APPROXIMATION ALGORITHMS FOR PACKING AND SCHEDULING PROBLEMS

Dissertation

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
(Dr. rer. nat.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

Ralf Thöle

Kiel

2008

1. Gutachter:	Prof. Dr. Klaus Jansen
2. Gutachter:	Prof. Roberto Solis-Oba
Datum der mündlichen Prüfung:	21. Oktober 2008

Acknowledgements

I have to thank many people for helping me in completing this thesis. Among these my special thanks are due to

- Prof. Klaus Jansen, my advisor, who supported me even while working on a completely different topic, and provided many ideas for research problems that led to this work,
- Prof. Roberto Solis-Oba, for his support, time, and hospitality at the University of Western Ontario (Oct 06 and Oct/Nov 07),
- Florian Diedrich, Rolf Harren, and Henning Thomas my co-authors,
- Ulrich Schwarz not only, but especially for his LaTeX-support,
- Ute Iaquinto, Parvaneh Karimi-Massouleh and Antje Sommerfeld for the administrative support,
- Christian Buck, Britta Kehden, and Jörg Thöle for proofreading, and
- Karina for her trust and support.

Ralf Thöle
Kiel, August 2008

Contents

1 Introduction

Algorithms for solving optimization problems play a major role in the industry. For example in the logistics industry, route plans have to be optimized according to various criteria. However, many natural optimization problems are hard to solve. That is, for many optimization problems no algorithms with running time polynomial in the size of the instance are known. Furthermore, supposed the widely accepted assumption P $\neq$ NP holds, it can be proved that many optimization problems do not allow algorithms that solve the problem optimally in polynomial time (NP-hard problems). One way of overcoming this dilemma is using approximation algorithms. These algorithms have a polynomial running time, but their solutions are in general not optimal but rather close to an optimum.

NP-hard optimization problems themselves cover a wide range of possibilities. Some of them allow polynomial time algorithms that find solutions which are arbitrarily close to the optimum, while other problems cannot be approximated in polynomial time within a constant ratio at all, unless P = NP.

In general, optimization problems allow efficient solving strategies if and only if they have an algorithmically relevant, combinatorial structure. Thus, the major challenge while developing efficient algorithms is to find these combinatorial, problem inherent structures and to develop adequate techniques to exploit these structures. Even if NP-hard problems have no structural properties that allow polynomial time algorithms to find optimal solutions, these problems might have properties which allow polynomial time algorithms to find nearly optimal solutions.

In the following, we will give a short outline of optimization problems and their classification. For a more formal introduction, we refer the reader to the books by Vazirani [67] or by Jansen & Margraf [29].

An optimization problem Π is either a minimization or a maximization problem. For every feasible instance I of Π, there exists a non-empty set of solutions. Furthermore, we have an objective function that assigns to each solution a non-negative value, the objective function value. If we have a solution for an NP-hard optimization problem, there exists a polynomial time algorithm that can check the feasibility and the correctness of the solution as well as calculate the objective function value of the solution. We call a feasible solution

optimal solution, if it obtains the optimal objective function value. By $\mathrm{OPT}_\Pi(I)$, we denote the objective function value of an optimal solution for an instance I of the optimization problem Π. We use OPT instead of $\mathrm{OPT}_\Pi(I)$ if Π and I are clear without ambiguity.

An approximation algorithm A for problem Π returns for each instance I in polynomial time a solution with objective function value $A(I)$ *close* to the optimal value, where *close* stands for the existence of some guaranteed constant; that is

$$A(I) \leq \alpha\,\mathrm{OPT} \qquad \text{minimization problem}$$

or

$$A(I) \geq \alpha\,\mathrm{OPT} \qquad \text{maximization problem}$$

for some constant $\alpha > 0$. We say A has *absolute* approximation ratio α. In particular for instances with large optimal value, *asymptotic* approximation algorithms are of interest; an algorithm A for minimization problem Π has asymptotic approximation ratio $\alpha > 0$ if

$$A(I) \leq \alpha\,\mathrm{OPT} + \beta$$

for some constant $\beta > 0$. Note that in general, the optimal objective function value is not known for a given instance. Thus, one of the main challenges is to find a good lower or upper bound for the optimum.

In the sense of approximability the best possible result is the existence of so called approximation schemes. An approximation scheme for a minimization problem Π is a family of algorithms A_ε such that for every $\varepsilon > 0$ and for every instance I,

$$A_\varepsilon(I) \leq (1+\varepsilon)\,\mathrm{OPT} + \beta_\varepsilon,$$

where $\beta_\varepsilon \geq 0$ is a constant that may depend on ε.

If $\beta_\varepsilon = 0$ and the running time is polynomially bounded in the size of the instance, we call this family of algorithms *polynomial time approximation scheme* (PTAS). If additionally the running time is polynomially bounded in $1/\varepsilon$, we call it *fully polynomial time approximation scheme* (FPTAS). If $\beta_\varepsilon \neq 0$, we call the family *asymptotic*, i.e. *asymptotic polynomial time approximation scheme* (APTAS) or *asymptotic fully polynomial time approximation scheme* (AFPTAS), respectively. Although problems for which a PTAS exists can be approximated in polynomial time with arbitrary accuracy ε, the running time might depend exponentially on ε. Obviously, a running time in $\mathcal{O}(n^{\varepsilon^{-1}!})$ is far from practical for small ε (high accuracy). Moreover, for some of these problems it can be shown that a FPTAS does not exist,

unless P = NP. For these problems, an interesting question is if they allow *efficient polynomial time approximation schemes* (EPTAS). The running time for an EPTAS is required to be $\mathcal{O}(f(\varepsilon^{-1})n^c)$ for a function f and a constant c independent of ε.

The Problems

The main subject of this thesis is approximation algorithms for (geometric) packing and scheduling problems. At first glance, these problems seem to be unrelated. On the one hand we have the geometric problem of arranging objects in a non-overlapping way, on the other hand we have to assign jobs at a specific time to one machine (sequential jobs) or a set of machines (parallel jobs). However, if we look more closely at the solutions for both problems, they are in fact very similar. But before we elaborate on this, let us give a short introduction to the problems.

Geometric Packing Problems

Geometric packing problems occur in many real world applications, like *cutting stock*, *selecting boxes to be transported in a container*, or *print layout*. We distinguish three basic scenarios: *bin packing*, *strip packing*, and *knapsack*.

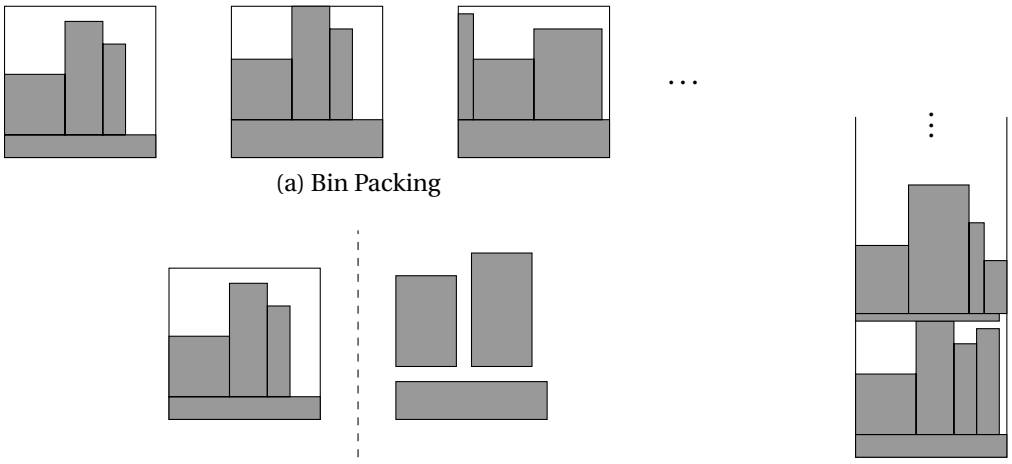
In each of these scenarios, we are given a list of geometric objects with sizes.

Bin Packing In the bin packing scenario, we have additionally an infinite number of bins with fixed sizes. The objective is to pack all objects into a minimal number of bins (see Figure 1.1a).

Strip Packing In the strip packing scenario, we have additionally a strip with fixed width but infinite height. The objective is to pack all objects into the strip such that the total height of all packed objects is minimized (see Figure 1.1c).

Knapsack In the knapsack scenario, additionally each object has a profit and we have one target container of fixed size. The objective is to pack a subset of objects with maximal profit into the target container (see Figure 1.1b).

In Chapter 2 we present approximation algorithms for the three-dimensional orthogonal knapsack problem. In this setting, a list of boxes with sizes and positive profits are given. Furthermore, a dedicated box (container) with fixed size is given. The objective is to find a non-overlapping axis-parallel packing of a sublist of the boxes into the container, such that



(a) Bin Packing

(b) Knapsack

(c) Strip Packing

Figure 1.1: Three basic scenarios

the sum of the profits of all selected boxes is maximal. As subroutine we use approximation algorithms for the (two-dimensional) strip packing problem (see Sections 2.2, 2.5).

Scheduling Problems

In general, scheduling is the problem of assigning a set of jobs to a set of resources subject to a set of constraints. Obviously, this ambiguous definition allows a huge conglomeration of different models. Examples of scheduling constraints include deadlines (e.g. job J_i must be completed by time t), resource capacities (e.g. all processors share a limited memory such that only a constant number of jobs can be executed simultaneously), precedence constraints on the order of jobs (e.g. a piece must be polished before it is painted), and priorities on jobs (e.g. finish job J_i as soon as possible while meeting the other deadlines).

In this thesis, we focus on variants of the non-preemptive, parallel job scheduling problem. In this setting, we are given a number of available machines and a list of jobs; furthermore, for each job an execution time and the number of required machines is given. A schedule is an assignment of each job to a starting time and to a subset of machines. The objective in our setting is to find a feasible schedule for all jobs, such that the completion time of the latest job is minimized. We call a schedule feasible if at each time at most one job is assigned to each machine. For example in Figure 1.2 we have a schedule with 5 machines $(P_1,\dots,P_5)$ and 6 jobs $(J_1,\dots,J_6)$.

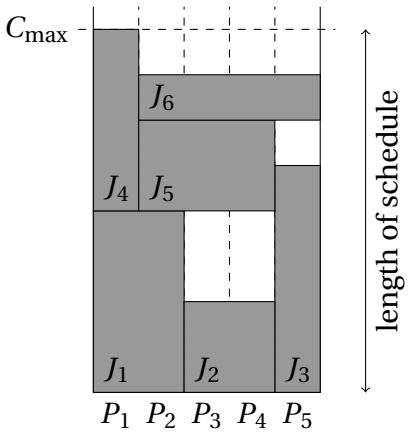


Figure 1.2: A simple schedule

Common Ground

Most packing problems can be viewed as special cases of scheduling problems. For example the (classical) strip packing problem corresponds to the scheduling problem where each job has to be allotted to contiguous machines. The three-dimensional strip packing problem corresponds to scheduling parallel jobs on a (two-dimensional) mesh. In these cases, the infinite dimension of the strip corresponds to the timeline in the scheduling problem. The three-dimensional orthogonal knapsack problem corresponds to scheduling parallel jobs on a (two-dimensional) mesh if all jobs have a given value and the objective is to find the most profitable subset of jobs that can be executed before a fixed deadline. Scheduling jobs on meshes or hypercubes is not only of theoretical interest, these special cases are motivated by specific network topologies. For example scheduling jobs on a connected mesh might be desired to maintain a physical proximity of machines allotted to a job in order to minimize communication delays.

Outline of the Thesis

This thesis consists of two main chapters. In Chapter 2, we present approximation algorithms for the three-dimensional orthogonal knapsack problem and in Chapter 3, we present approximation algorithms for variants of the non-preemptive parallel job scheduling problem. Although these topics are related, each chapter is intended to be self-contained. In particular, a reader interested in only one of the topics should have no problem reading only the corresponding part.

Parts of this thesis have been published in [15, 16, 36, 37].

Chapter 2

In Chapter 2, we present approximation algorithms for the three-dimensional orthogonal knapsack problem (OKP-3).

We start in Section 2.1 by giving a short introduction, where we summarize known, related and new results and mention some applications. In Section 2.2 we introduce a strip packing subroutine and, based on this subroutine, a $(9+\varepsilon)$-approximation algorithm for OKP-3. Furthermore, we show that our algorithm can be used to solve the three-dimensional strip packing problem with *absolute* approximation ratio 6 (improving upon the best known algorithm with absolute approximation ratio $45/4 = 11.25$ which follows from [51]). In Section 2.3, we use a stronger relaxation and a refinement of the strip packing algorithm to get a $(8+\varepsilon)$-approximation. In Section 2.4, we generalize a packability criterion based on the result by Steinberg [64] to the three-dimensional case. A $(7+\varepsilon)$-approximation algorithm is presented in Section 2.6. To achieve this ratio we exploit basically two observations. First, small boxes can be packed more efficiently, and second, only a constant number of large boxes may occur in any feasible solution. Thus, it is possible to generate an optimal solution for the large boxes by enumeration. Unfortunately, due to this large enumeration, the resulting algorithm is not very practical. However, it is the algorithm with the best approximation ratio (as far as we know). We discuss the cases where rotations are allowed in Section 2.7; we show that the $(7+\varepsilon)$-approximation can be modified to yield a $(6+\varepsilon)$-approximation, if 90° rotations around the z-axis are allowed (z-oriented OKP-3), and a $(5+\varepsilon)$-approximation, if 90° rotations around all axes are allowed. We conclude this chapter with open problems in Section 2.8.

Chapter 3

In Chapter 3, we present approximation algorithms for variants of the non-preemptive parallel job scheduling problem in which the number of available machines is polynomially bounded in the number of jobs.

We start this chapter in Section 3.1 by giving a short introduction, where we summarize known, related and new results. We outline the scheduling algorithms in Section 3.2. In Section 3.3, we present an algorithm for packing rectangles into a constant number of bins, which is essentially an extension of the algorithm by Kenyon & Rémila [46]. For the case that jobs are allotted to contiguous machines, we present an $(1.5+\varepsilon)$-approximation algorithm in Section 3.4. The basic ideas for this algorithm are similar to the ideas used in [34]. In order to achieve the 1.5 ratio, however, we have to take special care of jobs with execution

time greater than $\frac{1}{2}$OPT. In Section 3.5, we present an $(1+\varepsilon)$-approximation algorithm for the non-contiguous case. We generalize both scheduling algorithms to the malleable case in Section 3.6; to be more specific, we present a pre-processing step that allows us to assign a number of machines to each job. Subsequent application of the algorithms for the corresponding non-malleable cases yields an $(1.5+\varepsilon)$-approximation algorithm for the contiguous and a PTAS for the non-contiguous case. We conclude this chapter with open problems in Section 3.7.

2 The Three-Dimensional Orthogonal Knapsack Problem

2.1 Introduction

Given a list $L = \{R_1, \ldots, R_n\}$ of boxes with sizes $R_i = (x_i, y_i, z_i)$ and positive profits p_i for each $i \in \{1, \ldots, n\}$ and a dedicated box $Q = (a, b, c)$, we study *feasible* packings of sublists of L into Q. A packing is *feasible* if all boxes are packed non-overlapping and axis-parallel into the dedicated box. For simplicity, we call Q a *bin*. We wish to select a sublist which permits a packing and maximizes the profit (see Figure 2.1). This problem will be called the *three-dimensional orthogonal knapsack problem* or *OKP-3* for short and we denote the optimal profit by OPT. It is a natural generalization of the knapsack problem (KP) which is known to be NP-hard. This makes an exact algorithm with a polynomial worst-case runtime bound impossible unless P = NP holds. For this reason, we concentrate on approximation algorithms; we refer the reader to [67] for a detailed description of the approach and common notions. Without loss of generality we assume $a = b = c = 1$ and that each $R_i \in L$ can be packed by otherwise removing infeasible boxes and scaling in $\mathcal{O}(n)$ time. Note that the scaling is only possible in the non-rotational case. In the rotational case the fixed size of the bin is an explicit assumption.

The research on geometrically constrained two- and three-dimensional packing problems follows three main directions:

In *strip packing* the target area is a strip of infinite height; the objective is to minimize the height of the packing. One of the first results for this problem was given by Coffman

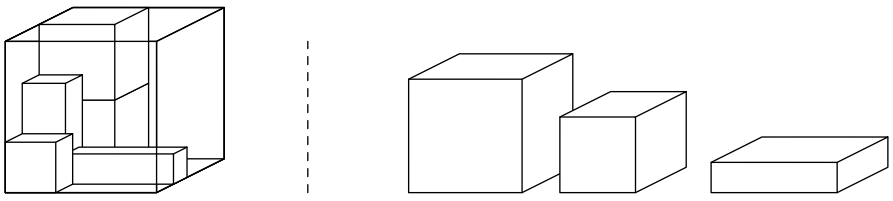


Figure 2.1: The three-dimensional knapsack problem

et al. [12]. They proved that the level-based algorithms NFDH (Next Fit Decreasing Height) and FFDH (First Fit Decreasing Height) algorithms have an approximation ratio of 3 and 2.7 respectively. Based on these level-based algorithms Sleator [63] presented an algorithm with approximation ratio $3/2$. Baker et al. [3] obtained an asymptotic approximation ratio of $5/4$. The best known absolute approximation ratio of 2 was obtained independently with different techniques by Schiermeyer [61] and Steinberg [64]. Coffman et al. [12] also analyzed the asymptotic performance of NFDH and FFDH, which is $2 \cdot \mathrm{OPT} + h_{\max}$ and $1.7 \cdot \mathrm{OPT} + h_{\max}$ respectively, where OPT is the height of an optimal solution and $h_{\max}$ denotes the height of the tallest rectangle. Kenyon & Rémila [46] found an AFPTAS (asymptotic fully polynomial time approximation scheme) for the problem; they obtained an additive error of $\mathcal{O}(1/\epsilon^2)$. This additive error was later improved by Jansen & Solis-Oba in [34], where the authors presented an APTAS (asymptotic polynomial time approximation scheme) with additive error 1. For the case that 90° rotations are allowed, Miyazawa and Wakabayashi [56] presented an algorithm with asymptotic performance bound 1.613. A simple algorithm with an upper bound of $3/2$ was presented by Epstein and van Stee [21]. An AFPTAS for this problem was found by Jansen and van Stee [38]. For the three-dimensional case, research has focused mainly on the *asymptotic* approximation ratio, where Miyazawa & Wakabayashi [57] found an algorithm with asymptotic approximation ratio between 2.5 and 2.67. An asymptotic ratio of $2+\epsilon$ was obtained by Jansen & Solis-Oba [33]; this was improved to 1.691 by Bansal et al. [6]. The best known *absolute* approximation ratio of $45/4$ follows from an asymptotic approximation ratio of $13/4$ by Li & Cheng [51]. The same authors [52] presented an *on-line* algorithm with asymptotic performance ratio arbitrarily close to 2.89.

