta))$ and $G = \mathcal{O}(1/\delta)$ the total number of possible vectors is at most

$$(K \cdot G)^D = 2^{D \log(K \cdot G)} = 2^{\mathcal{O}(1/\delta \log(1/\delta) \log(K/\delta)))}$$

By a dynamic program we can compute the set of all possible vectors and corresponding packings into the K bins with running time

$$2^{\mathcal{O}(1/\delta \log(1/\delta) \log(K/\delta)))} \tag{12.6}$$

Remark. Note that since $K \leq f(1/delta) = \frac{3 \cdot D \cdot G \cdot C(\widetilde{A}_\delta)}{\delta^2}$ we have $\log(K/\delta) = \mathcal{O}(\log(C(\widetilde{A}_\delta)))$ and the running time in (12.6) becomes

$$2^{\mathcal{O}(1/\delta \log(1/\delta) \log(C(\widetilde{A}_\delta)))}$$

(assuming that at least $C(\widetilde{A}_\delta) \geq D, G, 1/\delta$). Furthermore, if $K \in \mathcal{O}(1/\delta)$,

the running time is

$$2^{\mathcal{O}(1/\delta \log^2(1/\delta))}.$$

We simply go over the machines and store the possible vectors for the first i machines for $i = 1, \ldots, K$. Notice that all *huge* jobs with processing time larger than or equal to $c_{\min}(\mathcal{B}_0)$ have to be placed into $\mathcal{B}_0$. Consequently, a vector v is feasible, if v corresponds to a packing into the K bins and if all huge jobs are placed into $\mathcal{B}_0$. Finally, let S_0 be the free space $\sum_{i=1}^{K} \bar{c}_i - \sum_{j=1}^{D} r_j v_j$ in $\mathcal{B}_0$.

Algorithm 15 Algorithm for Case 2

Input: An instance $\mathcal{I}$ of $Q||C_{\max}$, $\varepsilon > 0$
Output: A schedule of length $(1 + \mathcal{O}(\varepsilon))\mathrm{OPT}(\mathcal{I})$

1: Obtain with LPT algorithm a schedule of length $LPT(\mathcal{I}) \geq 2\mathrm{OPT}(\mathcal{I})$.
2: **for** a candidate value for the makespan $T \in [LS(\mathcal{I})/2, LS(\mathcal{I})]$ **do**
3: Transform the instance to an instance of Bin Packing Different Bin Sizes.
4: **if** the total size of jobs is not bounded by $\sum_{i=1}^{m} c_m$ or
5: $\max \bar{p}_j > \max_i c_i$ **then**
6: increase T and go to Step 3.
7: **end if**
8: Round the processing times and bin capacities and get rounded instance $\bar{\mathcal{I}}$.
9: Preprocess assignment v of large jobs $\bar{p}_j \geq \delta c_{\min}(\mathcal{B}_0)$ for $\mathcal{B}_0$ via dynamic programming.
10: **if** there is no feasible assignment v for $\mathcal{B}_0$ **then**
11: increase T and go to Step 3.
12: **end if**
13: **for** a feasible assignment v **do**
14: Compute a solution of LP (12.7) that allocates remaining jobs with $\bar{p}_j \geq \delta$ fractionally into $\mathcal{B}_1$.
15: **if** LP (12.7) does not have a feasible solution **then**
16: go back to Step 9 (and compute a different vector v).
17: **end if**
18: Round solution of LP (12.7) with new rounding technique using a subroutine for Bin Packing with Different Bin Sizes.
19: Distribute the jobs in additional bins obtained by the above subroutine among the bins in $\mathcal{B}_0$.
20: **end for**
21: Schedule the tiny jobs ($\bar{p}_j < \delta$) behind the large ones slightly increasing the bins sizes.
22: **end for**

12.2.2 Linear Program for Remaining Jobs

Notice that tiny jobs with processing time $\bar{p}_j < \delta$ can be neglected. These jobs can be placed greedily into slightly enlarged bins in the last phase of the algorithm. The remaining jobs for $\mathcal{B}_0$ and $\mathcal{B}_1$ can be chosen and packed via a linear program relaxation. Here the gap $g(1/\delta)$ helps us to move some jobs later into $\mathcal{B}_0$.

We divide the set of all bins in $\mathcal{B}_1$ into N groups B_ℓ with m_ℓ bins of the same size $\bar{c}(\ell)$. For simplicity we use here $B_0 = \mathcal{B}_0$. Furthermore, let n_j be the number of jobs with processing time $\bar{p}_j = \delta(1+\delta)^j$ for $j = 0, \ldots, R$, so that $\delta(1+\delta)^R \leq c_{\min}(\mathcal{B}_0)$ is the largest non-huge processing time. In the following we give an LP-formulation that describes an assignment of the remaining jobs to $\mathcal{B}_1$. Therefore, for any group of bins B with maximum bin capacity $c_{\max}(B)$ we call a job size $\bar{p}_j$ *large in* B or *large with respect to* $c_{\max}(B)$ if $\bar{p}_j \geq \delta c_{\max}(B)$.

For every group B_ℓ with bin size $\bar{c}(\ell)$ we introduce configurations $C_i^{(\ell)}$. A configuration $C_i^{(\ell)}$ is a multiset of jobs that are large in B_ℓ, i.e. with processing times $\bar{p}_j \in [\delta\bar{c}(\ell), \bar{c}(\ell)]$, so that their total size is bounded by $\bar{c}(\ell)$. Let $a(j, i^{(\ell)})$ be the number of the occurrences of $\bar{p}_j$ in $C_i^{(\ell)}$ and let $size(C_i^{(\ell)}) = \sum_j a(j, i^{(\ell)})\bar{p}_j$. Let h_ℓ be the number of configurations for group B_ℓ. In the LP below we use a variable $x_i^{(\ell)}$ to indicate the fractional length configuration $C_i^{(\ell)}$.

Finally, we use a variable $y_{j,\ell}$ to indicate the fractional number of jobs of size $\delta \leq \bar{p}_j < \delta\bar{c}(\ell)$ packed as a small job in B_ℓ. For a job size with $\bar{p}_j \geq \delta\bar{c}(\ell)$ (that is large in B_ℓ) we set $y_{j,\ell} = 0$. For jobs with $\bar{p}_j \in (c_{\max}(\mathcal{B}_1), c_{\min}(\mathcal{B}_0))$ we have $y_{j,0} = n_j$. Furthermore, we may suppose that $y_{j,0} = 0$ for jobs with processing time at least $\delta c_{\min}(\mathcal{B}_0)$. This means that the large jobs for $\mathcal{B}_0$ are placed into $\mathcal{B}_0$ only in the preprocessing step. This assumption is useful for the rounding of the LP.

$$
\begin{aligned}
\sum_i x_i^{(\ell)} &\leq m_\ell \qquad \text{for } \ell = 1,\ldots,N \\
\sum_{\ell,i} a(j, i^{(\ell)}) x_i^{(\ell)} + \sum_\ell y_{j,\ell} &= n_j \qquad \text{for } j = 0,\ldots,R \\
\sum_i size(C_i^{(\ell)}) x_i^{(\ell)} + \sum_j y_{j,\ell}\delta(1+\delta)^j &\leq m_\ell \bar{c}(\ell) \text{ for } \ell = 1,\ldots,N \\
\sum_{j=1}^{R} y_{j,0}\delta(1+\delta)^j &\leq S_0 \\
x_i^{(\ell)} &\geq 0 \text{ for } \ell = 1,\ldots,N \text{ and } i = 1,\ldots,h_\ell \\
y_{j,\ell} &\geq 0 \text{ for } j = 0,\ldots,R \text{ and } \ell = 0,\ldots,N \\
y_{j,0} &= n_j
\end{aligned}
\tag{12.7}
$$

As in [33] the coefficients in the last $N+1$ constraints of the LP can be rounded and each inequality can be transformed so that the converted coefficients are bounded by $\mathcal{O}(n/\delta)$. Then we can bound the facet complexity of the polyhedron corresponding to the corresponding dual linear program by $(\mathcal{O}(R\log(n) + n\log(n/\delta))) = \mathcal{O}(n\log(n/\delta))$. The encoding length of the vector of the objective function of the dual is bounded by $poly(n, 1/\delta)$. Thus, according to 11.3 we achieve a basic optimum solution of (12.7) in time $poly(n, 1/\delta)$.

12.2.3 Rounding the LP-solution

We describe here a new approach to round the solution of the LP using a subroutine for Bin Packing with Different Bin Sizes.

Remark. Actually, we could also use the same rounding method as in [33]. But it requires the gap $g(1/\delta)$ to be much larger: In order to bound the additional $\mathcal{O}(1/\delta^2 \log(1/\delta))$ bins of size $c_{\max}(\mathcal{B}_1)$ by the term $\delta c_{\max}(\mathcal{B}_0)$, the gap $g(1/\delta)$ should be at least $\Omega(1/\delta^3 \log(1/\delta))$.

So recall the conditions of Case 2, that means that

$$
\begin{gathered}
\textit{there exists an index } K \leq f(1/\delta) \textit{ with} \\
\bar{c_1}/\bar{c_i} < g(1/\delta) \textit{ for } 1 \leq i \leq K \textit{ and } \bar{c_1}/\bar{c_i} \geq g(1/\delta) \textit{ for } K+1 \leq i \leq m
\end{gathered}
$$

In our new approach we subdivide $\mathcal{B}_1$ into groups of bins $D_1,\ldots,D_H$ with similar bin sizes. These groups are not necessary equal to the groups

$B_1, \ldots, B_N$ we considered to set up the above LP-relaxation. Then we use the solution of the LP relaxation above to pack the jobs or items via a subroutine for Bin Packing with Different Bin Sizes. For each group D_k the subroutine packs the selected items into the group D_k of bins with different bin sizes plus few additional bins of maximum capacity $c_{\max}(D_k)$. Based on the subdivision the number of large (with respect to $c_{\max}(D_k)$) item sizes can be bounded by $d = \mathcal{O}(1/\delta \log(1/\delta))$ for each group D_k. Using a recent result [35], we are able to pack the selected items into D_k plus $\mathcal{O}(\log^2(d)) = \mathcal{O}(\log^2(1/\delta))$ bins of capacity $c_{\max}(D_k)$. The overall goal is to obtain a packing of almost all jobs into $\mathcal{B}_1$ plus at most $\mathcal{O}(\log^2(1/\delta))$ bins of capacity $c_{\max}(\mathcal{B}_1)$. Finally we use the gap $g(1/\delta)$ to move the jobs which lie in the additional bins onto the faster machines with slightly increased makespan. In the following we explain in detail how this rounding works.

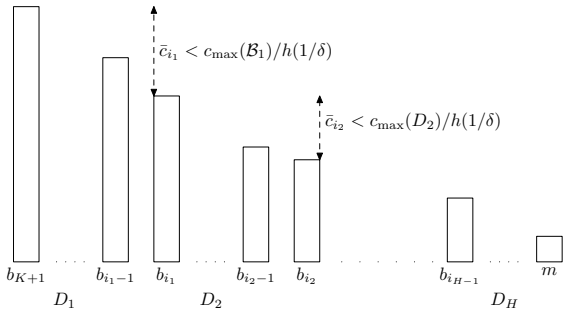

Figure 12.2: Grouping $\mathcal{B}_1$ into D_1 to D_H.

12.2.3.1 Grouping $\mathcal{B}_1$ into $D_1, \ldots, D_H$

Suppose that $\mathcal{B}_1$ has a bin b_{i_1} with $\bar{c}_{i_1} < c_{\max}(\mathcal{B}_1)/h(1/\delta)$, where $h : \mathbb{R}^+ \to \mathbb{R}^+$ is a function with $poly(1/\delta) \geq h(1/\delta) \geq 1/\delta$. Let $i_1 \in \{K+1, \ldots, m\}$ be minimal with that property. In this case we build $D_1 = \{b_{K+1}, \ldots, b_{i_1-1}\}$ and construct the other groups $D_2, \ldots, D_H$ iteratively in the same way. The next group $D_2 = \{b_{i_1}, \ldots, b_{i_2-1}\}$ fulfills the properties $c_{\min}(D_2) = \bar{c}_{i_2-1} \geq c_{\max}(D_2)/h(1/\delta)$ and $\bar{c}_{i_2} < c_{\max}(D_2)/h(1/\delta)$, see Figure 12.2. If all bins have capacity larger than or equal $c_{\max}(\mathcal{B}_1)/h(1/\delta)$, we have only one group $D_1 = \{b_{K+1}, \ldots, b_m\}$. Furthermore, if bin $b_{k_j} \in D_k$ has capacity $\bar{c}(\ell)$ for $\ell \in \{1, \ldots, N\}$ we have $B_\ell \subseteq D_k$. With Lemma 11.2 we con-

clude that the number of different bin sizes in each group D_k is at most $\mathcal{O}(1/\delta \log(h(1/\delta)))$ and the number of large job sizes with respect to D_k is at most $\mathcal{O}(1/\delta \log(h(1/\delta)/\delta))$. Since $h(1/\delta) \leq poly(1/\delta)$, we state the following.

Lemma 12.6. *In every group D_k the number of different bin sizes and the number of different jobs sizes that are large with respect to $c_{\max}(D_k)$ is at most $\mathcal{O}(1/\delta \log(1/\delta))$.*

12.2.3.2 Jobs that are Large in D_k

Consider now a linear program solution $(x_i^{(\ell)}, y_{j,\ell})$ of 12.7 and the reduced linear program (12.7$_k$) with corresponding constraints for the bin group D_k.

$$\begin{aligned} \sum_i x_i^{(\ell)} &= \bar{m}_\ell \text{ for bins with capacity } \bar{c}(\ell) \text{ in } D_k \\ \sum_{\ell,i} a(j, i^{(\ell)}) x_i^{(\ell)} &\geq n_j^{(k)} \text{ for each large jobs size in } D_k \qquad (12.7_k) \\ x_i^{(\ell)} &\geq 0 \text{ for } \ \ell = 1, \ldots, L \end{aligned}$$

The value $\bar{m}_\ell \leq m_\ell$ is the fractional number of bins of size $\bar{c}(\ell)$ in D_k and $n_j^{(k)}$ is the fractional number of large job sizes $\delta(1+\delta)^j$ placed into D_k according to the solution of (12.7). Again, note that a job size $\bar{p}_j$ is large in D_k if $\bar{p}_j = \delta(1+\delta)^j \in [\delta\bar{c}(\ell), \bar{c}(\ell)]$ for at least one bin group B_ℓ in D_k with capacity $\bar{c}(\ell)$. If in LP_k we replace the right hand sides $n_j^{(k)}$ by $\lfloor n_j^{(k)} \rfloor$ for each large job size, we have to cover an integral number of jobs per large job size. Moreover, the following holds.

Lemma 12.7. *If in (12.7$_k$) we replace the right hand side $n_j^{(k)}$ by $\lfloor n_j^{(k)} \rfloor$ the number of non-covered large jobs in D_k can be bounded by $c_{\max}(D_k)(1+\delta)/\delta$.*

Proof. We lose at most one job per large job size. By the rounding of the bin capacities there exists an integer $k \geq 0$, so that $c_{\max}(D_k) = (1+\delta)^k$, moreover for all large jobs sizes $\bar{p}_j$ in D_k we have $\delta(1+\delta)^j \leq (1+\delta)^k$. Therefore, using the the geometric sum over the job sizes the total execution time of non-covered large jobs sizes in D_k can be bounded by

$$\sum_{\bar{p}_j \text{ large in } D_k} \delta(1+\delta)^j \leq c_{\max}(D_k) \sum_{j=0}^{\infty} (1+\delta)^{-j} = c_{\max}(D_k)(1+\delta)/\delta.$$

□

The analysis for the first block D_1 with bound $c_{\max}(D_1)(1+\delta)/\delta$ for the non-covered jobs can be even further improved:

Lemma 12.8. *If* (12.7_k) *we replace* $n_j^{(1)}$ *by* $\lfloor n_j^{(1)} \rfloor$*, the total processing time of the non-covered large jobs in* D_1 *is bounded by* $(1+\delta)\min\{c_{\min}(\mathcal{B}_0), {}^{c_{\max}(D_1)}/_{\delta}\}$.