In *bin packing* the objective is to minimize the number of (identical) bins. A classical algorithm for the one-dimensional version of this problem is the *first fit decreasing* (FFD) algorithm. For this algorithm, Johnson [44] proved a bound of $11/9\,\mathrm{OPT}+4$. The additive constant was improved to 3 by Backer [2]. Li & Yue [53] proved the bound of $11/9\,\mathrm{OPT}+7/9$. The study of FFD was settled recently by Dósa [17] who proved that the bound $11/9\,\mathrm{OPT}+6/9$ is tight. An APTAS was presented by Fernandez de la Vega & Lueker [24]. For the two-dimensional case, an asymptotic approximation ratio of 1.691 was obtained by Caprara [10]; this result was improved to an asymptotic approximation ratio of 1.525 by Bansal et al. [4]. Furthermore, Bansal et al. [5] proved that two-dimensional bin packing does not admit an APTAS (asymptotic polynomial time approximation scheme) and no FPTAS (fully polynomial time approximation scheme) if P $\neq$ NP. They also presented an APTAS for packing d-dimensional cubes into the minimum number of unit cubes in the same paper. For the case that 90° rotations are allowed, Miyazawa and Wakabayashi [56] presented an algorithm with asymptotic performance bound 2.64. This result was improved

to 2.45 by Epstein [20] and to 2.25 by Epstein and van Stee [21]. Jansen and van Stee [38] presented an asymptotic $(2+\epsilon)$-algorithm for this problem.

In the *knapsack* scenario, the number of bins is a fixed constant [14], usually 1. Classical one-dimensional knapsack problems are relatively well understood, see [45, 55] for surveys. For the two-dimensional case, Jansen & Zhang [40] obtained an approximation ratio of $2+\epsilon$. For the special case of packing squares with unit profits into a rectangle Jansen & Zhang [39] proved the existence of an AFPTAS and a PTAS. Recently, Jansen & Solis-Oba [35] proved the existence of an PTAS for the square packing problem with arbitrary profits. This is the best possible algorithm since the square packing problem is strongly NP-hard [50]. For the three-dimensional case, Harren [27] obtained a ratio of $9/8+\epsilon$ for the special case of packing cubes and Chlebík and Chlebíková proved the APX-completeness of the general case [11].

Although these problems are closely related, results cannot be transferred directly. One main contrast between bin packing and strip packing on the one hand and knapsack on the other hand is that in the first setting all boxes of the instance must be packed but in the latter a feasible selection of items must be found.

A *cutting stock* application is cutting blocks with given profits from larger pieces of material to maximize the profit; another application is the problem of selecting boxes to be transported in a container. Besides these, the problem is motivated from multiprocessor scheduling on grid topology. In this perspective, for a time slice of fixed duration, a set of jobs to be executed must be chosen and each job requires a subgrid of prespecified rectangular shape. For a special case of this application, Ye & Zhang [68] presented an on-line algorithm; see [23] for a study of similar problems.

2.1.1 New Results

Our contribution is a fast and simple $(9+\epsilon)$-approximation algorithm based on strip packing which can be refined to an $(8+\epsilon)$-approximation algorithm. Both of these algorithms have practical running times. With more sophisticated techniques we obtain a $(7+\epsilon)$-approximation algorithm. Hoewer, the running time of this algorithm is not practical due to a large enumeration step. Furthermore, we show that the approximation ratios of these algorithms are tight. Note that the tightness results are based on the work by Henning Thomas [65].

We also study the case where rotation by 90° either around the z-axis or around all axes is permitted, where we improve upon the approximation ratios of the algorithms presented for OKP-3. We derive approximation ratios of $(6+\epsilon)$ for the former and $(5+\epsilon)$ for the latter case. Finally, our methods yield a three-dimensional generalization of a packability crite-

rion and a strip packing algorithm with absolute approximation ratio 6.

2.1.2 Structure

This chapter is organized as follows. In Section 2.2, we present a fast algorithm for non-rotational packing. Furthermore, we show that the algorithm can be used to improve a known result on three-dimensional strip packing. We present a refinement of the first algorithm in Section 2.3. In Section 2.4, we generalize a two-dimensional packability criterion for boxes and improve a known result on strip packing in Section 2.5 before turning back to the knapsack problem in Section 2.6, where we obtain a better yet more costly algorithm. Finally, we discuss the cases of rotational packing in Section 2.7, and conclude with open problems in Section 2.8.

2.2 An Algorithm Based on Strip Packing

We approximately solve a relaxation of OKP-3 by selecting $L' \subseteq L$, which is at least near-optimal and has a total volume of at most 1. This relaxed solution is partitioned into 9 sublists. For each of these, a packing into the bin will be generated. Out of these one with maximum profit is chosen, resulting in a $(9+\epsilon)$-approximation algorithm. More precisely, L' will be packed into a strip $[0,1]\times[0,1]\times[0,\infty)$ by a level-oriented algorithm, i.e. an algorithm which packs all boxes into disjoint levels and stacks these levels on top of one another into the strip. We partition the strip into packings of sublists of L' and among these, we return one with maximum profit.

For each box R_i the rectangle (x_i, y_i) is called the *base rectangle* of R_i, denoted as $\mathrm{br}(R_i)$. Such a rectangle (x_i, y_i) is called

$$\begin{aligned}
\textit{big} &:\Leftrightarrow x_i \in (1/2, 1] \quad \text{and} \quad y_i \in (1/2, 1],\\
\textit{long} &:\Leftrightarrow x_i \in (1/2, 1] \quad \text{and} \quad y_i \in (0, 1/2],\\
\textit{wide} &:\Leftrightarrow x_i \in (0, 1/2] \quad \text{and} \quad y_i \in (1/2, 1],\\
\textit{small} &:\Leftrightarrow x_i \in (0, 1/2] \quad \text{and} \quad y_i \in (0, 1/2].
\end{aligned}$$

For each list L of boxes use $\mathrm{Vol}(L)$ to denote the total volume of L and for each list L of rectangles use $\mathrm{A}(L)$ to denote the total area of L. Furthermore, $\mathrm{P}(L)$ denotes the total profit of L. Finally, for each list L of boxes use $\mathrm{h}(L)$ to denote the height of a packing of L where the packing itself will be clear from the context. We use the following theorem from [40] which is a refinement of the main result from [64].

Theorem 2.2.1 ([40])**.** *Let L be a list of n rectangles such that $A(L) \leq 1/2$ holds and no long rectangles or no wide rectangles occur in L, i.e. there might be either long or wide rectangles but not both. Then L permits a feasible packing into the unit square which can be generated in time $\mathcal{O}(n \log^2 n / \log\log n)$.*

Note that rectangles which are long or wide are not big. It is therefore possible in Theorem 2.2.1 that there is a big rectangle in L.

First, we present a level-based algorithm for the three-dimensional strip packing problem. In the analysis of this algorithm, we use Theorem 2.2.1 to obtain an *area guarantee* for each but the last level, improving a result from [51]. Then, we construct a partition of the generated strip.

Algorithm A.

1. Partition L into two sublists $L_1 := \{R_i \in L \mid \mathrm{br}(R_i) \text{ is long}\}$ and $L_2 := L \setminus L_1$. Without loss of generality let $L_1 = \{R_1, \ldots, R_m\}$ and $L_2 = \{R_{m+1}, \ldots, R_n\}$.

2. Generate the packing for L_1 as follows.

   2.1. Find the boxes R_i in L_1 for which the area of $\mathrm{br}(R_i)$ is greater than $1/4$ which are $R_{p+1}, \ldots, R_m$ without loss of generality Stack these on top of one another in direction z, each on its own level.

   2.2. Sort the remaining boxes $R_1, \ldots, R_p$ in non-increasing order of z_i, resulting in a list L_1'.

   2.3. Partition L_1' into consecutive sublists $L_1'', \ldots, L_v''$ where the total base area of each sublist is as close to $1/2$ as possible but not greater. Pack each of these sublists on a level by itself using Theorem 2.2.1. Stack all of these levels on top of one another in direction z.

3. Generate the packing for L_2 in a similar way as for L_1 by Theorem 2.2.1. The resulting steps are called Steps 3.1 – 3.3.

4. Concatenate the packings of L_1 and L_2 to obtain a packing of L.

Theorem 2.2.2. *For each list L of n boxes Algorithm A generates a packing of height at most $4\,\mathrm{Vol}(L) + Z_1 + Z_2$ where Z_1 and Z_2 are the heights of the first levels generated in Steps 2.3 and 3.3. The construction can be carried out in time $\mathcal{O}(n \log^2 n / \log\log n)$.*

The currently best known *asymptotic* approximation ratio for the three-dimensional strip packing problem is 1.691 by Bansal et al. [6]. Our result improves the best known *absolute* approximation ratio of $45/4 = 11.25$ (which follows from [51]) to 6.

Since $\mathrm{OPT}(L) \geq \max\{Z_1, Z_2, \mathrm{Vol}(L)\}$, we easily derive the following corollary:

Corollary 2.2.3. *The absolute approximation ratio of algorithm A is at most* 6.

Proof (Theorem 2.2.2). Consider Step 2.3 which generates levels $L''_1, \ldots, L''_v$. Let h_i denote the height of level L''_i for each $i \in \{1, \ldots, v\}$. Each box R_i for $i \in \{p+1, \ldots, m\}$ processed in Step 2.1 forms a level itself, so the total height for the packing of L_1 is

$$\mathrm{h}(L_1) := \sum_{i=p+1}^{m} z_i + \sum_{i=1}^{v} h_i \tag{2.1}$$

and we have $\mathrm{Vol}(L) = \mathrm{Vol}(L_1) + \mathrm{Vol}(L_2)$. Furthermore,

$$\mathrm{Vol}(L_1) = \sum_{i=p+1}^{m} x_i y_i z_i + \sum_{i=1}^{v} \mathrm{Vol}(L''_i)$$

holds. In Step 2.1 we assert that $x_i y_i > 1/4$ for $i \in \{p+1, \ldots, m\}$, so we get

$$\sum_{i=p+1}^{m} x_i y_i z_i > \frac{1}{4} \sum_{i=p+1}^{m} z_i. \tag{2.2}$$

For each $i \in \{1, \ldots, v-1\}$ the total base area of the boxes in L''_i is larger than $1/4$ since otherwise the next box would have also been put in L'_i in Step 2.3. As the boxes are sorted in non-increasing order of height, and thus $z_j \geq h_{i+1}$ for $R_j \in L''_i$, we get that

$$\sum_{R_j \in L''_i} x_j y_j > \frac{1}{4} \tag{2.3}$$

holds, hence

$$\mathrm{Vol}(L''_i) \geq \frac{1}{4} h_{i+1} \tag{2.4}$$

for each $i \in \{1, \ldots, v-1\}$. Combining (2.2) and (2.4) results in

$$\mathrm{Vol}(L_1) > \frac{1}{4} \sum_{i=p+1}^{m} z_i + \frac{1}{4} \sum_{i=2}^{v} h_i,$$

which together with (2.1) implies that

$$h(L_1) < 4\,\mathrm{Vol}(L_1) + h_1 \leq 4\,\mathrm{Vol}(L_1) + Z_1 \tag{2.5}$$

holds. For Step 3 we get

$$h(L_2) < 4\,\mathrm{Vol}(L_2) + Z_2 \tag{2.6}$$

with a very similar analysis. From (2.5) and (2.6) we conclude that the height of our packing of L is bounded from above by $h(L_1) + h(L_2) < 4\,\mathrm{Vol}(L) + Z_1 + Z_2$. The running time is dominated by the application of Theorem 2.2.1 and thus bounded by $\mathcal{O}(n \log^2 n / \log\log n)$. □

The second part of the algorithm is a partition of the arrangement generated by algorithm A into at most 9 bins (see Figure 2.2).

Algorithm B.

1. Set $\delta := \epsilon/(9+\epsilon)$. Use an FPTAS for KP from [45, 47] to select $L' \subseteq L$ such that $\mathrm{Vol}(L') \leq 1$ and $P(L') \geq (1-\delta)\,\mathrm{OPT}$ holds, where OPT denotes the optimum of the generated KP instance.

2. Use Algorithm A to generate a packing of L' into the strip but separate the first levels generated in Steps 2.3 and 3.3. Pack these into a bin each.

3. By Theorem 2.2.2 the remaining strip has a height of at most $4\,\mathrm{Vol}(L') \leq 4$. Consider the three cutting unit squares $[0,1] \times [0,1] \times \{i\}$ for $i \in \{1,2,3\}$. Generate a partition of the region $[0,1] \times [0,1] \times [0,4]$ into 7 subsets, namely 4 subsets which are each positioned in the regions $[0,1] \times [0,1] \times [i-1,i]$ for $i \in \{1,\ldots,4\}$ but not intersecting any of the unit squares and 3 subsets of boxes which each intersect with one of the three cutting unit squares.

4. Out of the 9 sets generated in Steps 2 and 3, return one with maximum profit.

Each set generated in Steps 2 and 3 permits a feasible packing into the unit cube which is available as a byproduct of Algorithm A. L' is partitioned into at most 9 subsets by Algorithm B, as illustrated in Figure 2.2.

Theorem 2.2.4. *Algorithm B is a $(9+\epsilon)$-approximation algorithm for OKP-3 with running time $\mathcal{O}(T_{KP}(n,\epsilon) + n \log^2 n / \log\log n)$, where $T_{KP}(n,\epsilon)$ is the running time of the FPTAS used for solving KP. Furthermore, this bound is tight.*

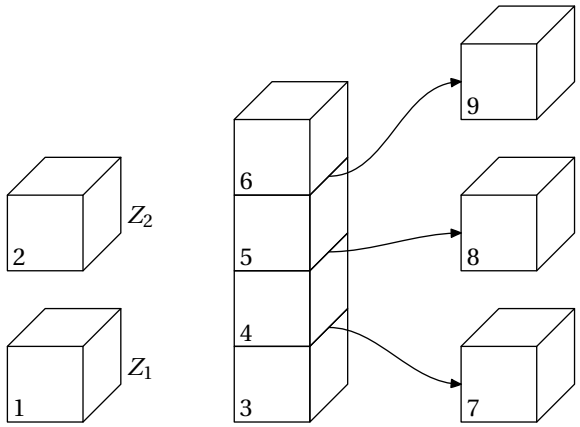


Figure 2.2: At most 9 bins are generated by Algorithm B

Proof. Clearly $9+\epsilon$ is an upper bound for the ratio since

$$\frac{1-\delta}{9} = \frac{1-\frac{\epsilon}{9+\epsilon}}{9} = \frac{\frac{9}{9+\epsilon}}{9} = \frac{1}{9+\epsilon} \tag{2.7}$$

holds. The running time is dominated by solving the knapsack instance and by Algorithm A. Note that $\mathcal{O}(n\min\{\log n, \log(1/\epsilon)\} + 1/\epsilon^2 \log(1/\epsilon)\min\{n, 1/\epsilon \log(1/\epsilon)\})$ is the best running time for an FPTAS for the knapsack problem in [45].

For the following instance this bound can be attained. We have 10 boxes

$$\begin{aligned} R_1 &:= (1/2, 1/2, 2/15) & R_2 &:= (1, 1/4, 2/15) \\ R_3 &:= (1, 2/7, 3/4) & R_4 &:= \cdots := R_7 := (1, 2/7, 1/2) \\ R_8 &:= (1, 2/7, 1/4 + 2/15) & R_9 &:= (1, 2/7, 2/15) \\ R_{10} &:= (1,1,1). \end{aligned}$$

Furthermore, $p_1 := \ldots := p_9 := 1/(9+\epsilon)$ and $p_{10} := 1$. Let $S_1 := \{R_1, \ldots, R_9\}$ and $S_2 := \{R_{10}\}$. Then the total volume of S_1 is

$$\begin{aligned} \mathrm{Vol}(S_1) &= \sum_{i=1}^{9} x_i y_i z_i \\ &= \frac{1}{30} + \frac{1}{30} + \frac{3}{14} + 4 \cdot \frac{1}{7} + \left(\frac{1}{14} + \frac{4}{7 \cdot 15}\right) + \frac{4}{7 \cdot 15} \\ &= \frac{1}{15} + \frac{6}{7} + \frac{8}{7 \cdot 15} = \frac{1}{15}\left(1 + \frac{8}{7}\right) + \frac{6}{7} \\ &= \frac{1}{15}\left(\frac{15}{7}\right) + \frac{6}{7} = 1, \end{aligned}$$

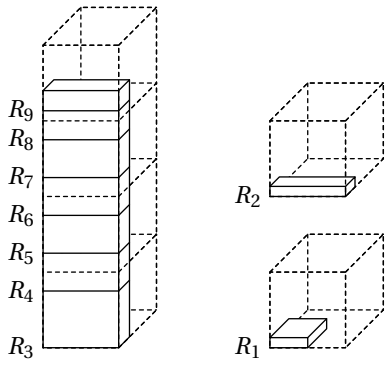


Figure 2.3: Packing of S_1 with Algorithm B

and the total profit of S_1 is

$$P(S_1) = 9 \cdot \frac{1}{(9+\epsilon)} \overset{(2.7)}{=} 1 - \delta.$$

It is clear that S_2 is an optimal solution. Since $\mathrm{Vol}(S_1) = 1$ and $P(S_1) = 1-\delta$, S_1 may be selected in Step 1 of Algorithm B. Application of Algorithm B and assuming that the boxes are stacked in increasing order of index in Step 2.1 of Algorithm A yields 9 bins each containing an item with profit $1/(9+\epsilon)$ (see Figure 2.3). □

Note that only the subset that is returned needs to be packed level-wise using the algorithm from Theorem 2.2.1 while the discarded subsets need not be arranged. Algorithm B can be used to solve the special cases where we wish to maximize the number of selected boxes or the volume by setting $p_i := 1$ or $p_i := x_i y_i z_i$ for each $i \in \{1,\dots,n\}$. This also holds for the other algorithms we present.

In [45, 47], approximation algorithms for various knapsack problems are found. Using these, Algorithm B can be generalized by replacing the KP solver in Step 1, yielding algorithms for *unbounded* OKP-3 and *multiple-choice* OKP-3. In the unbounded OKP-3 case an unbounded number of copies of each box might be selected, whereas in the multiple-choice OKP-3 case, all boxes are partitioned into classes and from each class exactly one box must be chosen; see [45, 47] for notions and details. Algorithm B can be modified to yield a ratio of 18 with a much better running time by using a 2-approximation algorithm for classical KP from [45, 47], thus replacing $T_{\mathrm{KP}}(n,\epsilon)$ by $\mathcal{O}(n)$ in Theorem 2.2.4.

2.3 A Refined Construction

In this section, we show how to refine Algorithm B to yield an approximation ratio of $(8+\epsilon)$. We identify two possible improvements on Algorithm A. First, by increasing the area

guarantee and second, by decreasing the heights Z_1 and Z_2 of the additional strip packing levels.

In Algorithm A and the proof of Theorem 2.2.2, the area bound $1/2$ from Theorem 2.2.1 was used. We separated boxes with base area greater than $1/4$, resulting in the area guarantee of $1/2 - 1/4 = 1/4$ for each level generated in Steps 2.2 and 2.3 except the last ones. By improving the area guarantee, we will improve the height bound of the strip.

So far we arbitrarily chose direction z to be the axis for level generation, but any direction $d \in \{x, y, z\}$ will do. This has the advantage that packing in a direction where all boxes are short, i.e. at most $1/2$, implies that the heights of the additional levels are bound by $1/2$.

Let us introduce the notion of big boxes in certain directions; a box R_i is called d-*big* $:\Leftrightarrow$ $d_i \in (1/2, 1]$ for any direction $d \in \{x, y, z\}$ and we use X, Y and Z to denote the set of boxes that are d-big for the corresponding direction. Any box that is d-big for every direction $d \in \{x, y, z\}$ will be called a *big* box. Finally, a box R_i is called *small* $:\Leftrightarrow$ $x_i \in (0, 1/2]$ and $y_i \in (0, 1/2]$ and $z_i \in (0, 1/2]$.

The remainder of the section is organized as follows. First, we give a refined version of Algorithm A that is restricted to packing small boxes. Second, we show how to partition the boxes which are not small into three sets according to the big direction. Third, we give the overall algorithm, which is based on the partition into d-big and small boxes.

The following algorithm is applied only on *small* items.

Algorithm C.

1. Find the boxes R_i in L for which the area of $\mathrm{br}(R_i)$ is greater than $1/10$, and they are $R_1, \ldots, R_m$ without loss of generality. Sort these in non-increasing order of z_i, resulting in a list L_1. Arrange these in groups of 4 boxes each, except for the last group. Each group can be put on a separate level by placing the boxes into the corners of the level. Stack these levels on top of one another in direction z.

2. Sort the remaining boxes $R_{m+1}, \ldots, R_n$ in non-increasing order of z_i, resulting in a list L_2.

3. Partition L_2 into consecutive sublists $L_1'', \ldots, L_v''$ where the total base area of each sublist is as close to $1/2$ as possible but not greater. Pack each of these sublists on a level by itself using Theorem 2.2.1. Stack all of these levels on top of one another in direction z.

4. Concatenate the packings of L_1 and L_2 to obtain a packing of L.

Note that we formed two groups and obtain an area guarantee of $2/5 = 1/2 - 1/10$ for each layer except the last ones generated in Steps 1 and 3. To avoid confusion we point out that the area guarantee does not hold for the *last* generated layers, while the summands Z_1 and Z_2 in Theorem 2.2.2 are the heights of the respective *first* layers. Similar to the proof of Theorem 2.2.2, we obtain the following results using the area guarantee of $2/5$.

Theorem 2.3.1. *For each list L of n small boxes, Algorithm C generates a feasible packing of height at most $5/2\,\mathrm{Vol}(L) + Z_1 + Z_2$ where $Z_1 \leq 1/2$ and $Z_2 \leq 1/2$ are the heights of the first levels generated in Steps 1 and 3. The construction can be carried out in time $\mathcal{O}(n \log^2 n / \log\log n)$.*

Lemma 2.3.2. *Each list L of n small boxes with $\mathrm{Vol}(L) \leq 1$ permits a feasible packing into at most 5 bins. The construction can be carried out in time $\mathcal{O}(n \log^2 n / \log\log n)$; the bound of 5 is tight for the used construction.*

Proof. Use Algorithm C to arrange L in a strip, but separate the first levels generated in Steps 1 and 3. Since L contains only small boxes, these two levels can be packed together into a bin. By Theorem 2.3.1, the remaining strip has a height of at most $5/2$. Consider the two cutting unit squares $[0,1] \times [0,1] \times \{i\}$ for $i \in \{1,2\}$. Generate a partition of the region $[0,1] \times [0,1] \times [0,5/2]$ into 5 subsets, namely 3 subsets which are each positioned in the regions $[0,1] \times [0,1] \times [i-1,i]$ for $i \in \{1,2\}$ as well as the region $[0,1] \times [0,1] \times [2,5/2]$ but not intersecting any of the unit squares, and furthermore 2 subsets of boxes which each intersect with one of the two cutting unit squares. The first three sets can be packed into one bin each. Since L contains only small boxes, the last two sets can be arranged together into one additional bin by aligning them at the top and bottom of the bin, respectively. We have at most 5 bins (see Figure 2.4). The running time is dominated by Algorithm C and thus bounded by $\mathcal{O}(n \log^2 n / \log\log n)$.

To show the tightness of the bound let $\gamma := 1/500$ and consider the instance L consisting of

$$\begin{aligned} R_1 := \cdots := R_{29} &:= (1/2, 1/5 + \gamma, 1/3 + \gamma) \\ R_{30} &:= (\gamma, \gamma, 1/2) \\ R_{31} := \cdots := R_{33} &:= (1/2, 1/5, \gamma) \\ R_{34} &:= (1/2, 1/5 - 2\gamma^2, \gamma). \end{aligned}$$

The total volume of L is bounded by 1 since

$$\begin{aligned} \mathrm{Vol}(L) &= 29 \cdot \left(\frac{1}{2}\left(\frac{1}{5} + \gamma\right)\left(\frac{1}{3} + \gamma\right)\right) + \frac{\gamma^2}{2} + 3 \cdot \frac{\gamma}{10} + \frac{1}{2}\left(\frac{1}{5} - 2\gamma^2\right)\gamma \\ &= 29 \cdot \left(\frac{1}{30} + \frac{\gamma}{10} + \frac{\gamma}{6} + \frac{\gamma^2}{2}\right) + \frac{\gamma^2}{2} + \frac{3\gamma}{10} + \left(\frac{\gamma}{10} - \gamma^3\right) \end{aligned}$$

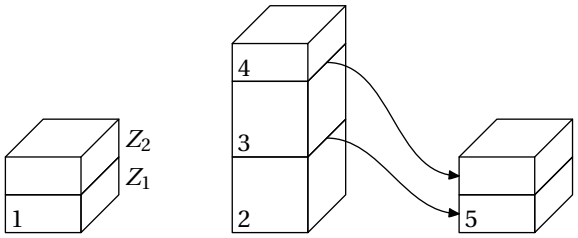


Figure 2.4: The small boxes can be packed into at most 5 bins

$$
\begin{aligned}
&= \frac{29}{30} + \frac{33}{10}\gamma + \frac{1}{6}\gamma + 15\gamma^2 - \gamma^3 \\
&= \frac{29}{30} + \frac{52}{15}\gamma + 15\gamma^2 - \gamma^3 \\
&\overset{\gamma=\frac{1}{500}}{<} \frac{29}{30} + \frac{1}{60}.
\end{aligned}
$$

Application of Algorithm C packs $R_1, \ldots, R_{29}$ in Step 1, resulting in 8 layers. All remaining boxes are packed into one more layer in Step 3. The height of each layer is greater than $1/3$, which means that the layers cannot be arranged in less than 5 bins. □

A partition of the boxes which are not small is given by the following lemma. In the sequel we use the notion of the *projection* of a box; for each box $R = (x, y, z)$ we call the rectangle (y, z) the x-projection, the rectangle (x, z) the y-projection and the rectangle (x, y) the z-projection of R.

Lemma 2.3.3. *Let L be a list of n boxes in which no small boxes and at most 3 big boxes occur. Then L can be partitioned into sets X', Y' and Z' in time $\mathcal{O}(n)$, such that each of these contains at most one big box and the x-projections of boxes in X', the y-projections of boxes in Y' and the z-projections of boxes in Z' contain no long or no wide rectangles.*

Proof. Remove the at most 3 big boxes from L and distribute them in X', Y' and Z' such that in each of these sets at most one big box occurs. Set $X' := X' \cup \{R_i \in L \mid x_i > 1/2, z_i \leq 1/2\}$, $Y' := Y' \cup \{R_i \in L \mid y_i > 1/2, x_i \leq 1/2\}$ and finally $Z' := Z' \cup \{R_i \in L \mid z_i > 1/2, y_i \leq 1/2\}$ to obtain the claim. To see that X', Y' and Z' form a partition consider a box R that is x-big without loss of generality. If this box is y-big and z-big, it is big and therefore included in one of the sets. Otherwise, it is either z-big and in Z' or small in direction z and thus in X'. On the other hand no box can be in more than one of the sets. □

We are now ready to give the overall algorithm. To avoid repetition, we enumerate the cases in the analysis only.

Algorithm D.

1. Set $\delta := \epsilon/(8+\epsilon)$. Use a PTAS for non-geometric 4-dimensional KP from [45, 47] to select $L' \subseteq L$ such that $P(L') \geq (1-\delta)\,\mathrm{OPT}$, where OPT denotes the optimum of the integral linear program
$$\text{maximize} \sum_{i=1}^{n} p_i R_i \text{ subject to } R \in P$$
where R_i is an indicator variable for the box of the same name. Furthermore, we define the set P by the constraints
$$\begin{aligned}
\sum_{i=1}^{n} x_i y_i z_i R_i &\leq 1,\\
\sum_{R_i \in X} y_i z_i R_i &\leq 1,\\
\sum_{R_i \in Y} x_i z_i R_i &\leq 1,\\
\sum_{R_i \in Z} x_i y_i R_i &\leq 1.
\end{aligned}$$
In total, P is a polytope of nonnegative integers.

2. Partition L' into at most 8 subsets which permit a feasible packing as described below. Out of these, return one with maximum profit.

Theorem 2.3.4. *Algorithm D is an $(8+\epsilon)$-approximation algorithm for OKP-3 with running time $\mathcal{O}(T_{4DKP}(n,\epsilon) + n\log^2 n/\log\log n)$, where $T_{4DKP}(n,\epsilon)$ is the running time of the PTAS used for 4-dimensional KP; furthermore this bound is tight.*

Proof. The first constraint of the integral linear program models the volume bound of the box. The other constraints are area bounds for d-big boxes for $d \in \{x,y,z\}$, motivated by the observation that the d- projections of d-big boxes do not overlap. Thus, the given program is a relaxation of our problem.

We have not imposed a bound on the number of big boxes in the relaxation, but due to the area conditions, there are at most 3 big boxes in the selected set. We consider two cases according to the total projection area of the d-big boxes.

Case 1: There is a direction $d \in \{x,y,z\}$ such that the total d-projection area of all d-big boxes in L' is larger than or equal to $1/2$. In this case all d-big boxes can be packed into at most 3 bins with a construction from [40], which can be done in time $\mathcal{O}(n\log^2 n/\log\log n)$,

resulting in a volume of at least $1/4$ being packed. The total volume of the remaining boxes is bounded by $3/4$ and each remaining box has a d-height of at most $1/2$. We apply Algorithm A in direction d, which results in a strip of d-height at most 3 and two additional levels of d-height at most $1/2$ each. Similar to the proof of Lemma 2.3.2, all these sets can be packed into at most 5 bins (see Figure 2.4), generating at most 8 bins in total.

Case 2: For each $d \in \{x, y, z\}$ the total projection area of all d-big boxes is smaller than $1/2$. By Lemma 2.3.3 we partition the set $\{R_i \in L' \mid R_i \text{ is not small}\}$ into sets X', Y' and Z' such that the total projection area of X', Y' and Z' for the corresponding direction is not greater than $1/2$ and the x-projections of boxes in X', the y-projections of boxes in Y' and the z-projection of boxes in Z' contain no long or no wide rectangles, respectively. Furthermore, each of these sets contains at most one big box. By Theorem 2.2.1 the sets X', Y' and Z' can be packed into at most one bin each, resulting in at most 3 bins in total. Let S denote the set of small boxes; these are not yet packed. Clearly $\text{Vol}(S) \leq 1$ holds, so by Lemma 2.3.2 the set S can be packed into at most 5 bins, which results in at most 8 bins in total. The runtime bound follows from the fact that we can distinguish between the two cases in time $\mathcal{O}(n)$. In [45] the best running time for a PTAS for the multi-dimensional knapsack problem is $\mathcal{O}(n^{\lceil d/\epsilon \rceil - d})$.

For the tightness of the bound, consider the instance L in which $R_1, \ldots, R_{34}$ are as in the proof of Lemma 2.3.2, and

$$\begin{aligned} R_{35} &:= (1, 1, 1/180) \\ R_{36} &:= (1, 1/180, 1) \\ R_{37} &:= (1/180, 1, 1) \\ R_{38} &:= (1, 1, 1). \end{aligned}$$

The profits are defined by

$$p_i := \begin{cases} \frac{1}{9(8+\epsilon)} & \text{for all } i \in \{1, \ldots, 4, 30, \ldots, 34\} \\ \frac{1}{8(8+\epsilon)} & \text{for all } i \in \{5, \ldots, 28\} \\ \frac{1}{(8+\epsilon)} & \text{for all } i \in \{29, 35, 36, 37\} \\ 1 & \text{for } i = 38. \end{cases}$$

Let $S_1 := L \setminus \{R_{38}\}$ and $S_2 := \{R_{38}\}$. Since $P(S_1) = 8/(8+\epsilon) = (1-\delta) < 1 = P(S_2)$, S_2 is an optimal solution. Elementary calculation verifies that S_1 may be chosen in Step 1 of Algorithm D. Application of Algorithm D leads to Case 2 in the analysis above, where $X' = \{R_{35}\}$, $Y' =$

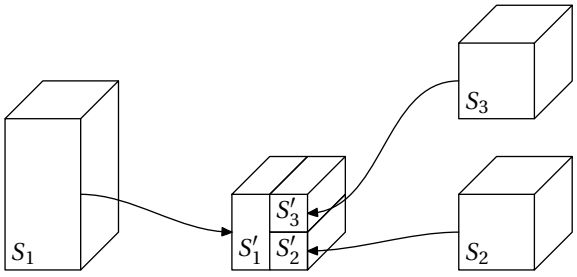


Figure 2.5: Arrangement of the stacks into a cube

$\{R_{37}\}$ and $Z' = \{R_{36}\}$. Each of these sets is packed into a separate bin. The remaining boxes are small and are packed into 5 bins as in the proof of Lemma 2.3.2. In total, 8 bins are generated; the profits are chosen such that each bin yields a profit of exactly $1/(8+\epsilon)$. □

2.4 A Packability Criterion

Theorem 2.2.1 has a number of applications besides the use in this chapter (see [40, 54]). In this section, we give a generalization to Theorem 2.2.1 for the three-dimensional case.

Lemma 2.4.1. *Let L be a list of n boxes such that* $\mathrm{Vol}(L) \leq 1/8$ *holds and no d_1-big and no d_2-big boxes occur in L for $d_1, d_2 \in \{x, y, z\}$ and $d_1 \neq d_2$. Then L permits a feasible packing into the unit bin which can be generated in time $\mathcal{O}(n \log^2 n / \log\log n)$.*

Observe that similar to Theorem 2.2.1, we are allowed to have items that are d-big in a certain direction d but not in any other direction. A difference between the original two-dimensional version and our generalization is that in the original version, a single big item, which is neither long nor wide, was allowed, but is forbidden here.

Proof. Without loss of generality assume that there are no x-big and no z-big boxes in L. Obtain the set L_{scaled} by scaling the given instance by 2 in direction x and z. We use Algorithm A in direction z to pack the boxes in L_{scaled} into one stack S_1 of height at most $4\,\mathrm{Vol}(L_{\text{scaled}}) \leq 4 \cdot 1/2 = 2$ and two additional stacks S_2 and S_3 of height limited by 1 which correspond to the summands Z_1 and Z_2 in Theorem 2.3.1. Rescaling everything yields a stack S'_1 of height at most 1 and two additional stacks S'_2 and S'_3 of heights $1/2$. All stacks have a width of at most $1/2$ in direction x. Thus we can arrange all stacks in a unit cube as shown in Figure 2.5. □

2.5 An Improved Strip Packing Algorithm

In Theorem 7 in [51], an approximation algorithm for three-dimensional strip packing with asymptotic approximation ratio $13/4$ is presented, more precisely the bound is $13/4\,\mathrm{OPT}+8Z$, where Z is the height of the highest item. In this section, we show that the additive constant of this bound can be easily improved upon by using Theorem 2.2.1; more precisely we obtain the following result by using a more suitable subdivision which results in fewer groups.

Theorem 2.5.1. *For each list L of n boxes a packing into the strip $[0,1]\times[0,1]\times[0,\infty)$ of height at most $13/4\,\mathrm{OPT}+4Z$ can be generated, where OPT denotes the minimum attainable packing height and Z is the height of the highest box. The running time is polynomial in n.*

Proof. The construction and the proof are similar as in [51] and it is included for completeness. We partition L in five groups by letting

$$
\begin{aligned}
L_1 &:= \left\{R_i \in L \,\middle|\, \mathrm{br}(R_i) \text{ is big}\right\},\\
L_2 &:= \left\{R_i \in L \,\middle|\, \mathrm{br}(R_i) \text{ is long and } \mathrm{A}(\mathrm{br}(R_i)) \le 1/6\right\},\\
L_3 &:= \left\{R_i \in L \,\middle|\, \mathrm{br}(R_i) \text{ is long and } \mathrm{A}(\mathrm{br}(R_i)) > 1/6\right\},\\
L_4 &:= \left\{R_i \in L \,\middle|\, \mathrm{br}(R_i) \text{ is wide or small, and } \mathrm{A}(\mathrm{br}(R_i)) \le 1/6\right\}, \text{ and}\\
L_5 &:= \left\{R_i \in L \,\middle|\, \mathrm{br}(R_i) \text{ is wide or small, and } \mathrm{A}(\mathrm{br}(R_i)) > 1/6\right\}
\end{aligned}
$$

and discuss how to obtain good corresponding area guarantees. Clearly, an area guarantee of $1/4$ can be obtained for L_1 by simply stacking the boxes on top of one another. Furthermore, the groups $L_2,\dots,L_5$ can be sorted in non-increasing order of z_i. Now for each of these, we proceed similar as in Step 2.3 of Algorithm A. For L_2 we generate layers using Theorem 2.2.1 and obtain an area guarantee of $1/2-1/6=1/3$ for each layer except the last one. In an even simpler way, the group L_3 can be packed by putting at least two boxes on each layer except for the last one; we obtain an area guarantee of $1/3$ for each but the last layer. By using Theorem 2.2.1, we generate layers for the items in L_4 and again obtain an area guarantee of $1/3$ for each but the last layer. Finally, the boxes of L_5 can be packed by placing two of them on each layer except for the last one; again we obtain an area guarantee of $1/3$ for each but the last layer.