Proof. All jobs with processing time $\bar{p}_j \in (c_{\max}(D_2), \delta c_{\min}(\mathcal{B}_0))$ are large or small corresponding to bins in D_1, but small corresponding to $\mathcal{B}_0$. Notice, that jobs with $\bar{p}_j \in [\delta c_{\min}(\mathcal{B}_0), c_{\max}(D_1)]$ are large corresponding to $\mathcal{B}_0$ and chosen via the preprocessing step above. This implies that we may assume $y_{j,0} = 0$ for the corresponding values. Since $c_{\min}(\mathcal{B}_0) > c_{\max}(D_1)$ and $c_{\max}(D_1)/c_{\max}(D_2) > h(1/\delta) \geq 1/\delta$, we have $c_{\max}(D_2) < \delta c_{\max}(D_1) < \delta c_{\min}(\mathcal{B}_0)$. Therefore, a job with $\bar{p}_j \in [\delta c_{\min}(\mathcal{B}_0), c_{\max}(D_1)]$ does not fit into D_2. Moreover, we can assume that the values $n_j^{(1)}$ are integral. Since the values $n_j^{(1)}$ are integral for processing times $\bar{p}_j \in [\delta c_{\min}(\mathcal{B}_0), c_{\max}(D_1)]$, the not covered large jobs have processing times $\leq \delta c_{\min}(\mathcal{B}_0)$. Consequently, the total sum of the execution times of the large jobs not covered by the bin packing subroutine in $\mathcal{D}_1$ is bounded by $\min\{(1+\delta)c_{\min}(\mathcal{B}_0), c_{\max}(D_1)(1+\delta)/\delta\}$ (as in Lemma 12.7 using the geometric sum argument applied to $\delta c_{\min}(\mathcal{B}_0)$ instead of $c_{\max}(\mathcal{D}_k)$). □

Now a (fractional) solution of the modified (12.7_k) can be transformed into an integral solution using the algorithm for Bin Packing with Different Bin Sizes in [35]. That means $\lfloor n_j^{(k)} \rfloor$ jobs of size $\delta(1+\delta)^j$ can be packed into the bins in D_k plus $\mathcal{O}(\log^2(d))$ additional bins of size $c_{\max}(D_k)$ [35] (where $d = \mathcal{O}(1/\delta \log(1/\delta))$ is the number of different job sizes that are large in D_k). Let $\gamma \log^2(1/\delta)$ for some constant $\gamma \geq 0$ be an upper bound for the number of additional bins for D_k. Notice that it is allowed to use m_ℓ bins instead of the fractional number $\bar{m}_\ell$ of bins in each group B_ℓ. This is sufficient, since the overall area $\sum_{\ell: B_\ell \subseteq D_k} Area(large, \ell)$ of the large jobs packed into D_k plus the extra bins remains the same.

12.2.3.3 Rounding the Small Jobs in B_ℓ

The $(y_{j,\ell})$ variables can be rounded as before [33] using a result of Lenstra et al. [50], but here we have to round up (instead of down) the values N_j, the (fractional) number of jobs of size $\delta(1+\delta)^{k_j}$ assigned as a small job to

the blocks B_ℓ for all job sizes j. Almost all corresponding small jobs can be packed directly into $\mathcal{B}_1$. For each block B_ℓ there is at most one fractional variable $\widetilde{y}_{j,\ell}$ and a corresponding small job that does not fit completely in B_ℓ. But this job can be packed into one bin of B_ℓ, if we increase the corresponding bin size slightly. So we do not need additional bins here.

12.2.4 Bounding and Distributing the Additional jobs

We can bound the total execution time of jobs in the extra bins caused by the rounding in the following way.

Lemma 12.9. *The total execution time of the jobs in the additional bins for $D_1, \ldots, D_H$ is at most*

$$(1+\delta)\min\{c_{\min}(\mathcal{B}_0), (1/\delta)c_{\max}(D_1)\} + \alpha \log^2(1/\delta)c_{\max}(D_1). \tag{12.8}$$

for some constant $\alpha \geq 0$.

Proof. According to Lemma 12.7 and by the rounding technique mentioned above the total execution time of all jobs in the additional bins for D_k is at most

$$\left(\frac{1+\delta}{\delta} + \gamma \log^2(1/\delta)\right) c_{\max}(D_k) \leq \frac{\lambda}{\delta} c_{\max}(D_k)$$

where $\lambda \geq 0$ is a constant. Using $h(1/\delta)c_{\max}(D_{k+1}) \leq c_{\max}(D_k)$ this implies the following bound for the extra execution time in $D_2, \ldots, D_H$

$$\begin{aligned}\sum_{k\geq 2} \tfrac{\lambda}{\delta} c_{\max}(D_k) \leq \tfrac{\lambda}{\delta} \sum_{k\geq 0} \tfrac{c_{\max}(D_2)}{h(1/\delta)^k} &\overset{h(1/\delta)\geq 1/\delta}{\leq} \tfrac{\lambda}{\delta} c_{\max}(D_2) \sum_{k\geq 0} \delta^k \\ &\overset{\delta\leq 1/2}{\leq} \frac{2\lambda}{\delta} c_{\max}(D_2) \leq 2\lambda c_{\max}(D_1).\end{aligned}$$

With Lemma 12.8 the total extra execution time in bin group D_1 is at most

$$(1+\delta)\min\{c_{\min}(\mathcal{B}_0), (1/\delta)c_{\max}(D_1)\} + \gamma \log^2(1/\delta)c_{\max}(D_1).$$

So in total we can bound the additional term by

$$(1+\delta)\min\{c_{\min}(\mathcal{B}_0), (1/\delta)c_{\max}(D_1)\} + (\gamma \log^2(1/\delta) + 2\lambda)c_{\max}(D_1).$$

□

Notice that all jobs involved in the first term of (12.8) are small corresponding to $\mathcal{B}_0$ (i.e. $\bar{p}_j \leq \delta c_{\min}(\mathcal{B}_0)$). The other jobs that contribute to $\mathcal{O}(\log^2(1/\delta))c_{\max}(D_1)$ could be large corresponding to $\mathcal{B}_0$, but have processing time $\bar{p}_j \leq c_{\max}(D_1) = c_{\max}(\mathcal{B}_1)$.

We can now distribute the jobs corresponding to the additional term among the bins of $\mathcal{B}_0$. We proceed by case distinction.

Case 2.1: $K \geq \lceil (1+\delta)/\delta \rceil$.

In this case the additional term $(1+\delta)c_{\min}(\mathcal{B}_0)$ with jobs of size at most $\delta c_{\min}(\mathcal{B}_0)$ can be distributed onto the first $\lceil (1+\delta)/\delta \rceil$ bins with additional load $\leq 2\delta c_{\min}(\mathcal{B}_0)$. Here simply use a greedy algorithm that distributes a load of at least $\delta c_{\min}(\mathcal{B}_0)$ and at most $2\delta c_{\min}(\mathcal{B}_0)$ on the first bins. The last bin gets load at most $\delta c_{\min}(\mathcal{B}_0)$. The number of bins needed is at most $\lceil (1+\delta)/\delta \rceil \leq K$. This implies that the first K bins get load at most $\bar{c}_i + 2\delta c_{\min}(\mathcal{B}_0) \leq (1+2\delta)\bar{c}_i$. This implies the following result

Lemma 12.10. *Let b_{K+1} be the first bin such that $\bar{c}_1/\bar{c}_{K+1} \geq g(1/\delta)$ and $K \geq \lceil (1+\delta)/\delta \rceil$, then we can compute an approximate schedule of length $(1+\mathcal{O}(\delta))\mathrm{OPT}(I)$ in time*

$$2^{\mathcal{O}(1/\delta \log(1/\delta) \log(C(\widetilde{A}_\delta)))} poly(1/\delta, n).$$

Proof. Since $g(1/\delta) \geq 1/\delta \log^2(1/\delta)$ and $\frac{c_1}{c_{K+1}} \geq g(1/\delta)$ we can schedule the remaining jobs of length $\bar{p}_j \leq c_{\max}(\mathcal{B}_1)$ with total execution time $\alpha \log^2(1/\delta)c_{\max}(D_1)$ into $\mathcal{B}_0$ in the first largest bin: We have

$$\alpha \log^2(1/\delta)c_{\max}(\mathcal{B}_1) \leq \alpha \frac{\log^2(1/\delta)}{g(1/\delta)} c_{\max}(\mathcal{B}_0) \leq \alpha\delta c_{\max}(\mathcal{B}_0).$$

With the Remark in Section 12.2.1 the running-time follows. □

Case 2.2: $c_{max}(\mathcal{B}_0)/c_{max}(\mathcal{B}_1) \geq 1/\delta^2$.