Now let $\mathrm{h}(L_1)$ denote the height of the packing generated for L_1 and, for each $i\in\{2,\dots,5\}$, let $\mathrm{h}(L_i)$ denote the height of the packing of L_i minus the height of the corresponding *first*

layer generated. Let $\mathrm{h}(L)$ denote the total height of the packing. For $H := \sum_{i=2}^{5} \mathrm{h}(L_i)$ clearly

$$\mathrm{h}(L) \leq \mathrm{h}(L_1) + H + 4Z \tag{2.8}$$

holds; furthermore, we have

$$V(L_1) \geq \frac{\mathrm{h}(L_1)}{4}.$$

Now let $i \in \{2,\dots,5\}$ and let $L_1^{(i)},\dots,L_j^{(i)}$ denote the levels generated for L_i; finally let $V(L_k^{(i)})$ denote the total volume of boxes in $L_k^{(i)}$ and let h_k denote the height of $L_k^{(i)}$ for each $k \in \{1,\dots,j\}$. Since for $L_1^{(i)},\dots,L_{j-1}^{(i)}$ we have an area guarantee of at least $1/3$, we obtain

$$V(L_i) > \sum_{k=1}^{j-1} V(L_k^{(i)}) \geq \frac{1}{3} \sum_{k=2}^{j} h_k = \frac{\mathrm{h}(L_i)}{3}.$$

In total, we obtain

$$\mathrm{OPT}(L) \geq V(L) = \sum_{i=1}^{5} V(L_i) > \frac{\mathrm{h}(L_1)}{4} + \frac{H}{3}.$$

Obviously, we also have $\mathrm{OPT}(L) \geq \mathrm{h}(L_1)$, and thus

$$\mathrm{OPT}(L) \geq \max\left\{\mathrm{h}(L_1), \frac{\mathrm{h}(L_1)}{4} + \frac{H}{3}\right\} \tag{2.9}$$

holds. Now we study the ratio

$$r := \frac{\mathrm{h}(L_1) + H}{\max\left\{\mathrm{h}(L_1), \frac{\mathrm{h}(L_1)}{4} + \frac{H}{3}\right\}}. \tag{2.10}$$

If $\mathrm{h}(L_1) \geq \mathrm{h}(L_1)/4 + H/3$, which means that $\mathrm{h}(L_1) \geq 4/9\,H$, then we have

$$r = \frac{\mathrm{h}(L_1) + H}{\mathrm{h}(L_1)} = 1 + \frac{H}{\mathrm{h}(L_1)} \leq 1 + \frac{9}{4} = \frac{13}{4}.$$

If $\mathrm{h}(L_1) \leq \mathrm{h}(L_1)/4 + H/3$, which means that $\mathrm{h}(L_1) \leq 4/9\,H$, then

$$r = \frac{\mathrm{h}(L_1) + H}{\frac{\mathrm{h}(L_1)}{4} + \frac{H}{3}}$$

holds. In this case, r is a strictly monotonically increasing function of $\mathrm{h}(L_1)$, thus the maximum is attained for $\mathrm{h}(L_1) = 4/9\,H$ and $13/4$ is the corresponding maximum value. In total, we

have $r \leq 13/4$. Rearrangement of the inequalities above yields

$$\begin{aligned} \mathrm{h}(L) &\overset{(2.8)}{\leq} \mathrm{h}(L_1) + H + 4Z \\ &\overset{(2.10)}{=} r \max\left\{\mathrm{h}(L_1), \frac{\mathrm{h}(L_1)}{4} + \frac{H}{3}\right\} + 4Z \\ &\overset{(2.9)}{\leq} \frac{13}{4}\,\mathrm{OPT}(L) + 4Z, \end{aligned}$$

which proves the claim. □

2.6 Enumerations and a Shifting Technique

Algorithms B and D generate cutting areas in the strip, resulting in subsets that have to be repacked. We permit further loss of profit by removing more boxes to discard inconvenient layers; the loss will be suitably bounded. The improvement will be at the cost of a considerably larger running time due to a large enumeration; we thus omit a run time analysis. First we introduce a shifting technique to remove sets intersecting the cutting areas and the additional layers. In the sequel we use the notion of a *gap* which is a rectangular region in the strip that does not intersect the interiors of packed boxes.

Lemma 2.6.1. *Let $L = \{R_1, \ldots, R_n\}$ be a list of boxes with $z_i \leq \epsilon$ for each $R_i \in L$. Suppose L admits a packing into a strip of height at most h and let m be a positive integer. Then we can create m gaps of shape $[0,1] \times [0,1] \times [0,\epsilon]$ in a packing of height h by deleting boxes such that for the remaining list $L' \subseteq L$ the inequality $\mathrm{P}(L') \geq (1 - 2(m+1)\epsilon/h)\,\mathrm{P}(L)$ holds. The construction can be done in time polynomial in n.*

Proof. Consider the original packing in a strip of height exactly h. We partition the strip into regions of height ϵ and eventually one region of smaller height. More precisely, we define $p := \lceil h/\epsilon \rceil$ and partition the strip of height h into p regions $S_1, \ldots, S_p$ of shape $[0,1] \times [0,1] \times [0,\epsilon]$ where the uppermost region is possibly of smaller height. Then for each $i \in \{1, \ldots, p\}$ let $T_i = \{R_j \in L \mid R_j \cap S_i \neq \emptyset\}$ and let $U_1, \ldots, U_{m+1}$ be the $m+1$ sets out of $T_1, \ldots, T_p$ which have the lowest profit. It is easy to see that removing these from the packing causes a loss of profit which is at most $2(m+1)/p\,\mathrm{P}(L)$, since each box is included in at most two of the sets $T_1, \ldots, T_p$. We remove $m+1$ sets, since the uppermost region can be among the sets with lowest profit, but this region might have height smaller than ϵ; by removing $m+1$ sets we assert that we create at least m gaps of height ϵ. Let L' be the set of remaining boxes; then

$$\mathrm{P}(L') \geq \mathrm{P}(L) - \frac{2(m+1)}{p}\,\mathrm{P}(L) = (1 - \frac{2(m+1)}{p})\,\mathrm{P}(L) \geq (1 - 2(m+1)\frac{\epsilon}{h})\,\mathrm{P}(L)$$

holds. □

Note that the construction above can be carried out in any direction.

Theorem 2.6.2. *Let $R_1, \dots, R_n$ be a list of boxes with $z_i \le \epsilon$ for each $R_i \in L$ and $\mathrm{Vol}(L) \le \alpha \le 1$. Then it is possible to select $L'' \subseteq L$ such that $\mathrm{P}(L'') \ge (1-12\epsilon)\,\mathrm{P}(L)$ holds and L'' admits a feasible packing into at most $\lceil 4\alpha \rceil$ bins. The construction can be carried out in time polynomial in n.*

Proof. Without loss of generality $\epsilon \le 1/12$ holds, since otherwise $L'' = \emptyset$ will do. Use Algorithm A to pack L into a strip which is of height $h \le 4\alpha$ and two additional layers L_1 and L_2 by Theorem 2.2.2. If $h \le 1-2\epsilon$ we can clearly pack L into one bin without losing any profit. For $1-2\epsilon < h < 1$ use Lemma 2.6.1 to generate 2 gaps in the strip, which causes a loss of profit of at most $6\epsilon/h\,\mathrm{P}(L)$. But since $\epsilon \le 1/12$ and $h > 1-2\epsilon \ge 1-2/12 = 5/6$ it holds that

$$\frac{6\epsilon}{h}\mathrm{P}(L) \le \frac{6\epsilon}{\frac{5}{6}}\mathrm{P}(L) = \frac{36}{5}\epsilon\,\mathrm{P}(L) < 12\epsilon\,\mathrm{P}(L).$$

Moreover, placing L_1 and L_2 into the gaps gives a feasible packing into one bin.

For the remainder of the proof we thus assume $h \ge 1$. The following construction is illustrated in Figure 2.6. Use Lemma 2.6.1 to generate at most 5 suitable gaps in the strip, resulting in a loss of profit of at most $12\epsilon/h\,\mathrm{P}(L)$; since $h \ge 1$, this loss is bounded by $12\epsilon\,\mathrm{P}(L)$. The remaining set of boxes in the strip and L_1 and L_2 is denoted as L''. Consider the 3 cutting unit squares $[0,1]\times[0,1]\times\{i\}$ for $i \in \{1,2,3\}$ and let L_3, L_4 and L_5 be the sets of boxes in the strip that intersect with these unit squares, respectively. Without loss of generality none of the sets $L_1, \dots, L_5$ is empty; otherwise it is removed from consideration. Note that each of the sets $L_1, \dots, L_5$ can be arranged on a layer of height at most ϵ, so we generate a feasible packing by arranging them into the 5 gaps. In the resulting packing, the 3 cutting unit squares $[0,1]\times[0,1]\times\{i\}$ for $i \in \{1,2,3\}$ do not intersect with any box. Furthermore, all layers $L_1, \dots, L_5$ are merged in the strip; the packing can be rearranged into $\lceil 4\alpha \rceil$ bins. □

Like before for any $d \in \{x,y,z\}$ we call a box R_i *d_ϵ-big* $:\Leftrightarrow$ $d_i \in (\epsilon, 1]$ and *d_ϵ-small* $:\Leftrightarrow$ $d_i \in (0,\epsilon]$.

We now give the overall algorithm, which is based on a separation into boxes that are d_δ-big in all directions and boxes that are d_δ-small in at least one direction. We explain the details in the proof only.

Algorithm E.

1. Set $\delta := \epsilon/[37(7+\epsilon)]$, let $L_1 := \left\{R_i \;\middle|\; R_i \text{ is } d_\delta\text{-big for each } d \in \{x,y,z\}\right\}$ and $L_2 := L \setminus L_1$.

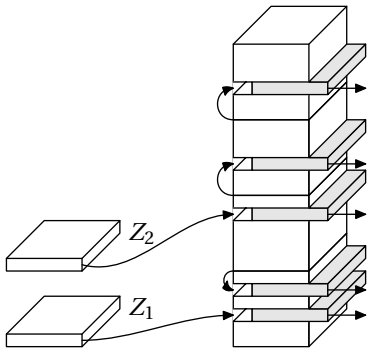


Figure 2.6: The shifting technique described in Theorem 2.6.2

2. For each $L_3 \subseteq L_1$ such that $|L_3| \leq \lfloor 1/\delta^3 \rfloor$ use an exact algorithm to test the packability of L_3. Store a feasible L_3 of maximum total profit.

3. Use an FPTAS for classical KP from [45, 47] to select $L_4 \subseteq L_2$ such that $\mathrm{Vol}(L_4) \leq 1$ and $\mathrm{P}(L_4) \geq (1-\delta)\,\mathrm{OPT}$ holds.

4. Use the construction described below to select $L_5 \subseteq L_4$ which can be packed into at most 6 bins under a small loss of profit.

5. Out of the at most 7 sets generated in Step 2 and Step 4, return one with maximum profit.

Theorem 2.6.3. *Algorithm E is a $(7+\epsilon)$-approximation algorithm for OKP-3.*

Proof. Note that $\lfloor 1/\delta^3 \rfloor$ is an upper bound for the number of boxes from L_1 in a feasible solution since δ^3 is a lower bound for the volume of each $R_i \in L_1$. Step 2 can be carried out in time polynomial in δ and thus polynomial in $1/\epsilon$ using an exact optimization algorithm as in [5]. We show that in Step 4 at most 6 sets are generated, resulting in at most 7 bins in total. Partition L_4 into 3 subsets X', Y' and Z' such that in each of these all boxes R_i are d_δ-small for the corresponding direction; note that $\mathrm{Vol}(X')+\mathrm{Vol}(Y')+\mathrm{Vol}(Z') \leq 1$ holds. We apply the construction from Theorem 2.6.2 in each of the three directions. Study the following cases, where $\mathrm{Vol}(X') \geq \mathrm{Vol}(Y') \geq \mathrm{Vol}(Z')$ holds without loss of generality.

Case 1: $\mathrm{Vol}(X') \in (3/4, 1]$. The boxes in X' can be packed into at most 4 bins. We have $\mathrm{Vol}(Y')+\mathrm{Vol}(Z') < 1/4$. This means $\mathrm{Vol}(Y') < 1/4$ and $\mathrm{Vol}(Z') < 1/4$ hold. Consequently Y' and Z' can be packed into at most one bin each, resulting in at most 7 bins.

Case 2: $\mathrm{Vol}(X') \in (1/2, 3/4]$. The boxes in X' can be packed into at most three bins. Furthermore, $\mathrm{Vol}(Y')+\mathrm{Vol}(Z') < 1/2$, which means that $\mathrm{Vol}(Y') < 1/2$ holds. Consequently the boxes

in Y' can be packed into at most 2 bins. Furthermore, $\mathrm{Vol}(Z') < 1/4$ holds and finally the boxes in Z' can be packed into at most one bin; this generates at most 7 bins in total.

Case 3: We have $\mathrm{Vol}(X') \in [0, 1/2]$. The boxes in X' can be packed into at most two additional bins. Furthermore, $\mathrm{Vol}(Y') \leq 1/2$ and $\mathrm{Vol}(Z') \leq 1/2$ hold. This means that the boxes in Y' and Z' can be packed into at most two bins each. In total at most 7 bins are generated.

To prove the approximation ratio fix an optimal solution S and let P_1^* be the profit of boxes in $S \cap L_1$ and let P_2^* be the profit of boxes in $S \cap L_2$. Consequently $P_1^* + P_2^* = \mathrm{OPT}$ holds. Let P_1 be the profit of the set that is stored in Step 2 and let P_2 be the profit of the set that is selected in Step 3. By construction we have $P_1 \geq P_1^*$ and $P_2 \geq (1-\delta)P_2^*$. Furthermore, by threefold application of the construction from Theorem 2.6.2 the loss of profit in P_2 is bounded by $36\delta P_2$; let P_2' denote the remaining list. The profit of the set returned in Step 5 is at least

$$\begin{aligned}\frac{1}{7}(P_1 + P_2') &\geq \frac{1}{7}(P_1^* + (1-\delta)(1-36\delta)P_2^*) \\ &\geq \frac{1}{7}(P_1^* + P_2^*)(1-\delta)(1-36\delta) \\ &= \frac{1}{7}\,\mathrm{OPT}(1-\delta)(1-36\delta) \geq \frac{1}{(7+\epsilon)}\,\mathrm{OPT}\end{aligned}$$

which proves the claimed approximation ratio. □

Theorem 2.6.4. *The bound of $(7+\epsilon)$ for the approximation ratio of Algorithm E is asymptotically tight in the sense that it cannot be improved for ϵ arbitrary small.*

Proof. Let $\epsilon \in (0, 1/4]$ and δ be defined as in Step 1 of Algorithm E. Note that $\delta \leq 1/259$. Set $\gamma := \max\{(12i)^{-1} \mid i \in \mathbb{N}, (12i)^{-1} \leq \delta\}$. Let $\alpha \in \mathbb{R}_+$ such that $(3/4 - \alpha)(1/3 + \alpha) > 1/4$, $(1/3 - 2\alpha)(3/4 + 6\alpha) > 1/4$ and $2\alpha \leq \gamma$ hold. It is easy to see that such an α exists. Note that $\{1/(2\gamma), 1/(3\gamma), 1/(4\gamma), 1/(6\gamma)\} \subseteq \mathbb{N}$. We use the boxes

$$\begin{aligned}
A_i &:= \left(\frac{1}{2}, \frac{1}{2}+\alpha, \gamma\right) && \text{for } i \in \left\{1, \ldots, \frac{1}{\gamma}\right\}, && A := \left\{A_1, \ldots, A_{\frac{1}{\gamma}}\right\}, \\
B_i &:= \left(\frac{1}{2}, \frac{1}{2}+\alpha, \gamma\right) && \text{for } i \in \left\{1, \ldots, \frac{1}{4\gamma}-1\right\}, && B := \left\{B_1, \ldots, B_{\frac{1}{4\gamma}-1}\right\}, \\
C_i &:= \left(\gamma, \frac{1}{2}+\alpha, \frac{1}{2}\right) && \text{for } i \in \left\{1, \ldots, \frac{1}{2\gamma}\right\}, && C := \left\{C_1, \ldots, C_{\frac{1}{2\gamma}}\right\}, \\
D_i &:= \left(\gamma, 1, \frac{1}{4}+\alpha\right) && \text{for } i \in \left\{1, \ldots, \frac{1}{2\gamma}\right\}, && D := \left\{D_1, \ldots, D_{\frac{1}{2\gamma}}\right\}, \\
E_i &:= \left(\gamma, \frac{1}{3}+\alpha, \frac{3}{4}-\alpha\right) && \text{for } i \in \left\{1, \ldots, \frac{1}{3\gamma}\right\}, && E := \left\{E_1, \ldots, E_{\frac{1}{3\gamma}}\right\},
\end{aligned}$$

$$
\begin{aligned}
F_i &:= \left(\frac{1}{3}+\alpha, \gamma, \frac{3}{4}-\alpha\right) && \text{for } i \in \left\{1,\dots,\frac{1}{6\gamma}-1\right\}, && F := \left\{F_1,\dots,F_{\frac{1}{6\gamma}-1}\right\},\\
G_i &:= \left(\frac{1}{3}+\alpha, \gamma, \frac{3}{4}-\alpha\right) && \text{for } i \in \left\{1,\dots,\frac{1}{2\gamma}-1\right\}, && G := \left\{G_1,\dots,G_{\frac{1}{2\gamma}-1}\right\},\\
H_i &:= \left(\frac{1}{3}-2\alpha, \gamma, \frac{3}{4}+6\alpha\right) && \text{for } i \in \left\{1,\dots,\frac{1}{2\gamma}-1\right\}, && H := \left\{H_1,\dots,H_{\frac{1}{2\gamma}-1}\right\},\\
R_1 &:= (\delta+\alpha, \delta+\alpha, \delta+\alpha).
\end{aligned}
$$

to define the list L.

In the following we will show that L admits a feasible packing into the unit cube. Note that each of the sets (without the box R_1) defined above can be arranged to a rectangular block by stacking them on top of each other in the direction d in which they are d_δ-small (see Figure 2.7). Furthermore, we note that

$$
\begin{aligned}
|A| &= \frac{1}{\gamma}, && \mathrm{br}_z(A_i) > \frac{1}{4} \text{ for } A_i \in A, && \sum_{A_i \in A} z(A_i) = 1,\\
|B| &= \frac{1}{4\gamma}-1, && \mathrm{br}_z(B_i) > \frac{1}{4} \text{ for } B_i \in B, && \sum_{B_i \in B} z(B_i) = \frac{1}{4}-\gamma,\\
|C| &= \frac{1}{2\gamma}, && \mathrm{br}_x(C_i) > \frac{1}{4} \text{ for } C_i \in C, && \sum_{C_i \in C} x(C_i) = \frac{1}{2},\\
|D| &= \frac{1}{2\gamma}, && \mathrm{br}_x(D_i) > \frac{1}{4} \text{ for } D_i \in D, && \sum_{D_i \in D} x(D_i) = \frac{1}{2},\\
|E| &= \frac{1}{3\gamma}, && \mathrm{br}_x(E_i) > \frac{1}{4} \text{ for } E_i \in E, && \sum_{E_i \in E} x(E_i) = \frac{1}{3},\\
|F| &= \frac{1}{6\gamma}-1, && \mathrm{br}_y(F_i) > \frac{1}{4} \text{ for } F_i \in F, && \sum_{F_i \in F} y(F_i) = \frac{1}{6}-\gamma,\\
|G| &= \frac{1}{2\gamma}-1, && \mathrm{br}_y(G_i) > \frac{1}{4} \text{ for } G_i \in G, && \sum_{G_i \in G} y(G_i) = \frac{1}{2}-\gamma,\\
|H| &= \frac{1}{2\gamma}-1, && \mathrm{br}_y(H_i) > \frac{1}{4} \text{ for } H_i \in H, && \sum_{H_i \in H} y(H_i) = \frac{1}{2}-\gamma
\end{aligned}
$$

holds, where $\mathrm{br}_d(R)$ denotes the base rectangle in direction d and $d(R)$ denotes component d for a box R and $d \in \{x, y, z\}$. As indicated above, these sets together with the box R_1 permit a feasible packing since

- F, E, C and F, E, B can be placed next to one another in direction y,
- C, B, D can be placed next to one another in direction z,

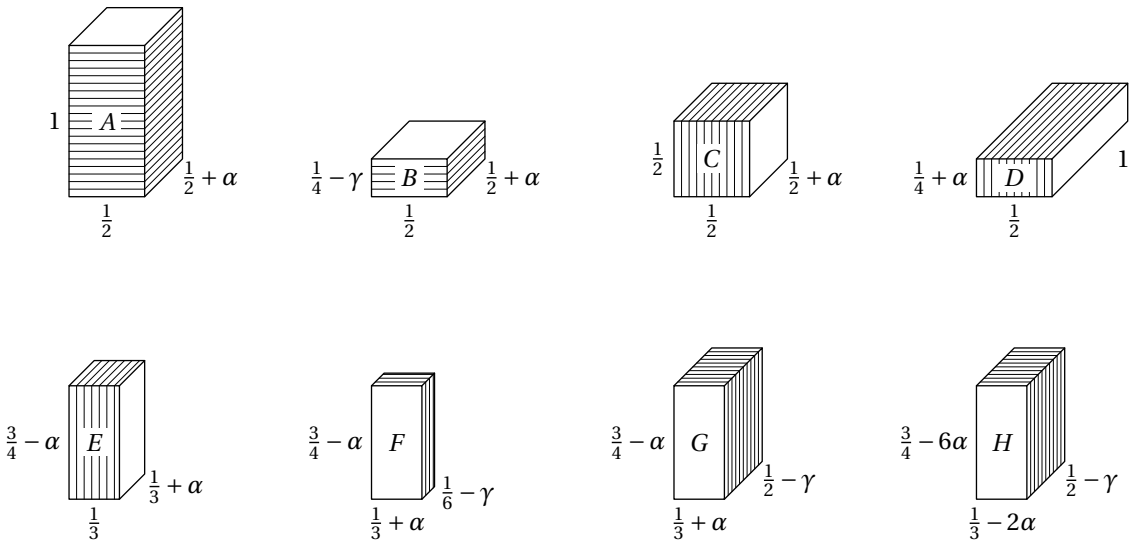


Figure 2.7: Boxes arranged in blocks

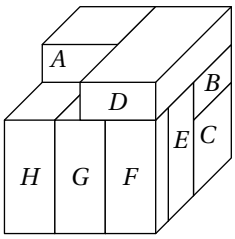


Figure 2.8: A feasible packing of the boxes into the unit cube

- A, H and A, G and B, G and C, G can be placed next to one another in direction y,

which can be verified by elementary calculation (see Figure 2.8).

In the following we discuss the execution of Algorithm E. The profits necessary for our construction will be described together with the presentation; first we require $\mathrm{P}(L) = 1$; this also equals OPT since L permits a feasible packing. In Step 1 we obtain $L_1 = \{R_1\}$ and $L_2 = L \setminus \{R_1\}$. Step 2 stores $\{R_1\}$ as a feasible candidate for selection later; here we require $\mathrm{P}(R_1) = 1/(7+\epsilon)$. In Step 3 we assume that $L_4 = L_2$, which is possible since L_2 is an optimal solution for the generated knapsack instance; note that here $\mathrm{P}(L_4) = 1 - 1/(7+\epsilon) = (6+\epsilon)/(7+\epsilon)$. Now we discuss how L_4 is packed in Step 4. First L_4 is partitioned into

$$X' = C \cup D \cup E, \quad Y' = F \cup G \cup H, \quad Z' = A \cup B.$$

Then each of these three sets is packed into a strip in the corresponding direction; note that

$$\mathrm{br}_x(R) > \frac{1}{4} \text{ for } R \in X', \quad \mathrm{br}_y(R) > \frac{1}{4} \text{ for } R \in Y', \quad \mathrm{br}_z(R) > \frac{1}{4} \text{ for } R \in Z'$$

holds. Hence Algorithm E packs all boxes on top of one another in the corresponding direction; we assume that the boxes in each of the sets are packed in "lexicographical" order. Note that the first layer is separated. We denote the heights of the generated strips for X', Y' and Z' by $h_{X'}$, $h_{Y'}$ and $h_{Z'}$, respectively, and obtain

$$h_{X'} = \frac{4}{3} - \gamma, \quad h_{Y'} = \frac{7}{6} - 4\gamma, \quad h_{Z'} = \frac{5}{4} - 2\gamma.$$

Since $\gamma \leq \delta \leq 1/259$, all of these strips have height more than 1 and will later result in 2 bins. Let X'_1 denote the set of boxes that are in or intersect the region $[0,1] \times [0,1] \times [0,1]$ in the strip and set $X'_2 := X' \setminus X'_1$. We define Y'_1, Y'_2 and Z'_1, Z'_2 in a similar way. Now we require that

$$P(X'_1) = P(X'_2) = P(Y'_1) = P(Y'_2) = P(Z'_1) = P(Z'_2) = \frac{1}{6}\frac{6+\epsilon}{7+\epsilon}.$$

The profit is evenly partitioned among the items in the sets. Algorithm E uses the shifting technique from Lemma 2.6.1 to generate feasible packings for X', Y' and Z', respectively, into 2 bins. Hence three gaps of height δ are generated into each of the strips causing a small loss of profit; the separated layer and the boxes intersecting the cutting square are merged into each strip. It is easy to see that the shifting technique acts only on X'_1, Y'_1, and Z'_1, respectively, since the profit on each item here is lower than in the corresponding second sets. Thus no item is swapped between sets by the shifting technique. We obtain 6 bins each containing subsets of X'_1, X'_2, Y'_1, Y'_2, Z'_1, and Z'_2.

Since each of the sets holds a profit of $(6+\epsilon)/[6(7+\epsilon)]$, each bin holds a profit less than or equal to $(1+\epsilon/6)/(7+\epsilon)$ which finishes the proof. □

2.7 The Rotational Case

Finally, we discuss the application of Algorithm E on two different rotational scenarios. In both scenarios, rotations of the boxes are only permitted by 90° around certain axes. In the first case, which we denote by *z-oriented* OKP-3, rotations are only permitted around the z-axis. This setting is motivated by packing fragile goods and has been considered in the strip packing variant in [21, 58]. In the second case, which we denote by *rotational* OKP-3, rotations are permitted around all three axes. Note that for both scenarios, $Q = (1,1,1)$ does not hold without loss of generality, but is an explicit assumption. Surprisingly, a better approximation ratio can be obtained easily although implicitly the search space is dramatically enlarged.

We show how Algorithm E can be revised to yield a better approximation ratio for both

scenarios. Step 2 is modified in such a way that the exact packing algorithm takes rotations into account; as before, here at most one bin is generated. The most important part is the modification of Step 4. We separate the description of the two scenarios and start with the z-oriented setting. Let Z be the set of boxes $R_i \in L_4$ that are z-δ-small and $X = L_4 \setminus Z$. We introduce a pre-processing step in which each $R_i \in X$ is rotated in such a way that the side length x_i is minimal. Consequently, $x_i \leq \delta$ holds for each $R_i \in X$. Hence, the generation of a strip for direction y can be removed; we build only two strips in directions x and z to which the shifting technique from Theorem 2.6.2 is applied. This again causes an additional loss of profit which is bounded, however. Let $i \in \{1,\ldots,4\}$ such that $(i-1)/4 \leq \text{Vol}(Z) \leq i/4$. Then by Theorem 2.6.2, exactly i bins are needed to pack the strip in direction z. Furthermore, we get

$$\text{Vol}(X) \leq \text{Vol}(L_4) - \text{Vol}(Z) \leq 1 - \frac{i-1}{4} = \frac{5-i}{4}$$

which yields a number of $5-i$ bins for the corresponding strip. In total $1+i+(5-i) = 6$ bins are needed to pack all items. We call the resulting approach Algorithm F and obtain the following result.

Theorem 2.7.1. *Algorithm F is a $(6+\epsilon)$-approximation algorithm for z-oriented OKP-3.*

For the *rotational* OKP-3 we have to continue the approach above. Since 90° rotations around all axes are permitted, all items that are d_δ-small for any direction $d \in \{x,y,z\}$ can be packed into one strip in direction z by orienting them such that they are z-δ-small. Since $\text{Vol}(L_4) \leq 1$, application of the construction of Theorem 2.6.2 yields at most 4 additional bins. We call the resulting approach Algorithm G and obtain the following result.

Theorem 2.7.2. *Algorithm G is a $(5+\epsilon)$-approximation algorithm for rotational OKP-3.*

2.8 Conclusion

In this chapter we contributed approximation algorithms for an NP-hard combinatorial optimization problem, where the running times of the simpler algorithms are practical. It is is an interesting open question whether an algorithm with ratio $(6+\epsilon)$ or less exists for the non-rotational case and whether there are algorithms with better approximation ratio for the rotational cases. Furthermore, we are interested in a reduction of the running time, especially for Algorithm E.

In [50] it was proved that it is NP-complete to decide whether a set of squares can be packed into the unit square. However, it is an open problem whether checking the packability of cubes into the unit cube is NP-complete.

3 Scheduling Problems

3.1 Introduction

In classical scheduling theory, each job is executed by only one processor at a time. In the recent years, however, due to the rapid development of parallel computer systems, new theoretical approaches have emerged to model scheduling on parallel architectures (for an overview about scheduling of multiprocessor jobs on such parallel architectures see for example [9, 18, 49]).

In this chapter, we study variants of the non-preemptive parallel job scheduling problem. An instance of this problem is given by a list $L := \{J_1, \ldots, J_n\}$ of jobs and for each job J_j an execution/processing time p_j and the number of required machines q_j is given. A schedule $S = ((s_1, r_1), \ldots, (s_n, r_n))$ is a sequence of starting times $s_j \geq 0$ together with the set of assigned machines $r_j \subseteq \{1, \ldots, m\}$ ($|r_j| = q_j$) for $j \in \{1, \ldots, n\}$. A schedule is feasible if each machine executes at most one job at a time. The length of a schedule is defined as its latest job completion time $C_{\max} = \max\{s_j + p_j \mid j \in \{1, \ldots, n\}\}$. The objective is to find a feasible schedule of minimal length. This problem is denoted by $P|\text{size}_j|C_{\max}$ (for more information on this three field notation see for example [18]).

3.1.1 Known Results

$P|\text{size}_j|C_{\max}$ is strongly NP-hard, since the problem $P5|\text{size}_j|C_{\max}$, where the number of available processors is 5, is NP-hard in the strong sense [19]. Furthermore, there is no approximation algorithm with a performance ratio better than 1.5 for $P|\text{size}_j|C_{\max}$ [43], unless P = NP.

The best known algorithm with polynomial running time for this problem was implicitly given by Garey and Graham [25]. They proposed a list-based algorithm with approximation ratio 2 for a resource-constrained scheduling problem. In this scheduling problem one or more resources are given and each job requires a certain amount of each resource for the duration of its execution time. As pointed out by Ludwig & Tiwari [54] this resource-constrained scheduling problem can be used to model $P|\text{size}_j|C_{\max}$ by using the

available processors as the single resource. The existence of a polynomial time approximation scheme (PTAS) for the case that the number of available processors is a constant, $Pm|\text{size}_j|C_{\max}$, was presented in [1, 31].

A problem closely related to $P|\text{size}_j|C_{\max}$ is the *strip packing* problem (i.e. packing of rectangles in a strip of width 1 while minimizing the packing height). The main difference is that machines assigned to a job need to be contiguous in a solution of the strip packing problem. Turek et al. [66] pointed out that using contiguous machine assignments is desirable in some settings; for example to maintain a physical proximity of processors allotted to a job. This contiguous case is known under different names in the literature, amongst others: scheduling on a line, $P|\text{line}_j|C_{\max}$, or non-fragmentable multiprocessor system. In the literature, even further models are considered to model the underlying network topology, such as *meshes* or hypercubes. Note that the one-dimensional mesh corresponds to the line model, whereas the two-dimensional mesh corresponds to three-dimensional strip packing (see for example [22, 23, 8, 68]). One of the first results for the strip packing problem was given by Coffman et al. [12]. They proved that the level-based algorithms NFDH (Next Fit Decreasing Height) and FFDH (First Fit Decreasing Height) algorithms have an approximation ratio of 3 and 2.7 respectively. Based on these level-based algorithms Sleator [63] presented an algorithm with approximation ratio 2.5. The currently best known algorithms with absolute approximation ratio 2 were given independently by Schiermeyer [61] and Steinberg [64]. Coffman et al. [12] also analyzed the asymptotic performance of NFDH and FFDH, which is $2 \cdot \text{OPT} + h_{\max}$ and $1.7 \cdot \text{OPT} + h_{\max}$ respectively, where OPT is the height of an optimal solution and $h_{\max}$ denotes the height of the tallest rectangle. An AFPTAS for the strip packing-problem was presented by Kenyon & Rémila [46]. Only recently, Jansen and Solis-Oba [34] presented an asymptotic polynomial time approximation scheme (APTAS) with additive term 1 at the cost of a higher running time.

A similar problem is the scheduling of so-called *malleable* jobs, where the number of required machines for each job is not known a priori; the execution time of each job depends on the number of allotted machines. That is, instead of p_j, q_j each job J_j has an associated function $p_j : \{1, \ldots, m\} \to \mathbb{Q}^+$ that gives the execution time $p_j(\ell)$ of that job in terms of the number ℓ of processors that are assigned to J_j. For this scheduling problem Ludwig & Tiwari [54] presented an algorithm with approximation ratio 2. Jansen & Porkolab [31] developed an approximation scheme with linear running time for both malleable jobs and non-malleable jobs as long as the number of machines is constant. For the case that preemptions are allowed (jobs can be interrupted at any time at no cost and restarted later on a possibly different set of processors) Jansen & Porkolab [32] provide an optimal algorithm with running time polynomial in m and linear in n. Mounié et al. [59] present

an $(1.5+\varepsilon)$ approximation algorithm for scheduling a set of independent monotonic malleable jobs, where the machines allotted to each job have consecutive addresses. Implicitly, this algorithm requires the number of machines to be polynomially bounded in the number of jobs, since the running time of the algorithm depends on the number of machines. Decker et al. [13] presented a 1.25-approximation for scheduling n independent identical malleable jobs on p identical processors (the jobs are called identical if the execution time on any number of processors is the same for all jobs). In [28], Jansen presented an asymptotic fully polynomial time approximation scheme (AFPTAS) for scheduling malleable jobs on an arbitrary number of machines.

In the literature, a lot of scheduling problems with additional constraints are studied. Numerous publications deal with so called *online scheduling of parallel jobs* (for a survey see [62]). In *online scheduling*, not all informations about the instance are known a priori, e.g. unknown release dates, unknown running times (see for example [60, 68]). Another often studied type of constraint are so-called *precedence constraints*, where a job can only be scheduled for execution if all of its predecessors have already completed their execution (see for example [7, 41, 42, 48]). In many papers also combinations of different constraints are studied; for example in [8, 22, 23]) *online* scheduling with *precedence constraints* is studied. Note that in [22, 23] results for different network topologies such as *PRAM*, *line*, *meshes*, *hypercubes* are presented; in [8] *hypercubes* and *arrays* are considered as underlying network topology; in [68] *hypercubes* are considered.

3.1.2 New Results

In this thesis, we focus on the natural case where the number of machines is polynomially bounded in the number of jobs (in most scenarios the number of machines will be even smaller than the number of jobs). We will denote this problem by $P\text{poly}|\text{size}_j|C_{\max}$ or by $P\text{poly}|\text{line}_j|C_{\max}$ in the contiguous case. Using a reduction from 3-PARTITION [26, SP15], it is easy to see that these problems are strongly NP-hard.

3-PARTITION: Given an integer value B and a list of $n=3k$ integers s_i with $\frac{B}{4} < s_i < \frac{B}{2}$ and $\sum_{i=1}^{n} s_i = kB$. The objective is to find a partition of the set of integer into k subsets where each subset has sum B. Note that the above constraints on the values imply that every subset must contain exactly three integers.

The reduction works as follows. We set the number of machines to B, and introduce n jobs J_i, each with processing time $p_i := 1$ and number of required machines $q_i := s_i$. If there is a schedule of length k, we have a solution for 3-PARTITION. On the other hand, if there is a solution to 3-PARTITION there exists a schedule of length k. Since 3-PARTITION is NP-

hard in the strong sense, the problem is also NP-hard for instances where all numbers are polynomially bounded in n. In particular, all instances in which B is polynomially bounded in n are NP-hard.

For the case that all machines assigned to a job have contiguous addresses, we show the existence of an algorithm with approximation ratio arbitrarily close to 1.5.

Theorem 3.1.1. *For every $\varepsilon > 0$ there exists an algorithm A such that for every instance I of $P\,poly|line_j|C_{\max}$*

$$A(I) \leq (1.5 + \varepsilon)\,\mathrm{OPT}(I)$$

holds and the running time is polynomial in n, where $A(I)$ is the length of the schedule for instance I generated by algorithm A and $\mathrm{OPT}(I)$ is the length of an optimal schedule for instance I.

The previous best known result for this problem is a 2-approximation algorithm by Ludwig & Tiwari [54]. The algorithm for scheduling monotonic malleable jobs presented by Mounié et al. [59] with approximation ratio $(1.5+\varepsilon)$ makes use of the monotonic character and thus is not applicable to the non-malleable case. Interestingly, the result is otherwise very similar. They also generate a schedule with contiguous machine addresses and they also (implicitly) assume that the number of machines is polynomially bounded in the number of jobs since the running time depends on the number of machines.

This result holds also for the strip packing problem if we restrict the instances such that the width of each rectangle is a multiple of $1/m$ for some integer value m that is polynomially bounded in the number of rectangles.

In the general case (non-contiguous addresses) we show the existence of a polynomial time approximation scheme (PTAS).

Theorem 3.1.2. *For every $\varepsilon > 0$ there exists an algorithm A such that for every instance I of $P\,poly|size_j|C_{\max}$*

$$A(I) \leq (1 + \varepsilon)\,\mathrm{OPT}(I)$$

holds and the running time is polynomial in n, where $A(I)$ is the length of the schedule for instance I generated by algorithm A and $\mathrm{OPT}(I)$ is the length of an optimal schedule for instance I.

The previous best known result for this problem is the above mentioned 2-approximation algorithm for the resource-constrained scheduling problem by Garey & Graham [25]. Other algorithms with absolute approximation ratio lower than 2 presume that the number of machines is a constant.

Furthermore, we show in Section 3.6, how these results can be extended to the malleable case (with the same approximation ratios), even if the execution time of the jobs is not monotone. This result cannot be obtained by applying the framework described by Ludwig & Tiwari [54], since the analysis of the approximation ratio of our algorithm is not based on lower bounds that are required for their framework.