In this case the additional processing time is bounded by

$$((1+\delta)/\delta)c_{\max}(D_1) + \alpha \log^2(1/\delta)c_{\max}(D_1) \leq \frac{\alpha'}{\delta} c_{\max}(D_1) \leq \alpha'\delta c_{\max}(\mathcal{B}_0).$$

for some constant $\alpha' \geq 0$. This amount can be packed onto the largest bin by increasing its bin size to $(1 + \mathcal{O}(\delta))c_{\max}(\mathcal{B}_0)$.

Case 2.3: $K < \lceil (1+\delta)/\delta \rceil$ and $c_{max}(\mathcal{B}_0)/c_{max}(\mathcal{B}_1) < 1/\delta^2$.
In this case we use our general assumption on $g(1/\delta)$. This implies that $\bar{c}_1/\bar{c}_{K+1} \geq 1/\delta \log^2(1/\delta)$.

Let us consider jobs with processing time $\bar{p}_j$ in the interval

$$I = [\delta c_{\max}(D_1), \min\{\delta c_{\min}(\mathcal{B}_0), c_{\max}(D_1)\}]$$

and denote with $P_{cri} = \{\bar{p}_j \in I\}$ the processing times of those critical jobs. Since $\max(P_{cri})/\min(P_{cri}) \leq 1/\delta$, with Lemma 11.2 there are at most $\mathcal{O}(1/\delta \log(1/\delta))$ different rounded processing times for the critical jobs. Furthermore, the total number of critical jobs that can be packed into $\mathcal{B}_0$ is at most

$$\frac{K c_{\max}(\mathcal{B}_0)}{\delta c_{\max}(D_1)} \leq \frac{\mathcal{O}(1/\delta^2) c_{\max}(\mathcal{B}_0)}{c_{\max}(D_1)} \leq \mathcal{O}(1/\delta^4)$$

under the second assumption above.

The critical jobs can be chosen for $\mathcal{B}_0$ and packed into the bins in $\mathcal{B}_0$ during our preprocessing step in Section 12.2.1. The large ($\geq \delta c_{\min}(\mathcal{B}_0)$) and critical jobs can be selected and pre-packed into $\mathcal{B}_0$ in time

$$(1/\delta^4)^{\mathcal{O}(1/\delta \log(1/\delta))} \leq 2^{\mathcal{O}(1/\delta \log^2(1/\delta))}.$$

Then, for each feasible vector v we set up a linear program, round the solution and place the large (corresponding to $D_1, \ldots, D_h$) jobs into the bins plus few additional bins (again as in [35]) of total size

$$\gamma \log^2(1/\delta) c_{max}(D_1) \leq \frac{\gamma \log^2(1/\delta) c_{\max}(\mathcal{B}_0)}{g(1/\delta)} \leq \gamma \delta c_{max}(\mathcal{B}_0)$$

since $g(1/\delta) \geq 1/\delta \log^2(1/\delta)$. In this case, for all critical jobs with $\bar{p}_j \in P_{cri}$ we may suppose that $y_{j,0} = 0$ (no further such jobs are packed into $\mathcal{B}_0$ via the LP). This implies that the corresponding values $n_j^{(1)}$ are integral. Here again we use the fact that a large job in D_1 with $\bar{p}_j \geq \delta c_{max}(D_1) > c_{max}(D_2)$ does not fit into bin group D_2. So comparing to Lemma 12.8 in this case

the term for the non-covered jobs in D_1 can be bounded by

$$(1+\delta)c_{max}(D_1) \leq \frac{(1+\delta)c_{max}(\mathcal{B}_0)}{g(1/\delta)} \leq \mathcal{O}(\delta)c_{\max}(\mathcal{B}_0).$$

Furthermore, note that for $K = \mathcal{O}(1/\delta)$, the running time of the dynamic program given by (12.6) for the large jobs in $\mathcal{B}_0$ is bounded by

$$2^{\mathcal{O}(1/\delta \log^2(1/\delta))} poly(1/\delta, n).$$

This gives

Lemma 12.11. *Let b_{K+1} be the first bin such that $\bar{c}_1/\bar{c}_{K+1} \geq 1/\delta \log^2(1/\delta)$, $K < \lceil (1+\delta)/\delta \rceil$ and $\bar{c}_1/\bar{c}_{K+1} < 1/\delta^2$, then we can compute an approximate schedule of length $(1+\mathcal{O}(\delta))\mathrm{OPT}(I)$ in time $2^{\mathcal{O}(1/\delta \log^2(1/\delta))} poly(1/\delta, n)$.*

The analysis of all three subcase implies the following result.

Theorem 12.12. *In each subcase of Case 2 Algorithm 15 produces a schedule of length $(1+\mathcal{O}(\delta))\mathrm{OPT}(\mathcal{I})$ in time $2^{\mathcal{O}(1/\delta \log(1/\delta) \log(C(\widetilde{A}_\delta))} poly(1/\delta, n)$.*

12.3 Algorithm for Case 3

For this section we need to define the value $C(\widetilde{A}_\delta)$ more precisely.

Definition 12.13. From now on let

$$C(\widetilde{A}) := L \cdot (G^2 + G)^D.$$

In this case we have two or three bin groups depending on the shape of bin sizes as depicted in Figure 12.3. Let

$$\mathcal{B}_0 = \{b_1, \ldots, b_K\}$$

be the set of the largest bins. Then, we define

$$\mathcal{B}_1 = \{b_i | i > K, \bar{c}_i \geq \delta c_{min}(\mathcal{B}_0)\} \text{ and } \mathcal{B}_2 = \mathcal{B} \setminus (\mathcal{B}_0 \cup \mathcal{B}_1)$$

as the remaining bins. If $\mathcal{B}_1 \neq \emptyset$ we distinguish large, medium and small jobs. A job is called

large if $\bar{p}_j > \delta c_{min}(\mathcal{B}_0)$, *medium* if $\bar{p}_j \in (\delta c_{min}(\mathcal{B}_1), \delta c_{min}(\mathcal{B}_0)]$ and else *small*.

Note that for a medium job we have $\bar{p}_j \leq \delta c_{\min}(\mathcal{B}_0) \leq c_{\max}(\mathcal{B}_1)$ by construction.

Remark. If $\mathcal{B}_1 = \emptyset$ we do not have medium jobs. In this case we have for all $i > K$ that $\bar{c}_i \leq \delta c_{\min}(B_0)$. Thus, we have an additional gap between $\bar{c}_K$ and $\bar{c}_{K+1}$, i.e.

$$\frac{\bar{c}_K}{\bar{c}_{K+1}} = \frac{c_{\min}(\mathcal{B}_0)}{c_{\max}(\mathcal{B}_2)} > \frac{c_{\min}(\mathcal{B}_0)}{\delta c_{\min}(\mathcal{B}_0)} = \frac{1}{\delta}.$$

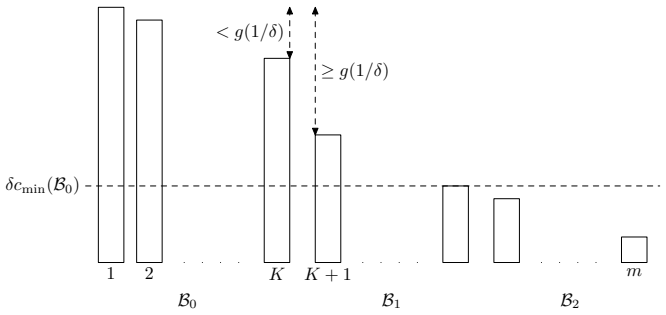

Figure 12.3: Structure of bins in Case 3.

With Lemma 12.2 we conclude that the number of different rounded bin sizes is in $\mathcal{B}_0$ is bounded by L and the number of different rounded large job sizes in $\bar{\mathcal{I}}$ is bounded by D.