For the non-contiguous case these are the best possible results (in the sense of approximation ratio), since the problem is NP-hard in the strong sense for both the malleable and the non-malleable case.

3.1.3 Structure

We start this chapter with a short outline of the algorithms in Section 3.2. In Section 3.3, we present an algorithm to pack rectangles into a constant number of bins. This algorithm will be used as *subroutine* in the following scheduling algorithms. In Section 3.4, we present the algorithm for scheduling jobs on machines with contiguous addresses. We present the scheduling algorithm for the case that the machines allotted to each job are not required to have contiguous addresses in Section 3.5. In Section 3.6, we show how the algorithms for the non-malleable cases can be extended to solve the corresponding malleable versions with the same approximation ratio. We conclude with open problems in Section 3.7.

3.2 Outline of the Algorithms

Before we go into the details, we give a short outline of the algorithms.

A crucial part of the algorithms is to schedule *critical* jobs (jobs with long execution time or large number of required processors) nearly optimally. This is done by enumerating a polynomial number of schedules for the critical jobs. To ensure that there is at least one schedule among these that allows a nearly optimal solution, we have to reduce the search space. Therefore, we show that it is possible to modify an optimal schedule such that the resulting schedule is nearly optimal and has a simpler structure.

To be more specific, the first step in our algorithms is to guess (enumerate) the approximate value of an optimal solution (Sections 3.4.1.1, 3.5.1.1, 3.6.1.1). This allows us to divide the solution into a constant number of *slots* with height depending on the accuracy. After that, we partition the set of jobs. The purpose of this step is to create a gap in size (processing time / number of required machines) between *big* and *small* jobs. This is done by discarding *middle-sized* jobs (Sections 3.4.1.2, 3.5.1.2, 3.6.1.2). We schedule all discarded jobs using a greedy algorithm in a post-processing step. Then, we round the long jobs (Sec-

tions 3.4.1.3, 3.5.1.3, 3.6.1.3) and define containers into which we place (some of) the short jobs. From here on the algorithms for the non-contiguous and the contiguous case differ significantly. For the contiguous case we guess (enumerate) a set of containers. Since the actual packing algorithm cannot guarantee to schedule all long jobs, a crucial step is to take care of jobs with running time $> 1/2$ (Section 3.4.2.3). We then solve a linear program (Section 3.4.3) and use its solution to create the actual schedule for the containers and for a subset of the long jobs (Section 3.4.4). The scheduling of short jobs inside the container is done by a modified version of the algorithm by Kenyon & Rémila [46] (Section 3.4.4.6).

For the non-contiguous case we show that the long jobs can be scheduled in a canonical way (Section 3.5.3) and use a dynamic program to assign the long jobs to *slots* (Section 3.6.2). The scheduling of the short jobs is again done by using the modified version of the algorithm by Kenyon & Rémila (Section 3.5.4).

The extension to the malleable cases is done by choosing an assignment of jobs to a number of machines in a first phase (Section 3.6). After that the solution can be found by applying the algorithms used for the non-malleable cases.

3.3 Packing into a Constant Number of Bins

In the following, we present a modification of the algorithm by Kenyon and Rémila [46], which we will call mKR. Instead of packing into one target strip, we want to pack into a constant number of bins with different sizes. We show that under certain assumptions (see (A1)–(A6)) almost all rectangles can be packed into the bins, i.e. the rectangles that are not packed have small total area.

Let $\mathscr{C} = \{C_1, \dots, C_k\}$ be a set of k bins. We assume that all bins have width and height bounded by 1. Let $L = \{R_1, \dots, R_n\}$ denote the set of rectangles and let $L = L_{\text{sm}} \cup L_{\text{wi}}$ be a partition of the set of rectangles into wide and small rectangles and define

- $h^w_{\max} := \max_{R \in L_{\text{wi}}} \mathrm{h}(R)$, $w^w_{\min} := \min_{R \in L_{\text{wi}}} \mathrm{w}(R)$, $w^w_{\max} := \max_{R \in L_{\text{wi}}} \mathrm{w}(R)$,
- $h^s_{\max} := \max_{R \in L_{\text{sm}}} \mathrm{h}(R)$, $w^s_{\max} := \max_{R \in L_{\text{sm}}} \mathrm{w}(R)$.

With $\mathrm{h}(Q), \mathrm{w}(Q)$ we denote the height and the width of a rectangle Q or a bin Q, respectively. Furthermore, we denote with $\mathrm{A}(Q) := \mathrm{h}(Q)\,\mathrm{w}(Q)$ the area of Q. We extend the notations to sets in the straight-forward manner (for example $\mathrm{A}(\mathscr{C}) = \sum_{C \in \mathscr{C}} \mathrm{A}(C)$).

Let $c, \delta > 0$. With the following assumptions we can formulate the theorem.

$$\text{There exists a packing of } L_{\text{wi}} \text{ into the bins } \mathscr{C}, \tag{A1}$$

$$\mathrm{A}(L) \leq \mathrm{A}(\mathscr{C}), \tag{A2}$$

$$w^w_{\min} \geq \delta, \tag{A3}$$

$$w^s_{\max} \leq \frac{1}{k}\frac{\delta}{7}, \tag{A4}$$

$$h^w_{\max} \leq \frac{\delta}{7} \min\left\{\frac{1}{k}, w^w_{\min}\frac{\delta}{7}\right\}, \tag{A5}$$

$$h^s_{\max} \leq \frac{\delta}{7} \min\left\{\frac{1}{4k}, w^w_{\min}\frac{\delta}{4\cdot 7}\right\}. \tag{A6}$$

Theorem 3.3.1. *Under the assumptions (A1)–(A6) there exists an algorithm with running time polynomial in n, k and $1/\delta^2$ that packs almost all rectangles into the bins, i.e. the unpacked rectangles have total area at most $\delta\,\mathrm{A}(L)$ if $\mathrm{A}(L) \geq 1$ or δ otherwise.*

In the following sections, we briefly describe the algorithm.

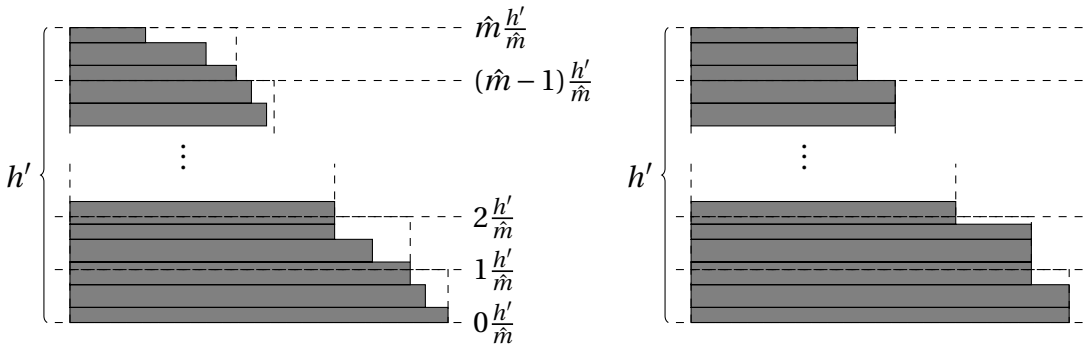
3.3.1 Grouping and Rounding

In order to simplify the problem, we transform the rectangles from L_{wi} into a set L_{sup} that consists only of rectangles with $\hat{m}$ (a constant depending on $w^w_{\min}$) different widths. This transformation is similar to the grouping technique used by Kenyon and Rémila in [46]. Note that this simplification is feasible, since it is not necessary to pack the rectangles optimally.

First we order all rectangles from L_{wi} by non-increasing width. Then we stack them left-aligned on top of each other, resulting in a stack of height $h' = \mathrm{h}(L_{\text{wi}})$ (see Figure 3.1a). Next we draw horizontal cutting lines at heights $i(h'/\hat{m})$ for $i \in \{0, \ldots, \hat{m}\}$ across the stack. We say that rectangle $R \in L_{\text{wi}}$ belongs to group i, if its upper side u_R satisfies $i(h'/\hat{m}) < u_R \leq (i+1)(h'/\hat{m})$.

We generate L_{sup} by rounding up the width of each rectangle $R_i \in L_{\text{wi}}$ such that its width is the same as the width of the widest rectangle belonging to the same group (see Figure 3.1b). Since all wide rectangles are packable (Assumption (A1)) we can ensure the existence of a feasible fractional packing for L_{sup} by removing all rectangles intersecting $[0, w^w_{\max}] \times [0, h'/\hat{m}]$. These intersecting rectangles have total area bounded by

$$V_1 := \left(\frac{h'}{\hat{m}} + h^w_{\max}\right) w^w_{\max} \leq \frac{h'}{\hat{m}} + h^w_{\max}.$$



(a) Grouping

(b) Rounding rectangles, L_{sup}

Figure 3.1: Grouping of wide rectangles

The feasibility follows bascially by area arguments; since the width of each rectangle in group i is smaller than the width of each rectangle in group $(i-1)$, all rectangles can be packed (at least fractionally) into the space occupied by rectangles from the next lower group.

3.3.2 Fractional Bin Packing

To find a fractional packing for L_{sup} we use basically the same linear program as in [46]. The main difference is that

- we have a set of configurations for each bin instead of one set of configurations, and
- we need k constraints in addition to the $\hat{m}$ constraints in [46].

The k additional constraints ensure that the total height of the configurations for each bin does not exceed the height of the bin. The $\hat{m}$ *original* constraints ensure that all rectangles are covered (one constraint for each group of rectangles). Thus, instead of $\hat{m}$ constraints we have $\hat{m}+k$ constraints and solving the linear program results in at most $\hat{m}+k$ variables with non-zero value instead of $\hat{m}$ non-zero variables.

Since we assume that all wide rectangles are packable (Assumption (A1)) and since we discarded all rectangles intersecting $[0, w^w_{\max}] \times [0, h'/\hat{m}]$, there exists a feasible (fractional) solution for the rounded instance.

3.3.3 Packing the Rectangles

Each of the $\hat{m}+k$ non-zero variables corresponds to one of the configurations. In order to generate a packing, we define layers inside each bin with height corresponding to the value

of the variables. In contrast to the algorithm by Kenyon and Rémila we do not increase the height of the layers.

The space for each layer can now be divided into the *left* side with width equal to the width of the corresponding configuration and the *right* side, which will be used for packing the rectangles from L_{sm}.

The packing of the rectangles from L_{sup}, or rather the packing of the rectangles from L_{wi}, is done in the same way as in [46], but since we have not increased the height of the layers, the topmost rectangles might be overlapping into the next layer or over the upper border of the bin. In order to make the packing feasible, we simply remove all overlapping rectangles. The rectangles removed in this step have total area bounded by

$$V_2 := (\hat{m} + k)(1 \cdot h^w_{\max}),$$

since the height of each of these rectangles is bounded by $h^w_{\max}$ and the width of each layer is bounded by 1 and the number of layers is bounded by $\hat{m} + k$.

The rectangles from L_{sm} will be added by a modified version of the Next Fit Decreasing Height (mNFDH) algorithm. After ordering L_{sm} by non-increasing height, we use NFDH layer by layer. Note that we add an additional layer for each bin, if the upper border of the last layer is below the upper border of this bin. Furthermore, we add an additional layer if the space reserved by a configuration is not completely packed. Since this can only happen if there are not *enough* rectangles belonging to a specific group, we get at most $\hat{m}$ additional layers. Thus, in total we pack the low rectangles into at most $2\hat{m} + k$ layers.

If all of L_{sm} is packed by the mNFDH algorithm the total area of all unpacked rectangles is bounded by $V_1 + V_2$. Otherwise, we have to calculate how much of the total area of all bins is covered by the (fractionally) packed rectangles from L_{wi} and by the packed rectangles from L_{sm} in order to get an upper bound for the total area of the remaining rectangles from L_{sm}.

Obviously, the wasted space on the right side of each bin and layer is bounded by the width of the small rectangles (since we use NFDH); this bound holds even for the last layer, since not all rectangles could be packed. Additionally, we can guarantee for each layer (including the additional layers) that an area with height corresponding to the height of the layer minus two times the height of the tallest rectangles from L_{sm} is covered, i.e. the uncovered space is bounded in total by

$$\begin{aligned} V_3 &:= \mathrm{h}(\mathscr{C})\, w^s_{\max} && \text{right side} \\ &\quad + (2\hat{m} + k + k) 2 h^s_{\max} && \text{upper bound for uncovered area for each layer} \\ &\leq k w^s_{\max} + (2\hat{m} + 2k) 2 h^s_{\max}. \end{aligned}$$

In total, we obtain as upper bound for the total area of all unpacked rectangles

$$\begin{aligned}
V_1 + V_2 + V_3 &= \frac{h'}{\hat{m}} + h^w_{\max} + (\hat{m}+k)(1 \cdot h^w_{\max}) + k w^s_{\max} + (2\hat{m}+2k)2h^s_{\max} \\
&\leq \frac{\mathrm{A}(L)}{w^w_{\min}\hat{m}} + h^w_{\max} + (\hat{m}+k)(1 \cdot h^w_{\max}) + k w^s_{\max} + (2\hat{m}+2k)2h^s_{\max} \\
&\overset{(A5)}{\leq} \frac{\mathrm{A}(L)}{w^w_{\min}\hat{m}} + \frac{\delta}{7} + \hat{m}h^w_{\max} + \frac{\delta}{7} + k w^s_{\max} + 4\hat{m}h^s_{\max} + 4k h^s_{\max} \\
&\overset{(A4),(A6)}{\leq} \frac{\mathrm{A}(L)}{w^w_{\min}\hat{m}} + \frac{\delta}{7} + \hat{m}h^w_{\max} + \frac{\delta}{7} + \frac{\delta}{7} + 4\hat{m}h^s_{\max} + \frac{\delta}{7}.
\end{aligned}$$

Choosing $\hat{m} := \frac{7}{\delta w^w_{\min}} \overset{(A3)}{\leq} \frac{7}{\delta^2}$, this equals

$$\begin{aligned}
&= \mathrm{A}(L)\frac{\delta}{7} + \frac{4\delta}{7} + \frac{7}{\delta w^w_{\min}} h^w_{\max} + \frac{4 \cdot 7}{\delta w^w_{\min}} h^s_{\max} \\
&\overset{(A5),(A6)}{\leq} \mathrm{A}(L)\frac{\delta}{7} + \frac{4\delta}{7} + \frac{7}{\delta w^w_{\min}}\frac{\delta}{7} w^w_{\min}\frac{\delta}{7} + \frac{4 \cdot 7}{\delta w^w_{\min}}\frac{\delta}{7} w^w_{\min}\frac{\delta}{4 \cdot 7} \\
&= \mathrm{A}(L)\frac{\delta}{7} + \frac{4\delta}{7} + \frac{\delta}{7} + \frac{\delta}{7} \\
&= \mathrm{A}(L)\frac{\delta}{7} + \frac{6\delta}{7} \\
&\leq \begin{cases} \delta & \text{if } \mathrm{A}(L) \leq 1 \\ \delta\,\mathrm{A}(L) & \text{if } \mathrm{A}(L) \geq 1. \end{cases}
\end{aligned}$$

The running time of our algorithm is polynomial in n, k and $1/\delta^2$. For a detailed analysis of the running time we refer the reader to the analysis used in [46]. The main difference is the number of configurations and the additional k constraints.

This proves Theorem 3.3.1.

3.4 Contiguous Parallel Job Scheduling

In this section, we present the algorithm for scheduling parallel jobs such that the machines assigned to a job have contiguous indices, $P\mathrm{poly}|\mathrm{line}_j|C_{\max}$. Note that the algorithm presented in this section uses some of the techniques presented in [34]. The main difference concerns jobs with processing time $> 1/2$. Our modifications ensure that no job with processing time $> 1/2$ is discarded.

Since each job is required to be executed on contiguous machines, an instance of this

problem can be translated directly into a strip packing instance; for each job J_i create a rectangle $R_i = (w_i, h_i)$ of width $w_i = q_i/m$ and height $h_i = p_i$ (where q_i is the number of required machines and p_i is the execution time of J_i). We scale the width of R_i by $1/m$, such that the target strip in the resulting strip packing instance has width 1. The objective is then to find an orthogonal, axis parallel arrangement of all rectangles into a strip of width 1 and minimal height (without rotations). Thus, the strip packing and the scheduling notation can be used synonymously in the contiguous case. In this chapter (especially in this section), we will use the strip packing notation since this notation is more descriptive. The non-contiguous case can also be viewed as strip packing problem if *fragmentation* of rectangles is allowed in one dimension (width).

Obviously, a solution for the contiguous case is a feasible solution for the non-contiguous case. However, an optimal solution for the contiguous case is in general not optimal for the non-contiguous case. Turek et al. [66] presented an example instance (see Table 3.1) that shows that the length of an optimal schedule for the non-contiguous case can be shorter than for the contiguous case (see Figure 3.2).

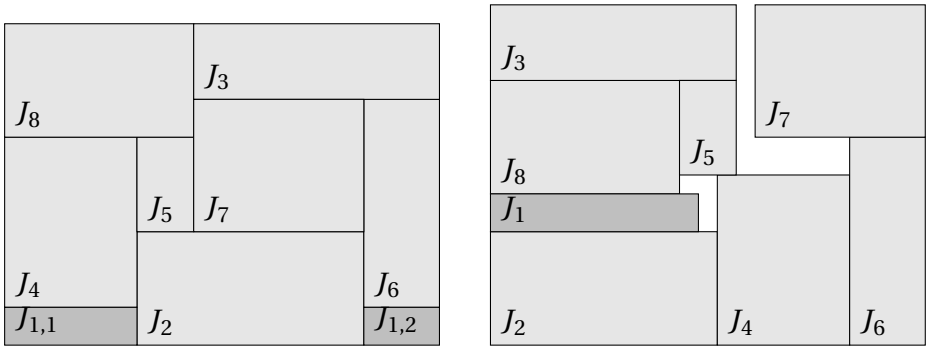
Job	Processors	Time
1	11	2
2	12	6
3	13	4
4	7	9
5	3	5
6	4	11
7	9	7
8	10	6

Table 3.1: Jobs to be scheduled in the example of Figure 3.2

3.4.1 Near-Optimal Schedule with Simple Structure

In the following, we describe the construction of a nearly optimal solution with simple structure based on a given optimal solution. This simply-structured solution is similar to the solution constructed in [34]. The main difference is that here we make sure that the total area of the discarded jobs is small, while the total profit of the discarded rectangles is small in the construction in [34].

In the following, let $0 < \varepsilon \leq 1$ be the required accuracy and let $L = \{R_1, \ldots, R_n\}$ be an instance of $P\text{poly}|\text{line}_j|C_{\max}$. For each rectangle (job) R_i let w_i be its width (number of processors q_i/m) and let h_i be its height (execution time p_i). A packing (schedule) P for instance



(a) Non-contiguous machine indices (b) Contiguous machine indices

Figure 3.2: Optimal schedules can have different lengths

L is given as a set of pairs $P = \{(x_1, y_1), \ldots, (x_n, y_m)\}$, where each pair $(x_i, y_i) \in \mathbb{R}^2_{\geq 0}$ denotes the position of the lower left corner of rectangle R_i in the strip. Note that in this case the representation of the schedule by a packing is sufficient, since the subset of assigned processors is well-defined by the first assigned processor. We assume that the lower left corner of the strip coincides with the origin of a Cartesian system of coordinates. A packing P is valid if the rectangles do not overlap and $x_i + w_i \leq 1$ for all $i \in \{1, \ldots, n\}$. The height of packing P is given by

$$\mathrm{h}(P) := \max_{i \in \{1,\ldots,n\}} (y_i + h_i).$$

3.4.1.1 Bounded Height

Since we want to divide the solution into a constant number of *slots*, we need to know the height of an optimal solution, at least up to the required accuracy ε.