Furthermore, let $P := 2/\delta \log(1/\delta) + 1$, then we can show the following

Lemma 12.14. *The number of different rounded medium large job sizes in $\bar{\mathcal{I}}$ and the number of different rounded bin sizes in $\mathcal{B}_1$ is bounded by P.*

Proof. Since

$$\frac{\delta c_{\min}(\mathcal{B}_0)}{\delta c_{\min}(\mathcal{B}_1)} \leq \frac{c_{\min}(\mathcal{B}_0)}{\delta c_{\min}(\mathcal{B}_0)} \leq 1/\delta$$

the assertion follows with Lemma 11.2. □

12.3.1 LP-Formulation

Now we divide the set $\mathcal{B} = \mathcal{B}_0 \cup \mathcal{B}_1 \cup \mathcal{B}_2$ into N groups B_ℓ with m_ℓ bins with the same rounded bin size $\bar{c}(\ell)$ for $\ell = 1, \ldots, N$ and set up a linear program similar to (12.7) in Section 12.2.2. Later we also consider reduced LPs for the first two bin groups separately. Again we neglect here jobs with processing time $\bar{p}_j < \delta$.

In the LP (12.10) below we use a variable $x_i^{(\ell)}$ to indicate the fractional length of a multiset (configuration) $C_i^{(\ell)}$ of processing times $\bar{p}_j \in [\delta\bar{c}(\ell), \bar{c}(\ell)]$ packed into bins of size $\bar{c}(\ell)$. The value h_ℓ denotes the number of different configurations for bins of size $\bar{c}(\ell)$. Let $a(j, i^{(\ell)})$ be the number of the occurrences of $\bar{p}_j$ in $C_i^{(\ell)}$ and let $size(C_i^{(\ell)}) = \sum_j a(j, i^{(\ell)})\bar{p}_j$. Furthermore, let n_j be the number of jobs with processing time $\bar{p}_j = \delta(1+\delta)^j$ for $j = 0, \ldots, R$ (where $\delta(1+\delta)^R$ is the largest jobs size). Finally, we use a variable $y_{j,\ell}$ to indicate the fractional number of jobs of size $\delta \leq \bar{p}_j < \delta\bar{c}(\ell)$ packed as a small job in B_ℓ.

$$
\begin{aligned}
\sum_i x_i^{(\ell)} &\leq m_\ell && \text{for } \ell = 1, \ldots, N \\
\sum_{\ell,i} a(j, i^{(\ell)}) x_i^{(\ell)} + \sum_\ell y_{j,\ell} &= n_j && \text{for } j = 0, \ldots, R \\
\sum_i size(C_i^{(\ell)}) x_i^{(\ell)} + \sum_j y_{j,\ell}\delta(1+\delta)^j &\leq m_\ell \bar{c}(\ell) && \text{for } \ell = 1, \ldots, N \qquad (12.9) \\
x_i^{(\ell)} &\geq 0 && \text{for } \ell = 1, \ldots, N \text{ and } i = 1, \ldots, h_\ell \\
y_{j,\ell} &\geq 0 && \text{for } j = 0, \ldots, R \text{ and } \ell = 0, \ldots, N
\end{aligned}
$$

We suppose that all jobs fit into the bins, i.e. $\delta(1+\delta)^R \leq c_{max}(\mathcal{B}_0)$; otherwise there is no schedule with the corresponding makespan in the binary search.

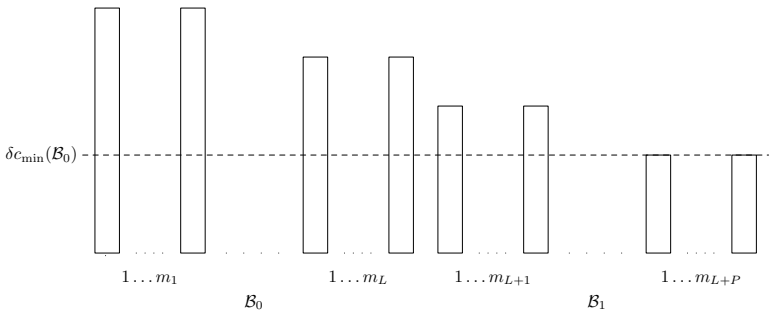

Figure 12.4: Groups of similar capacities in $\mathcal{B}_0 \cup \mathcal{B}_1$.

12.3.2 Allocating Large Jobs into $\mathcal{B}_0$ and $\mathcal{B}_1$.

From Lemma 12.2 we know that $\mathcal{B}_0$ consists of L bin groups B_ℓ and with Lemma 12.14 we have that $\mathcal{B}_1$ consists of P bin groups, see also Figure 12.4.

Furthermore, assume that the LP (12.9) and the corresponding ILP have a solution. Let $(x_i^{(\ell)}, y_{j,\ell})$ be a linear program solution of (12.9). In 12.10 we consider the $(x_i^{(\ell)})$ variables of a solution of (12.9) and the constraints for the first $L+P$ bin groups. Let $\delta(1+\delta)^{R_m}$ be the smallest medium job size and let $\delta(1+\delta)^{R_\ell}$ be the smallest large job size.

$$\begin{aligned} \sum x_i^{\ell} \leq \bar{m}_\ell \leq m_\ell \qquad & \text{for } \ell = 1, \ldots, L+P \\ \sum_{\ell,i} a(j, i^{(\ell)}) x_i^{(\ell)} \geq \bar{n}_j \qquad & \text{for } j = R_m, \ldots, R \\ x_i^{(\ell)} \geq 0 \quad & \text{for } \ell = 1, \ldots, L+P \text{ and } i = 1, \ldots, h_\ell \end{aligned} \tag{12.10}$$

The value $\bar{m}_\ell \leq m_\ell$ is the fractional number of bins of size $\bar{c}(\ell)$ in $\mathcal{B}_0 \cup \mathcal{B}_1$ and $\bar{n}_j$ is the (fractional) number of job sizes $\delta(1+\delta)^j$ placed into $\mathcal{B}_0 \cup \mathcal{B}_1$ according to the solution of (12.9).

For a large job size we have $\bar{p}_j > \delta c_{min}(\mathcal{B}_0) > c_{\max}(\mathcal{B}_2)$. Hence, all the large jobs have to be scheduled in $\mathcal{B}_0 \cup \mathcal{B}_1$. Consequently, in (12.9) we describe them by configuration variables only and so the number $\bar{n}_j$ for large jobs covered by the LP (12.10) above is integral and therefore $\bar{n}_j = n_j$ for $j = R_\ell, \ldots, R$.

For medium job sizes, there are $y_{j,\ell}$ variables in (12.9) and we have in general fractional variables $\bar{n}_j \leq n_j$. Note that a configuration $C_i^{(\ell)}$ in $\mathcal{B}_0$ contains only large job sizes by construction and a configuration $C_i^{(\ell)}$ in $\mathcal{B}_1$ may contain both, large and medium job sizes.

Let $\hat{C}_k^{(\ell)}$ be a configuration with only large job sizes in bin group B_ℓ in $\mathcal{B}_1$. For the rest of this chapter we call $\hat{C}_k^{(\ell)}$ a big configuration. Then, the configurations with both, medium and large job sizes, can be partitioned into groups with the same arrangement of large jobs according to configuration $\hat{C}_k^{(\ell)}$ (containing only large jobs). Let $Index(k, \ell)$ be the set of all indices i such that $C_i^{(\ell)}$ coincides with $\hat{C}_k^{(\ell)}$ for the large job sizes. Let $z_k^{(\ell)}$ denote the the length of configuration $\hat{C}_k^{(\ell)}$ and set $z_k^{(\ell)} := \sum_{i \in Index(k,\ell)} x_i^{(\ell)}$. Then $(x_i^{(\ell)}, z_k^{(\ell)})$ is a feasible solution of the following LP (12.11) for the large job sizes in $\mathcal{B}_0 \cup \mathcal{B}_1$. Since all large jobs have to be placed into the first $L+P$ bin groups, there exists an integral solution for (12.11), too.

$$
\begin{aligned}
\sum_i x_i^{(\ell)} &\leq m_\ell \text{ for } \ell = 1, \ldots, L \\
\sum_k z_k^{(\ell)} &\leq m_\ell \text{ for } \ell = L+1, \ldots, L+P \\
\sum_{\ell,i} a(j, i^{(\ell)}) x_i^{(\ell)} + \sum_{\ell,k} a(j, k^{(\ell)}) z_k^{(\ell)} &\geq n_j \text{ for } j = R_\ell, \ldots, R \\
x_i^{(\ell)} &\geq 0 \qquad \text{for } \ell = 1, \ldots, L+P \text{ and } i = 1, \ldots, h_\ell
\end{aligned} \tag{12.11}
$$

Lemma 12.15. *If $\widetilde{A}'_\delta$ denotes the constraint matrix of the linear program* (12.11) *above, then we have* $\max\text{-}gap(\widetilde{A}'_\delta) \leq \left(1 + \frac{P}{L}\right) C(\widetilde{A}_\delta) \leq 3 \cdot C(\widetilde{A}_\delta)$.