By using the strip packing algorithm of Steinberg [64], we can find a solution for the strip packing instance with height $v \leq 2 \cdot \mathrm{OPT}$, where OPT is the height of an optimal solution. Obviously, there exists a value

$$v^* \in \left\{ (1+0\varepsilon)\frac{v}{2}, (1+1\varepsilon)\frac{v}{2}, \ldots, (1+\left\lceil \frac{1}{\varepsilon} \right\rceil \varepsilon)\frac{v}{2} \right\}$$

such that $\mathrm{OPT} \leq v^* \leq (1+\varepsilon)\,\mathrm{OPT}$; we only have to consider $\lceil 1/\varepsilon \rceil + 1$ different candidates to find the right one. For simplicity we divide the height of each rectangle by v^*, such that the height $\mathrm{OPT}' := \frac{\mathrm{OPT}}{v^*}$ of an optimal solution for the scaled instance satisfies

$$1 - \varepsilon < \frac{1-\varepsilon}{1-\varepsilon^2} = \frac{1}{1+\varepsilon} = \frac{\mathrm{OPT}}{(1+\varepsilon)\,\mathrm{OPT}} \leq \frac{\mathrm{OPT}}{v^*} \leq 1.$$

In the following, we show the existence of an algorithm that packs all rectangles of a

scaled instance into a strip of height at most $(1+\varepsilon+1/2)$ (see Section 3.4.5). This height bound is sufficient to prove Theorem 3.1.1, since rescaling yields

$$\begin{aligned} v^*\left(1+\varepsilon+\frac{1}{2}\right) &\leq (1+\varepsilon)\,\mathrm{OPT}\left(\frac{3}{2}+\varepsilon\right) = \left(\frac{3}{2}+\frac{5\varepsilon}{2}+\varepsilon^2\right)\mathrm{OPT} \\ &\leq (1.5+4\varepsilon)\,\mathrm{OPT}. \end{aligned} \tag{3.1}$$

In the following, we assume that the instance is already scaled such that an optimal packing P^* has height $\mathrm{h}(P^*)$, where $(1-\varepsilon) < \mathrm{h}(P^*) \leq 1$.

3.4.1.2 Partitioning the Set of Rectangles/Creating a Gap

In this section, we create a gap between *tall* and *low* rectangles and between *wide* and *narrow* rectangles. To create this gap we need to remove some of the rectangles. The following lemma proves that the rectangles we remove have small total area.

Let ε' be the largest value of the form $\varepsilon' = 1/(2a)$ for an integer a such that $\varepsilon' \leq \varepsilon/15$. Let $\sigma_0 := 1, \sigma_1 := \varepsilon'$, and $\sigma_k := (\sigma_{k-1})^{8/\sigma_{k-1}^3}$ for all $k \geq 2$. Define

$$L^{>1/2} := \left\{ R_i \in L \;\middle|\; h_i > \frac{1+2\varepsilon'}{2} \right\},$$

and

$$L_k := \left\{ R_i \in L \setminus L^{>1/2} \;\middle|\; w_i \in (\sigma_k, \sigma_{k-1}] \text{ or } h_i \in (\sigma_k, \sigma_{k-1}] \right\}.$$

Define for each subset $L' \subseteq L$ the total area of L' by $\mathrm{A}(L') = \sum_{R_i \in L'} (w_i \cdot h_i)$.

Lemma 3.4.1. *There exists $k \in \{2, \ldots, 2/\varepsilon' + 1\}$ such that*

$$A(L_k) \leq \varepsilon' A(L).$$

Proof. Since each rectangle belongs to at most two sets L_k,

$$\sum_{j \in \{2,\ldots,\frac{2}{\varepsilon'}+1\}} A(L_j) \leq 2A(L).$$

Obviously, there exists $k \in \{2, \ldots, 2/\varepsilon' + 1\}$ with $A(L_k) \leq \varepsilon'/2 \cdot 2A(L) = \varepsilon' A(L)$, since otherwise

$$\sum_{j \in \{2,\ldots,\frac{2}{\varepsilon'}+1\}} A(L_j) > \frac{2}{\varepsilon'} \varepsilon' A(L) = 2A(L). \qquad \square$$

Choose the smallest value k satisfying the conditions of Lemma 3.4.1 and define $\delta := \sigma_{k-1}$ and $s := 8/\delta^3$ and $\gamma := \delta^s = \sigma_k$.

Note 3.4.2. Since $\frac{1}{\varepsilon'}$ is integral, $\frac{1}{\sigma_i}$ is integral for all $i \in \mathbb{N}$ and thus $\frac{1}{\delta}$, $\frac{1}{\gamma}$ and $\frac{1}{\delta^s}$ are integral.

For simplicity, we define the following sets and call rectangles belonging to each set accordingly

$$\begin{aligned} L_{\text{ta}} &:= \{R_i \in L \mid h_i > \delta\} && \text{tall rectangles} \\ L_{\text{lo}} &:= \{R_i \in L \mid h_i \leq \delta^s\} && \text{low rectangles} \\ L_{\text{wi}} &:= \{R_i \in L \mid w_i > \delta\} && \text{wide rectangles} \\ L_{\text{na}} &:= \{R_i \in L \mid w_i \leq \delta^s\} && \text{narrow rectangles.} \end{aligned}$$

Note that $L = (L_{\text{ta}} \cup L_{\text{lo}} \cup L_{\text{wi}} \cup L_{\text{na}}) \cup L_k$ and $(L_{\text{ta}} \cup L_{\text{lo}} \cup L_{\text{wi}} \cup L_{\text{na}}) \cap L_k = \emptyset$ and $L^{>1/2} \subseteq L_{\text{ta}}$. We will denote the subset of low-wide rectangles in the following with $L_{\text{lo-wi}} := L_{\text{lo}} \cap L_{\text{wi}}$. For the following steps, we discard the *middle-sized* rectangles L_k. They will be packed in a post-processing step by a simple greedy algorithm (see Section 3.4.5).

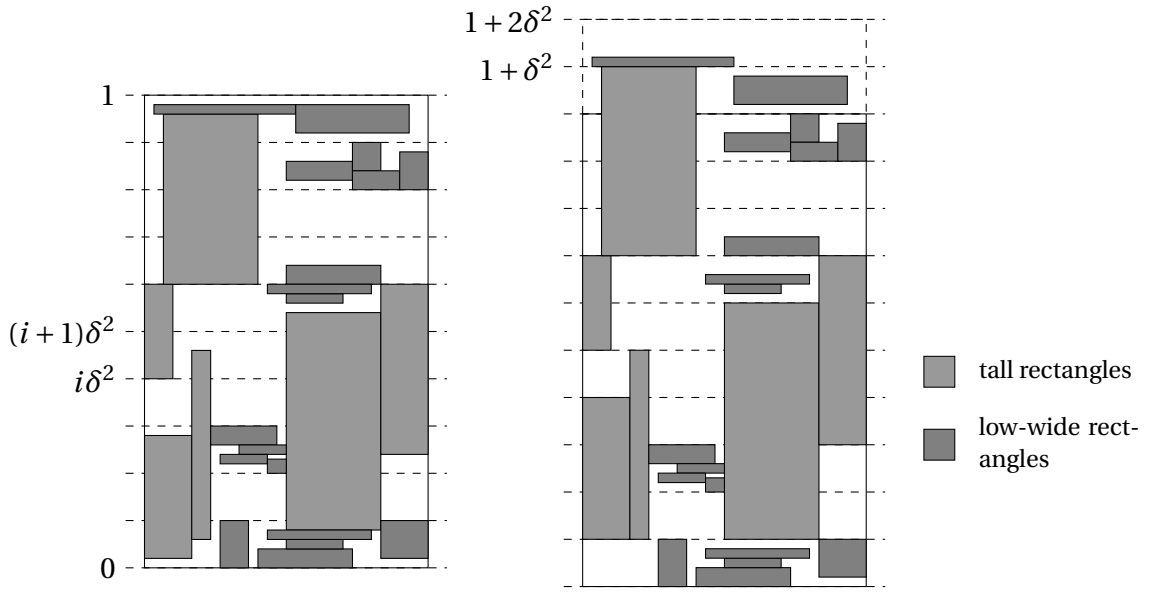
3.4.1.3 Rounding and Shifting Tall Rectangles

A crucial property of our simple structure is the restricted set of positions and heights of the tall rectangles. Let P be an optimal packing for all rectangles L. First we increase the height of each tall rectangle $R_i \in L_{\text{ta}}$ to the nearest (integral) multiple of δ^2. Then, we shift the rectangles up such that all rectangles $R_i \in L_{\text{ta}}$ have their corners placed at points (x'_i, y'_i), such that there exist integral values k_i with $x'_i = x_i$ and $y'_i = k_i\delta^2$ (see Figure 3.3). These modifications increase the height of the solution by at most 2δ.

Lemma 3.4.3. *Let P be a packing for all rectangles L with $\mathrm{h}(P) \leq 1$. At the cost of an increase in height of at most 2δ we can round up all tall rectangles to the nearest multiple of δ^2 and we can shift the rectangles such that the lower left corner of all tall rectangles is a multiple of δ^2.*

Proof. Let P be a packing for all rectangles L with $\mathrm{h}(P) \leq 1$. Let (x_i, y_i) be the position that P assigns to each rectangle R_i and let $z_i := y_i + h_i$ be the upper bound of R_i in P. Multiply each z_i by $1+2\delta$; that is, we shift up all rectangles depending on their upper bound without changing their size or the feasibility of the packing. Obviously, this modification increases the height of the packing by at most $2\delta\,\mathrm{h}(P) \leq 2\delta$. Since tall rectangles have height at least δ this shifting creates a gap with height at least

$$z_i(1+2\delta) - z_i = z_i 2\delta \geq 2\delta^2$$



(a) Original packing (b) Packing with simpler structure

Figure 3.3: Rounding and shifting rectangles

below each tall rectangle. Thus, rounding up the size of each tall rectangle to the next multiple of δ^2 (without changing z_i) and shifting down each tall rectangle to a multiple of δ^2 does not change the feasibility of the packing. □

After scaling and rounding the set of tall rectangles can be partitioned into a constant number of subsets. Define

$$I_{\text{ta}} := \left\{ \frac{1}{\delta} + i \;\middle|\; i \in \mathbb{N} : 1 \le i \le \frac{1-\delta}{\delta^2} \right\} \quad \text{and} \tag{3.2}$$

$$I^{>1/2} := \left\{ \frac{1+2\varepsilon'}{2\delta^2} + i \;\middle|\; i \in \mathbb{N} : 1 \le i \le \frac{1-2\varepsilon'}{2\delta^2} \right\} \quad \text{and} \tag{3.3}$$

$$L(i) := \left\{ R_i \in L \;\middle|\; h_i = i \cdot \delta^2 \right\}. \tag{3.4}$$

Lemma 3.4.4. *The set of all tall rectangles can be partitioned into a constant number of subsets:*

$$L_{ta} = \dot{\bigcup_{i \in I_{ta}}} L(i) \quad \textit{and} \tag{3.5}$$

$$L^{>1/2} = \dot{\bigcup_{i \in I^{>1/2}}} L(i). \tag{3.6}$$

In particular, the number of partitions of L_{ta} and $L^{>1/2}$ are bounded by $|I_{ta}| = \frac{1-\delta}{\delta^2} \le \frac{1}{\delta^2}$ and

$|I^{>1/2}| = \frac{1-2\varepsilon'}{2\delta^2} \leq \frac{1}{2\delta^2}$, *respectively.*

Proof. Obviously for all $i, j \in I_{\text{ta}}, i \neq j : L(i) \cap L(j) = \emptyset$. Due to the scaling, the height of each rectangle is at most 1 and since $1/\delta$ is integral (see Note 3.4.2), this bound still holds after rounding. After rounding, each tall rectangle has height $a\delta^2$ for an integer value a. Let $R_i \in L_{\text{ta}}$. Then $\delta < \text{h}(R_i) \leq 1$ and since δ is a multiple of δ^2, the smallest possible value for $\text{h}(R_i)$ is

$$\delta + \delta^2 = \left(\frac{1}{\delta} + 1\right)\delta^2.$$

The biggest possible value for $\text{h}(R_i)$ is

$$1 = \delta + 1 - \delta = \frac{\delta^2}{\delta} + \frac{(1-\delta)\delta^2}{\delta^2} = \left(\frac{1}{\delta} + \frac{1-\delta}{\delta^2}\right)\delta^2.$$

Since all multiples of δ^2 between these bounds are contained in I_{ta},

$$L_{\text{ta}} = \dot{\bigcup_{i \in I_{\text{ta}}}} L(i).$$

If $R_i \in L^{>1/2}$ then $\frac{1+2\varepsilon'}{2} < \text{h}(R_i) \leq 1$. Since $\frac{1}{\varepsilon'}$ is an even integer and δ^2 is a multiple of ε', $\frac{1+2\varepsilon'}{2}$ is a multiple of δ^2. The smallest possible value for $\text{h}(R_i)$ is

$$\frac{1+2\varepsilon'}{2} + \delta^2 = \left(\frac{1+2\varepsilon'}{2\delta^2} + 1\right)\delta^2.$$

The biggest possible value for $\text{h}(R_i)$ is

$$1 = \frac{1+2\varepsilon'+1-2\varepsilon'}{2} = \left(\frac{1+2\varepsilon'}{2\delta^2} + \frac{1-2\varepsilon'}{2\delta^2}\right)\delta^2.$$

Again, since all multiples of δ^2 between these bounds are contained in $I^{>1/2}$,

$$L^{>1/2} = \dot{\bigcup_{i \in I^{>1/2}}} L(i).$$

□

3.4.1.4 Containers for Low Rectangles

Since we want to increase only the height but not the width of the packing, we cannot round up the widths of the wide rectangles in order to reduce the complexity. Instead we introduce *containers*, into which all low-wide ($L_{\text{lo-wi}}$) and a subset of the low-narrow rectangles will be packed.

Consider a scaled and shifted packing P. Draw horizontal lines spaced by a distance

δ^2 across the strip (due to the rounding and shifting, the lower and upper sides of the tall rectangles coincides with two of these lines). These lines split the strip into at most $(1+2\delta)/\delta^2$ horizontal rectangular regions that we call *slots* (see Figure 3.3). A container is a rectangular region inside a slot whose left boundary is either the right side of a tall rectangle or the left side of the strip, and whose right boundary is either the left side of a tall rectangle or the right side of the strip. In the following, we consider only containers that contain at least one low-wide rectangle. In Figure 3.4 for example, we have two containers that contain low-wide rectangles.

Lemma 3.4.5. *Let P be a scaled and shifted packing for all rectangles L. The number of containers which contain at least one low-wide rectangle is bounded by $\frac{2}{\delta^3}$.*

In particular, the number of all possible sets of containers (containing at least one low-wide rectangle) is polynomial in n.

Proof. The height of each container is δ^2 by definition. Since each container in consideration contains at least one low-wide rectangle the width of each container is at least δ. Thus, the total number of containers (containing at least one low-wide rectangle) is bounded by

$$(1+2\delta)\frac{1}{\delta\cdot\delta^2} \leq 2\frac{1}{\delta^3}.$$

Furthermore, the width of each container is a multiple of $1/m$, since in each packing (schedule) x_i is a multiple of $1/m$ for each rectangle R_i and the width w_i is a multiple of $1/m$. Thus, the width of each container is in

$$\left\{\frac{1}{m},\ldots,\frac{m}{m}\right\}.$$

Therefore, a rough upper bound for the number of different sets of containers is $(m+1)^{\frac{2}{\delta^3}}$ (encode each set as a $\frac{2}{\delta^3}$-tupel, where each entry denotes the width of the corresponding container or 0 if it is not in the set). Note that in general this number will be (much) smaller, since the ordering is not relevant and the width of each container containing a low-wide rectangle is $> 1/\delta$. Since we assume that m is polynomial in n, the number of all possible sets of containers (containing at least one low-wide rectangle) is polynomial in n. □

Since the number of different sets of containers is polynomial, we can find a set corresponding to the set induced by an optimal packing by enumerating all possible sets of containers in polynomial time.

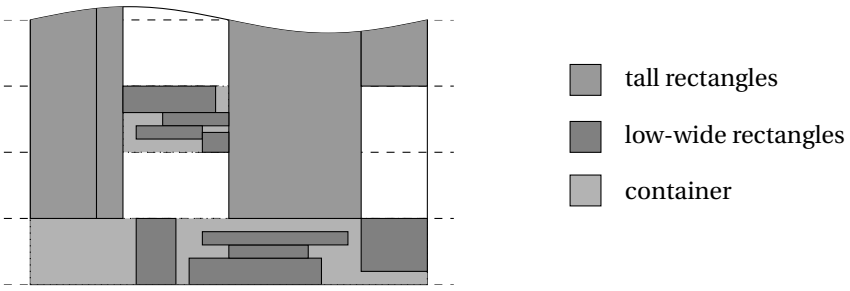


Figure 3.4: Container for low rectangles

3.4.1.5 Properties / Summary

In summary, we have shown in this section that an optimal packing P for rectangle set L with height bounded by 1 can be transformed into a packing $\hat{P}$ for rectangle set $\hat{L}$ with height at most $1+2\delta$ and simple structure, as follows

(a) every tall rectangle R_i in $\hat{L}$ has its height rounded up to the nearest multiple of δ^2 and its lower border is at a position y_i that is a multiple of δ^2 (see Lemma 3.4.3),

(b) each container C containing at least one low-wide rectangle has height δ^2 and width $i \cdot 1/m \geq \delta$ where $i \leq m$ is a non-negative integer (see Section 3.4.1.4),

(c) there is in gap in size between tall and low rectangles and between wide and narrow rectangles (see Lemma 3.4.1),

(d) the total area of the discarded rectangles is bounded, $A(L \setminus \hat{L}) \leq \varepsilon'/2$ and the height of each discarded rectangle is bounded by $(1+2\varepsilon')/2$, since we did not discard any rectangles belonging to $L^{>1/2}$ (see Section 3.4.1.2).

In the following, we present an algorithm to pack almost all rectangles from $\hat{L}$ into a strip with height at most $1+2\delta$. Obviously, this is also a feasible packing for all corresponding rectangles from L. Since the remaining rectangles have a small total area, it is possible to add these rectangles to a strip with height at most $0.5+\varepsilon'$ during a post-processing step. Overall, this leads to a packing of L with height at most $1.5+\varepsilon$.