Proof. The LP (12.11) is a configuration LP for a scenario where jobs with D different sizes has to be placed in bins of $L + P = \mathcal{O}(1/\delta \log(1/\delta))$ different bin sizes that each can hold at most G jobs. Therefore, the constraint matrix $\widetilde{A}'_\delta$ has a similar structure as $\widetilde{A}_\delta$ in Definition 12.3, but has has at most $(L+P)G^D$ column vectors of the form (12.2). But since $rank = (\widetilde{A}'_\delta) = D = rank(\widetilde{A}_\delta)$ the absolute value of any subdeterminant in both matrices is bounded by $(G+1)^D$. Then with Theorem 11.4 and Lemma 12.4 we can bound $\max\text{-gap}(\widetilde{A}'_\delta)$ by

$$
\begin{aligned}
(L+P)(G^2+G)^D &\leq \left(1 + \frac{P}{L}\right) L \cdot (G^2+G)^D \\
&\leq \left(1 + \frac{P}{L}\right) C(\widetilde{A}_\delta).
\end{aligned}
$$

With the definition of P, L and since $g(1/\delta) \geq 1/\delta \log^2(1/\delta))$ we compute that $\frac{P}{L} \leq \frac{2\log(1/\delta)}{\log(g(1/\delta))} \leq \frac{2\log(1/\delta)}{\log(1/\delta)} \leq 2$ Then according to Lemma 12.4 we have

$$
\begin{aligned}
\left(1 + \frac{P}{L}\right) C(\widetilde{A}_\delta) &\leq 3 \cdot C(\widetilde{A}_\delta) \\
&= 2^{\mathcal{O}(1/\delta \log^2(1/\delta))}.
\end{aligned} \tag{12.12}
$$

□

With Lemma 12.15 we conclude that there exists an integral solution $(\hat{x}_i^{(\ell)}, \hat{z}_k^{(\ell)})$ with

$$
\|\hat{x}_i^{(\ell)} - x_i^{(\ell)}\|_\infty \leq 3 \cdot C(\widetilde{A}_\delta) \text{ and } \|\hat{z}_k^{(\ell)} - z_k^{(\ell)}\|_\infty \leq 3 \cdot C(\widetilde{A}_\delta).
$$

where $C(\widetilde{A}_\delta)$ is defined in Definition 12.13.

We can now obtain an assignment of a subset of the large jobs to the bins in $\mathcal{B}_0 \cup \mathcal{B}_1$ and as in equation (12.3) reduce the instance $\bar{\mathcal{I}}$ to an instance $\bar{\mathcal{I}}_{red}$ with As in Lemma 11.1 the number of large jobs $\widetilde{n}$ in $\bar{\mathcal{I}}_{red}$ is bounded by

$$\widetilde{n} \leq 3 \cdot D \cdot C(\widetilde{A}_\delta) \cdot G \tag{12.13}$$

Moreover, this implies that we need at most $\widetilde{M} \leq \widetilde{n}$ bins in $\mathcal{B}_0 \cup \mathcal{B}_1$ for the large jobs in $\bar{\mathcal{I}}_{red}$. By dynamic programming we can find an assignment of the large jobs in $\bar{\mathcal{I}}_{red}$ to the bins in $\mathcal{B}_0 \cup \mathcal{B}_1$ in time

$$\widetilde{n}^D = 2^{D \log(\widetilde{n})} = 2^{\mathcal{O}(1/\delta \log(1/\delta) \log(C(\widetilde{A}_\delta)))}. \tag{12.14}$$

since $D = \mathcal{O}(1/\delta \log(1/\delta))$. The main difficulty now is to handle the medium jobs.

12.3.3 Allocating Medium Jobs

The main difficulty now is to handle the medium jobs. We proceed stepwise.

12.3.3.1 Medium Jobs for $\mathcal{B}_1$.

We first allocate medium jobs into $\mathcal{B}_1$. Therefore consider the part of LP (12.11) for the medium and large jobs corresponding to bin group $\mathcal{B}_1$.

Take out for a moment the large jobs $I_{large,dp}$ placed by the dynamic program into $\mathcal{B}_1$. Notice that these large jobs have occupied a subset $M_{large,dp}$ of only $\widetilde{M} \leq \widetilde{n}$ bins in $\mathcal{B}_1$. Furthermore, notice that there are still large jobs preassigned via the big configurations $\hat{C}_k^{(\ell)}$ of length $\lceil z_k^{(\ell)} - 3C(\widetilde{A}_\delta) \rceil = \lceil \sum_{i \in Index(k,\ell)} x_i^{(\ell)} - 3C(\widetilde{A}_\delta) \rceil$ in $\mathcal{B}_1$. Since we have a feasible solution of LP (12.9) for all jobs, the residual configurations $C_i^{(\ell)}$ (restricted to medium job sizes) with fractional lengths $x_i^{(\ell)}$ fit into the gaps either besides their corresponding big configurations of lengths $\lceil z_k^{(\ell)} - 3C(\widetilde{A}_\delta) \rceil$ or after them. Therefore, the placement of medium jobs into the gaps can be seen as an instance of Bin Packing with Different Sin Sizes.

Step 1: Round the $x_i^{(\ell)}$ variables for $\mathcal{B}_1$ as in Section 12.2.3 and use the subroutine for Bin Packing with Different Bin Sizes in [35] to pack $\lfloor \bar{n}_j \rfloor$

Algorithm 16 Algorithm for Case 3

Input: An instance $\mathcal{I}$ of $Q||C_{max}$, $\varepsilon > 0$
Output: A schedule of length $(1 + \mathcal{O}(\varepsilon))\text{OPT}(\mathcal{I})$

1: Obtain with LPT algorithm a schedule of length $LPT(\mathcal{I}) \geq 2\text{OPT}(\mathcal{I})$.
2: **for** a candidate value for the makespan $T \in [LS(\mathcal{I})/2, LS(\mathcal{I})]$ **do**
3: Transform the instance to an instance of Bin Packing Different Bin Sizes.
4: **if** the total size of jobs is not bounded by $\sum_{i=1}^{m} c_m$ or
5: $\max \bar{p}_j > \max_i c_i$ **then**
6: increase T and go to Step 3.
7: **end if**
8: Round the processing times and bin capacities and get rounded instance $\bar{\mathcal{I}}$.
9: Distinguish small, medium and large jobs.
10: Compute a solution of LP (12.9) that allocates the jobs with $\bar{p}_j \geq \delta$ fractionally into $\mathcal{B}_0 \cup \mathcal{B}_1 \cup \mathcal{B}_2$.
11: **if** LP (12.9) does not have a feasible solution **then**
12: increase T and go back to Step 3.
13: **end if**
14: Allocate a subset of the large jobs into $\mathcal{B}_0 \cup \mathcal{B}_1$ and reduce the instance to $\bar{\mathcal{I}}_{red}$.
15: Via dynamic programming obtain an assignment of the remaining large jobs in $\bar{\mathcal{I}}_{red}$ into $\mathcal{B}_0 \cup \mathcal{B}_1$.
16: **if** the dynamic program does not find a feasible solution **then**
17: increase T and go to Step 3.
18: **end if**
19: Round solution of LP (12.9) with new rounding technique using a subroutine for Bin Packing with Different Bin Sizes and obtain assignment of the remaining jobs with $\bar{p}_j \geq \delta$.
20: Rearrangement of some medium and small jobs compensating the error possibly made by the dynamic program for large jobs in Step 15.
21: **end for**
22: Schedule the small jobs with $\bar{p}_j < \delta$ behind the large ones. .

jobs of each medium job size into the gaps of $\mathcal{B}_1$ plus $\mathcal{O}(\log^2(1/\delta))$ bins of size $c_{max}(\mathcal{B}_1)$.

The (medium) jobs in the additional bins obtained by the packing subroutine have processing times $\bar{p}_j \leq \min\{\delta c_{\min}(\mathcal{B}_0), c_{\max}\mathcal{B}_1\}$. Furthermore, caused by the rounding (since we replace the right hand sides n_j with $\lfloor n_j \rfloor$) we have to cover one additional job per medium size and to repack medium jobs with total execution time $\mathcal{O}(1/\delta \min\{c_{\max}(\mathcal{B}_1), \delta c_{\min}(\mathcal{B}_0)\})$. As $K \geq f(1/\delta) \geq \lceil (1+\delta)/\delta \rceil$ we can distribute those medium jobs among the first K bins only slightly increasing the makespan as in Case 2.1 in Section 12.2.

Step 2: Remove the set A_{medium} of medium jobs placed into the bins in the subset $M_{large,dp} \subseteq \mathcal{B}_1$. Reinsert the large jobs from $I_{large,dp}$ in these bins and place fractionally a subset of A_{medium} into the remaining gaps.

If the total area of the large jobs $I_{large,dp}$ placed via the dynamic programm is at most

$$\sum_{\ell=L+1}^{L+P} \sum_{k} \left(z_k^{(\ell)} - \lceil z_k^{(\ell)} - 3C(\widetilde{A}_\delta) \rceil \right) size(\hat{C}_k^{(\ell)})$$

then all medium jobs fit fractionally into the gaps in $\mathcal{B}_1$. Let $A_{medium,fr}$ be the set of fractional medium jobs placed into $\mathcal{B}_1$. Notice that the cardinality $|A_{medium,fr}|$ is at most $\widetilde{M} \leq \widetilde{n}$. These jobs can be placed separately in the first K bins increasing the makespan only by $\mathcal{O}(\delta)$ since $K \geq f(1/\delta) = \frac{3 \cdot D \cdot C(\tilde{A}_\delta) \cdot G}{\delta^2} \geq \widetilde{n}$

If the total area of large jobs is larger than the LP bound, we have to place the remaining set $A'_{medium} \subseteq A_{medium}$ of medium jobs into $\mathcal{B}_0$. Since we have $A_{medium} \leq \widetilde{M} c_{\max}(\mathcal{B}_1)$, we conclude

$$|A'_{medium}| \leq \frac{\widetilde{M} c_{\max}(\mathcal{B}_1)}{\delta c_{\min}(\mathcal{B}_1)} \leq \frac{\widetilde{M} c_{\max}(\mathcal{B}_1)}{\delta^2 c_{\min}(\mathcal{B}_0)} \leq \widetilde{M}/\delta^2 \leq \widetilde{n}/\delta^2.$$

As $K \geq f(1/\delta) = \widetilde{n}/\delta^2$ also in this case we can distribute these jobs among the first K bins.

12.3.3.2 Medium Jobs for $\mathcal{B}_2$ and $\mathcal{B}_0$.

Step 3: Round the $(x_i^{(\ell)})$ variables corresponding to $\mathcal{B}_2$ and place the jobs via a bin packing subroutine similar to Case 2 into $\mathcal{B}_2$ plus some additional

bins of size $c_{max}(\mathcal{B}_2)$.

Step 4: Round the $(y_{j,\ell})$ variables over the bin groups B_ℓ and place the corresponding jobs greedily onto the machines.

Here we have to increase the bin sizes slightly and to place in addition one fractional job per bin group on one machine. As in Step 2 it is also possible that the total area of large jobs placed via the dynamic program is larger than $\sum_{\ell=1}^{L}\sum_i \left(x_i^{(\ell)} - \lceil x_i^{(\ell)} - 3C(\widetilde{A}_\delta)\rceil\right) size(C_k^{(\ell)})$. Then those large jobs block an area in $\mathcal{B}_0$ can not be used for medium and small jobs. We show in the lemma below that the corresponding medium jobs can be placed separately into bins in $\mathcal{B}_0$. Furthermore, some small jobs have to be placed into $\mathcal{B}_1$. But this is easier and possible, since these jobs are also small corresponding to the bins in $\mathcal{B}_1$ and the total area of large, medium and small jobs corresponding to the variable values $x_i^{(\ell)}$ and $y_{j,\ell}$ for $\ell = 1,\ldots,K+L$ fits into $\mathcal{B}_0 \cup \mathcal{B}_1$.

Lemma 12.16. *The medium jobs assigned via the $(y_{j,\ell})$ variables to $\mathcal{B}_0$ that can not be placed into $\mathcal{B}_0$ because of blocking large jobs can be placed separately into bins in $\mathcal{B}_0$.*

Proof. Since the number of non-zero variables $z_k^{(\ell)}$ in (12.11) is at most $rank(\widetilde{A}'_\delta) \leq D$, the area of large jobs that have to be placed via the dynamic program in $\mathcal{B}_1$ is at most

$$\begin{aligned}\sum_{\ell=L+1}^{L+P}\sum_k \left(z_k^{(\ell)} - \lceil z_k^{(\ell)} - 3C(\widetilde{A}_\delta)\rceil\right) size(\hat{C}_k^{(\ell)}) &\leq \sum_{\ell,k:z_k^{(\ell)}\neq 0} 3C(\widetilde{A}_\delta) size(\hat{C}_k^{(\ell)}) \\ &\leq 3\cdot D\cdot C(\widetilde{A}_\delta) c_{\max}(\mathcal{B}_1)\end{aligned}$$

This is also the area of medium jobs assigned to $\mathcal{B}_0$ by $(y_{j,\ell})$ variables that can not be placed because of additional large jobs assigned by the dynamic program. The number of this medium jobs can be bounded by

$$\begin{aligned}\frac{3\cdot D\cdot C(\widetilde{A}_\delta) c_{\max}(\mathcal{B}_1)}{\delta c_{min}(\mathcal{B}_1)} &\leq \frac{3\cdot D\cdot C(\widetilde{A}_\delta) c_{\max}(\mathcal{B}_1)}{\delta^2 c_{min}(\mathcal{B}_0)} \\ &\leq \frac{3\cdot D\cdot C(\widetilde{A}_\delta)}{\delta^2}\end{aligned}$$

Since $K \geq f(1/\delta) \geq \frac{3\cdot D\cdot C(\widetilde{A}_\delta)}{\delta^2}$ these jobs can be placed separately onto

machines in B_0 □

Then we can conclude

Theorem 12.17. *In Case 3 Algorithm 16 produces a schedule of length*

$$(1 + \mathcal{O}(\delta))\mathrm{OPT}(\mathcal{I})$$

in time

$$2^{\mathcal{O}(1/\delta \log(1/\delta) \log(C(\widetilde{A}_\delta))} poly(1/\delta, n).$$

Thus, we proved Theorem 10.1.

13. Concluding Remarks

As usual, there are still some open questions and areas that need improvement. To name some essentials let us consider the individual parts.
In Part I the 2-approximation for MSP can be applied to SPP with identical platforms, but the approximation ratio is not preserved as in case of a constant number of platforms the subroutine for rectangle packing with area maximization cannot be directly applied to select jobs. Here one has to find an alternative.

The running time of the $(2+\varepsilon)$-approximation for SPP in Part II is dominated by the running time of the dynamic program for selecting x-large and large jobs in $\mathcal{B}_0$. Therefore it is doubly exponential in $1/\varepsilon$. To obtain a better running time one idea is to use a linear program to preassign large and x-large jobs and by adequate rounding of the solution select the corresponding subset of large and x-large jobs. Furthermore, it would be interesting to consider in which cases the approximation ratio can be decreased to 2 since this would give a tight approximation algorithm for SPP.

In Part III a lower bound for max-gap(A_δ) where A_δ describes a scheduling problem remains open. It also remains open wether the upper bound for max-gap(A_δ) can be improved to $poly(1/\delta)$. This would considerably reduce the running time.

List of Figures

Bibliography

[1] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:55–66, 1998.

[2] A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis. Scheduling independent multiprocessor tasks. *Algorithmica*, 32(2):247–261, 2002.

[3] N. Bansal, A. Caprara, K. Jansen, L. Prädel, and M. Sviridenko. A structural lemma in 2-dimensional packing, and its implications on approximability. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC 2009), LNCS 5878*, pages 77–86, 2009.

[4] N. Bansal, X. Han, K. Iwama, M. Sviridenko, and G. Zhang. Harmonic algorithm for 3-dimensional strip packing problem. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (SODA 2007)*, pages 1197–1206, 2007.

[5] M. Bougeret, P. F. Dutot, K. Jansen, C. Otte, and D. Trystram. Approximation algorithms for multiple strip packing. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms (WAOA 2009), LNCS 5893*, pages 37–48, 2009.

[6] M. Bougeret, P. F. Dutot, K. Jansen, C. Otte, and D. Trystram. Approximating the non-contiguous multiple organization packing problem. In *Proceedings of the 6th IFIP International Conference on Theoretical Computer Science (TCS 2010)*, pages 316–327, 2010.

[7] M. Bougeret, P.-F. Dutot, K. Jansen, C. Otte, and D. Trystram. A fast 5/2-approximation algorithm for hierarchical scheduling. In *Proceed-*

ings of the 16th International Euro-Par Conference- Parallel Processing Part I (Euro-Par 2010), LNCS 6272, pages 157–167, 2010.

[8] M. Bougeret, P.-F. Dutot, K. Jansen, C. Robenek, and D. Trystram. Approximation algorithms for multiple strip packing and scheduling parallel jobs in platforms. *Discrete Mathematics, Algorithms and Applications*, 3(4):553–586, 2011.

[9] M. Bougeret, P.-F. Dutot, K. Jansen, C. Robenek, and D. Trystram. Scheduling jobs on heterogeneous platforms. In *Computing and Combinatorics - 17th Annual International Conference (COCOON 2011), LNCS 6842*, pages 271–283, 2011.

[10] M. Bougeret, P.-F. Dutot, K. Jansen, C. Robenek, and D. Trystram. Tight approximation for scheduling parallel jobs on identical clusters. In *26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPS Workshops 2012)*, pages 878–885, 2012.

[11] A. Caprara. Packing d-dimensional bins in d stages. *Mathematics of Operations Research*, 33:203–215, February 2008.

[12] Bo Chen. Tighter bound for multifit scheduling on uniform processors. *Discrete Applied Mathematics*, 31(3):227–260, 1991.

[13] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, 1978.

[14] E. G. Coffman Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

[15] W. Cook, A.M.H. Gerards, A. Schrijver, and É. Tardos. Sensitivity theorems in integer linear programming. *Mathematical Programming*, 34:251–264, 1986.

[16] F. Dietrich. *Approximation Algorithms for Linear Programs and Geometrically Constrained Packing Problems (Dissertation University of Kiel)*. 2009.

[17] J. Du and J. Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM Journal of Discrete Mathematics*, 2(4):473–487, 1989.

[18] A. Feldmann, J. Sgall, and S.-H. Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science*, 130(1):49–72, 1994.

[19] D. K. Friesen. Tighter bounds for the multifit processor scheduling algorithm. *SIAM Journal on Computing*, 13(1):170–181, 1984.

[20] D. K. Friesen. Tighter bounds for lpt scheduling on uniform processors. *SIAM Journal on Computing*, 16(3):554–560, 1987.

[21] M. R. Garey and R. L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, 1975.

[22] T. F. Gonzalez, O. H. Ibarra, and S. Sahni. Bounds for lpt schedules on uniform processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.

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

[24] M. D. Grigoriadis, L. G. Khachiyan, L. Porkolab, and J. Villavicencio. Approximate max-min resource sharing for structured concave optimization. *SIAM Journal on Optimization*, 11(4):1081–1091, 2001.

[25] L. A. Hall and D. B. Shmoys. Approximation schemes for constrained scheduling problems. In *30th Annual Symposium on Foundations of Computer Science (FOCS 1989)*, pages 134–139, 1989.

[26] R. Harren, K. Jansen, L. Prädel, and R. van Stee. A $(5/3+\epsilon)$-approximation for strip packing. In *Proceedings of the 12th International Symposium on Algorithms and Data Structures (WADS 2011)*, pages 475–487, 2011.

[27] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.

[28] D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

[29] D.S. Hochbaum. *Approximation Algorithms for NP-Hard Problems*, chapter 9: "'Various notions of approximations: good, better, best, and more"', pages 346–398. Prentice Hall, 1997.

[30] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23(2):317–327, 1976.

[31] K. Jansen. Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme. *Algorithmica*, 39(1):59–81, 2004.

[32] K. Jansen. *Efficient Approximation and Online Algorithms, LNCS 3484*, chapter "Approximation algorithms for min-max and max-min resource sharing problems and applications", pages 156–202. Springer, 2006.

[33] K. Jansen. An eptas for scheduling jobs on uniform processors: Using an milp relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, 24(2):457–485, 2010.

[34] K. Jansen. A $(3/2+\epsilon)$-approximation algorithm for scheduling moldable and non-moldable parallel tasks. In *24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2012)*, pages 224–235, 2012.

[35] K. Jansen. A fast approximation scheme for the multiple knapsack problem. In *38th International Conference on Current Trends in Theory and Practise of Computer Science (SOFSEM 2012)*, pages 313–324, 2012.

[36] K. Jansen and S. Kraft. An improved approximation scheme for variable-sized bin packing. In *37th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 529–541, 2012.

[37] K. Jansen and L. Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. *Algorithmica*, 32(3):507–520, 2002.

[38] K. Jansen and C. Robenek. Scheduling on uniform processors revisited. In *Proceedings of the 9th Workshop on Approximation and Online Algorithms (WAOA 2011), LNCS 7164*, pages 109–122, 2011.

[39] K. Jansen and R. Solis-Oba. Rectangle packing with one-dimensional resource augmentation. *Discrete Optimization*, 6(3):310–323, 2009.

[40] K. Jansen and R. Thöle. Approximation algorithms for scheduling parallel jobs. *SIAM Journal on Computing*, 39(8):3571–3615, 2010.

[41] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

[42] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (FOCS 1982)*, pages 312–320, 1982.

[43] H. Kellerer and U. Pferschy. Improved dynamic programming in connection with an fptas for the knapsack problem. *Journal of Combinatorial Optimization*, 8(1):5–11, 2004.

[44] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[45] C. Kenyon and E. Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25(4):645–656, 2000.

[46] M. Kunde. A multifit algorithm for uniform multiprocessor scheduling. In *Theoretical Computer Science*, pages 175–185, 1983.

[47] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

[48] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *Journal of the ACM*, 32(3):562–572, 1985.

[49] H.W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[50] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[51] J. Y.-T. Leung. On scheduling independent tasks with restricted execution times. *Operations Research*, 30(1):pp. 163–171, 1982.

[52] J. Y.-T. Leung. Bin packing with restricted piece sizes. *Information Processing Letters*, 31(3):145–149, 1989.

[53] L. Lovasz M. Grötschel and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1987.

[54] G. Mounie, C. Rapine, and D. Trystram. A 3/2-approximation algorithm for scheduling independent monotonic malleable tasks. *SIAM Journal on Computing*, 37(2):401–412, 2007.

[55] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995.

[56] J. Remy. Resource constrained scheduling on multiple machines. *Information Processing Letters*, 91(4):177–182, 2004.

[57] P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009.

[58] I. Schiermeyer. Reverse-fit: A 2-optimal algorithm for packing rectangles. In *Proceedings of the 2nd European Symposium on Algorithms (ESA 1994), LNCS 855*, pages 290–299, 1994.

[59] U. M. Schwarz. *Approximation Algorithms for Scheduling and Two-Dimensional Packing Problems (Dissertation University of Kiel)*. 2010. `http://eldiss.uni-kiel.de/macau/receive/dissertation_diss_00005147`.

[60] U. Schwiegelshohn, A. Tchernykh, and R. Yahyapour. Online scheduling in grids. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2008)*, pages 1–10, 2008.

[61] S. S. Seiden and R. van Stee. New bounds for multidimensional packing. *Algorithmica*, 36(3):261–293, 2003.

[62] A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.

[63] A. Tchernykh, J. Ramírez, A. Avetisyan, N. Kuzjurin, D. Grushin, and S. Zhuk. Two level job-scheduling strategies for a computational grid. In *Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics (PPAM 2005), LNCS 3911*, pages 774–781, 2005.

[64] J. Villavicencio and M. Grigoriadis. *Applied Mathematics and Parallel Computing*, chapter "Approximate Structured Optimization by Cyclic Block-Coordinate Descent", pages 359–371. Physica-Verlag HD, 1996.

[65] D. Ye, X. Han, and G. Zhang. Online multiple-strip packing. *Theoretical Computer Science*, 412(3):233–239, 2011.

[66] M. Yue. On the exact upper bound for the multifit processor scheduling algorithm. *Annals of Operations Research*, 24(1):233–259, 1990.

[67] S.N. Zhuk. Approximate algorithms to pack rectangles into several strips. *Discrete Mathematics and Applications*, 16(1):73–85, 2006.