3.4.2 Pre-Positioning

The next step is to determine the positions of the containers and a subset of the tall rectangles. On the one hand we have to make sure that all rectangles we are discarding have height bounded by $1/2$; otherwise, the NFDH algorithm used to pack all discarded rectangles during post-processing produces a packing of height $> 1/2$, leading to an overall approximation

ratio greater than $1.5+\varepsilon'$. On the other hand we have to make sure that the pre-positioning has a polynomial running time. In particular, we can only enumerate the positions of a constant number of tall rectangles and containers.

From here on let $\mathscr{C}$ be the (current) set of containers and let $L'_{\text{ta}} \subseteq L_{\text{ta}} \setminus L^{>1/2}$ be the subset of K tall rectangles with largest area for some constant K, which we will define in Section 3.4.4.4; we set $L'_{\text{ta}} := L_{\text{ta}} \setminus L^{>1/2}$ if $|L_{\text{ta}} \setminus L^{>1/2}| \leq K$. Furthermore, let

$$L' = \mathscr{C} \cup L'_{\text{ta}}$$

be the union of the set of containers and the chosen subset of at most K tall rectangles. Note that since $|\mathscr{C}| \leq 2\delta^{-3}$ (see Lemma 3.4.5),

$$|L'| \leq K + 2\delta^{-3}. \tag{3.7}$$

In order to determine the positions of the rectangles from L', first we guess (enumerate) assignments of the K tall rectangles L'_{ta} and of the containers $\mathscr{C}$ to *slots* and *snapshots*. Then we describe a dynamic program that assigns the tall rectangles from $L^{>1/2}$ to snapshots without enumerating all possibilities, since there might be too many of them. Using these assignments we set up a linear program (LP). If this LP has a solution, we have found a fractional solution for the packing problem. Furthermore, if almost all low-wide rectangles fit into the containers, we show that the fractional solution can be transformed into a feasible integral solution by discarding some rectangles with small total area.

3.4.2.1 Slot Assignment

We split again the strip into horizontal slots of height δ^2. A slot assignment for L' is a mapping $f : L' \to M$ where $M = \{1, \ldots, {}^{(1+2\delta)}/_{\delta^2}\}$ corresponds to the set of slots. For a given slot assignment f the set of slots that will be used for packing a rectangle $R_j \in L'$ is given by $\{f(R_j), \ldots, f(R_j)+\gamma_j-1\}$, where $\gamma_j\delta^2 = h_j$ is the height of rectangle R_j (in particular $\gamma_j \geq {}^1/_\delta$ for each $R_j \in L_{\text{ta}}$, since $h_j > \delta$, and $\gamma_j = 1$ if R_j is a container). Since the number of different mappings f is bounded by

$$|M|^{|L'|} \leq \left(\frac{1+2\delta}{\delta^2}\right)^{K+2\delta^{-3}} \tag{3.8}$$

and thus constant, we can consider all mappings f in polynomial time and try to find a packing for L that is consistent with f.

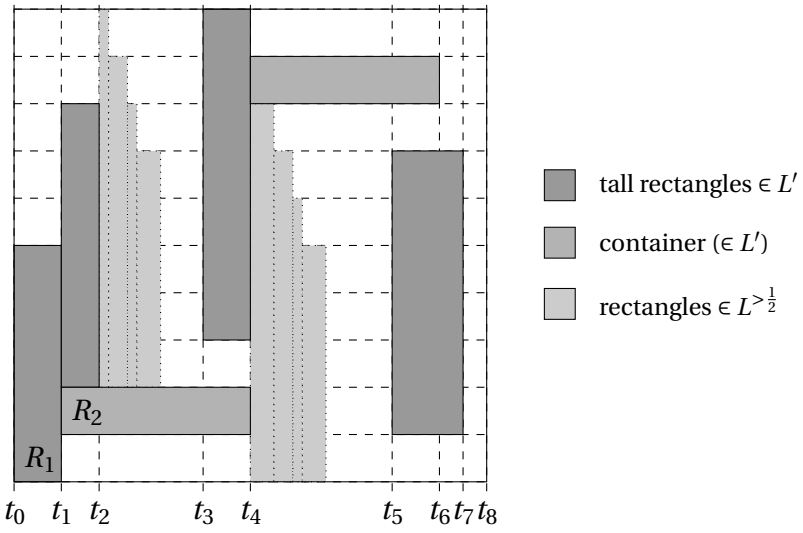


Figure 3.5: Packing of rectangles and containers and induced snapshots

3.4.2.2 Snapshots

In order to handle the x position of the rectangles, we introduce snapshots. We use the snapshots to model the relative horizontal positions of all rectangles in L'.

Consider a packing for L'. Trace vertical lines extending the sides of the rectangles in L' (see Figure 3.5). The region between two adjacent lines is called a snapshot. If we index all snapshots from left to right, every rectangle $R_j \in L'$ appears in a sequence of consecutive snapshots $S_{\alpha_j}, \ldots, S_{\beta_j}$, where α_j denotes the index of the first snapshot in which rectangle R_j occurs and β_j denotes the index of the last snapshot. In Figure 3.5 for example rectangle R_1 is contained in snapshot S_1, while R_2 is contained in snapshots S_2, S_3, S_4, thus $\alpha_1 = 1, \beta_1 = 1, \alpha_2 = 2$ and $\beta_2 = 4$. More formal, an assignment of all rectangles in L' to snapshots is given by two functions $\alpha, \beta : L' \to \{1, \ldots, g\}$, where g denotes the number of snapshots.

Since $|L'| \leq K + 2\delta^{-3}$ (Equation (3.7)) the maximum number of snapshots g in any packing for L' is at most

$$g \leq 2|L'| \leq 2(K + 2\delta^{-3}), \tag{3.9}$$

and thus the number of different assignments of L' to snapshots is polynomial, $\mathcal{O}(g^{2|L'|})$.

3.4.2.3 Dynamic Program for $L^{>1/2}$-rectangles

In general, we cannot consider all assignments of rectangles in $L^{>1/2}$ to snapshots, because there might be up to n rectangles in $L^{>1/2}$. In the following, we introduce an algorithm that allows us to enumerate a subset of all snapshot assignments for $L^{>1/2}$ such that the size of the subset is polynomially bounded in n and there exists one snapshot assignment in this subset that is *equivalent* to a snapshot assignment induced by an optimal packing.

Rectangles in $L^{>1/2}$ that intersect more than one snapshot are handled separately (see end of section), since our packing algorithm can only be used for tall rectangles that do not intersect more than one snapshot.

As already mentioned in Section 3.4.1.3, $L^{>1/2}$ can be partitioned into sets $L(i)$ for all $i \in I^{>1/2}$ such that $L^{>1/2} = \dot{\bigcup}_{i \in I^{>1/2}} L(i)$ (see Equation (3.3) and (3.6)) and

$$|I^{>1/2}| = \frac{1-2\varepsilon'}{2\delta^2} < \frac{1}{2\delta^2}. \tag{3.10}$$

Consider a packing of all rectangles and snapshots as defined above. Then we can define a vector $v^i = (v^i_1, \ldots, v^i_g)$ for each height $i\delta^2, i \in I^{>1/2}$, where $v^i_j \in \{0, \ldots, m\}$ is chosen such that $v^i_j \cdot 1/m$ is the sum of widths of all rectangles of height $i\delta^2$ contained in snapshot S_j. Obviously,

$$\sum_{i \in I^{>1/2}} \sum_{j=1}^{g} v^i_j \frac{1}{m} = \sum_{R_i \in L^{>1/2}} w_i \leq 1 \tag{3.11}$$

(otherwise the packing is not feasible) and thus, we have

$$v^i_j \leq m \tag{3.12}$$

for each $i \in I^{>1/2}, j \in \{1, \ldots, g\}$.

With a dynamic programming approach we can compute a list of all feasible vectors satisfying (3.11) and (3.12). A rough upper bound for the number of feasible vectors for each height $i\delta^2$ is given by $(m+1)^g$ since there are g components and every component $v^i_j \in \{0, \ldots, m\}$. The algorithm to calculate all feasible vectors for a given height $i\delta^2$ works as follows.

Assume that $L(i) = \{R_1, \ldots, R_{k_i}\}$. Starting with a set $V := \{(0, \ldots, 0)\}$ containing only the null vector we replace in step $l \in \{1, \ldots, k_i\}$ each vector $v \in V$ with all vectors that can be generated by adding $\gamma_l := w_l \cdot m$ to one of its components. To ensure that the number of vectors is bounded by $(m+1)^g$, we discard any vector that equals (componentwise) an already added vector. This can be done efficiently by keeping the list sorted (for example in

lexicographical order). Since in each step at most $g(m+1)^g$ vectors are generated and the number of operations used for the insertion (insertion sort with binary search) is bounded by $\log((m+1)^g) = g\log(m+1)$, the number of operations for each step is bounded by

$$g(m+1)^g \cdot g\log(m+1) \leq g^2(m+1)^{g+1}.$$

Thus, the number of operations for each height $i \in I^{>1/2}$ is bounded by

$$k_i \cdot g^2(m+1)^{g+1} \leq m \cdot g^2(m+1)^{g+1} \leq g^2(m+1)^{g+2},$$

since

$$k_i = |L(i)| \leq |L^{>1/2}| \leq m.$$

Let V^i denote the set of vectors generated for this height class $L(i)$. Obviously, the vector induced by a given packing can be found among the generated vectors.

Repeating this computation for every $i \in I^{>1/2}$ leads to $|I^{>1/2}| \leq 1/(2\delta^2)$ sets of at most $(m+1)^g$ vectors. We build the direct product $V := \bigtimes_{i \in I^{>1/2}} V^i$ of these sets. V contains at most

$$|V| \leq ((m+1)^g)^{|I|} \leq ((m+1)^g)^{\frac{1}{2\delta^2}} \tag{3.13}$$

elements and each of these elements consists of one vector for each height class. One element $v \in V$ corresponds to the vectors induced by the given packing. In our packing algorithm we guess an element $v \in V$ consisting of components $v^i, i \in I^{>1/2}$ and use these vectors v^i to pack the tall rectangles into the snapshots. Note that in practice we do not need the direct product, we can simply enumerate all elements in an arbitrary order. We use this notation only for convenience.

Using the dynamic program results in (many) vectors of widths only. However, for our packing algorithm we need to know what combination of rectangles leads to the given width per snapshot. This can be achieved by extending the dynamic program such that for each vector a component consists not only of the current width, but also of a set of rectangles. During the vector generation step, a rectangle is added to this set if its width is added to the corresponding width component. Due to this modification the space needed to store the vectors increases but is still polynomial in n; the running time of the dynamic program is not affected significantly.

In order to handle rectangles intersecting snapshot boundaries, we simply guess the subset $\hat{L} \subseteq L^{>1/2}$ of rectangles intersecting snapshot boundaries, which can be done in polynomial time since there are at most g of these rectangles. In order to pack these rectangles we

add them to L' (the set of K tall rectangles and containers). This modification increases the size of L' such that

$$|L'| \leq 3 \cdot (K + 2\delta^{-3}), \tag{3.14}$$

and of g such that

$$g \leq 6(K + 2\delta^{-3}). \tag{3.15}$$

Note that this modification does not increase the dimension of the vectors we defined above, since the snapshots introduced by these added rectangles obviously do not allow further $L^{>1/2}$ rectangles to be packed in them (height $> 1/2$).

As stated above, among all generated vectors there is one that is equivalent to the vector induced by a nearly optimal schedule with simpler structure. They are equivalent in the sense that the total width in each component is in both vectors the same. For our algorithm this is sufficient, since our packing algorithm ensures that all rectangles assigned to one component are packed next to each other (see Sections 3.4.4.1, 3.4.4.2).

3.4.3 Linear Program

In this subsection, we present a linear program (LP), which allows us to calculate the width of all snapshots, and thus determine the positions of all rectangles in $L' = \mathscr{C} \cup L'_{\text{ta}}$. We now assume that we have chosen a slot assignment f (see Section 3.4.2.1), functions α, β (see Section 3.4.2.2), and $v \in V$ consisting of vectors v^i of widths for each height class as described in Section 3.4.2.3.

Since all low-wide rectangles and a subset of the low-narrow rectangles get packed into the containers, we do not need to consider them in the LP. For convenience we call the subset of the low-narrow rectangles packed into the containers $L^C_{\text{lo-na}}$, and the remaining low-narrow rectangles $L_{\text{lo-na}}$. We construct $L^C_{\text{lo-na}}$ by greedily adding low-narrow rectangles as long as

$$\mathrm{A}(L^C_{\text{lo-na}}) + \mathrm{A}(L_{\text{lo-wi}}) \leq \mathrm{A}(\mathscr{C}). \tag{3.16}$$

We discard the first low-narrow rectangle that exceeds the total area in order to ensure that enough space can be reserved for the remaining rectangles in the following LP. This discarded rectangle has an area of at most δ^{2s}. In fact, we will show that this discarded rectangle can be packed along with the rectangles from $L_{\text{lo-na}}$ (see Section 3.4.4.5).

Since f, α, β are fixed, we can calculate the set of *free* slots (i.e. the slots not occupied by L'

rectangles) for each snapshot. These free slots will be used for the remaining tall rectangles and for the small rectangles from $L_{\text{lo-na}}$. In order to formulate constraints to ensure that enough space is reserved for these rectangles, we introduce configurations. We define a configuration as a pair (SN, Π) where SN is a subset of the free slots reserved for rectangles from $L_{\text{lo-na}}$ and Π is a partition of the remaining free slots into sets of consecutive slots reserved for rectangles from $L_{\text{ta}} \setminus L'$; every subset $F \in \Pi$ of cardinality $l = |F|$ is reserved to pack rectangles from L_{ta} of height $l\delta^2$. Let n_j denote the number of different configurations for each snapshot S_j and let $c_i^j := (\text{SN}_i^j, \Pi_i^j)$ denote the different configurations for snapshot S_j, $i \in \{1, \dots, n_j\}$ and let $n_i^j(\ell) := |\{F \in \Pi_i^j : |F| = \ell\}|$ denote the number of sets of cardinality ℓ in Π_i^j for each $\ell \in I_{\text{ta}}$. The total width of all rectangles in $L_{\text{ta}} \setminus L'$ of height ℓ is denoted as W_ℓ. The variables $x_i^j, j \in \{1, \dots, g\}, i \in \{1, \dots, n_j\}$ are used to determine the width of each configuration c_i^j. Additional variables $t_j, j \in \{1, \dots, g\}$ are used to determine the width of each snapshot S_j.

$$
\begin{aligned}
\text{LP}(f, \alpha, \beta, v): \quad & t_0 = 0, t_g \le 1 && && \\
& t_j \ge t_{j-1} && \forall j \in \{1, \dots, g\} && \\
& t_{\beta_j} - t_{\alpha_j} = w_j && \forall R_j \in L' && (3.17) \\
& \sum_{i=1}^{n_j} n_i^j(\ell) x_i^j \ge \frac{1}{m} v_j^\ell && \forall j \in \{1, \dots, g\}, \ell \in I^{>1/2} && (3.18) \\
& \sum_{j=1}^{g} \sum_{i=1}^{n_j} n_i^j(\ell) x_i^j \ge W_\ell && \forall \ell \in I_{\text{ta}} \setminus I^{>1/2} && (3.19) \\
& \sum_{j=1}^{g} \sum_{i=1}^{n_j} x_i^j |\text{SN}_i^j| \delta^2 \ge \text{A}(L_{\text{lo-na}}) && && (3.20) \\
& \sum_{i=1}^{n_j} x_i^j \le t_j - t_{j-1} && \forall j \in \{1, \dots, g\} && (3.21) \\
& x_1^j, \dots, x_{n_j}^j \ge 0 && \forall j \in \{1, \dots, g\} &&
\end{aligned}
$$

Constraint (3.17) ensures that the width of the snapshots corresponds to the width of the assigned pre-positioned rectangles or containers. Constraint (3.18) makes sure that the total width of all configurations in each snapshot is greater or equal than the width needed for packing the rectangles from $L^{>1/2}$ as given by the vector v. Similarly, constraint (3.19) ensures that the chosen configurations reserve enough space to pack all rectangles from $L_{\text{ta}} \setminus L'$ (at least fractionally). Constraint (3.20) ensures that enough space is reserved for (fractionally) packing the rectangles from $L_{\text{lo-na}}$. Constraint (3.21) makes sure that the

width of all configurations for a snapshot does not exceed the width of that snapshot.

Since $g, n_j, |L'|, |J|, |I|$ are independent of n, this linear program can be solved in polynomial time. If LP(f, α, β, v) has no feasible solution, we construct a new LP with a new combination of $\mathscr{C}, f, \alpha, \beta, v$.

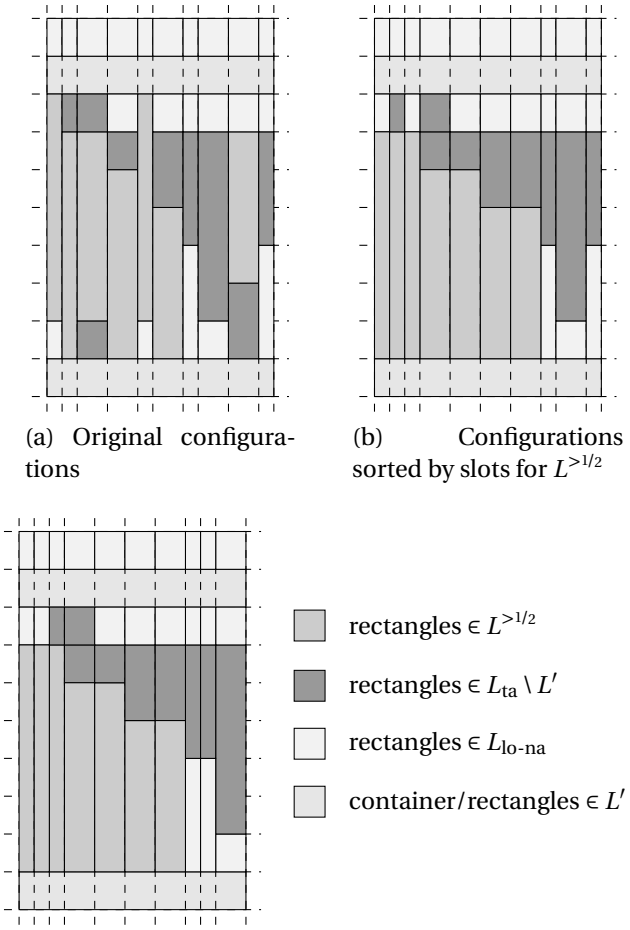
3.4.4 Packing the Rectangles

Let (t^*, x^*) be a feasible solution for LP(f, α, β, v). For simplicity, we remove all snapshots $[t_j^*, t_{j+1}^*)$ of zero width and combine all snapshots that do not contain any L' rectangles, i.e. the set of free slots is $F = M$ (remember that M corresponds to the set of all slots), as the last snapshot. Obviously the modified solution is still feasible. Let g^* denote the number of resulting snapshots.

Since we solve the LP fractionally, the solution might contain configurations with widths that are not multiples of $1/m$. Nevertheless, in the following we pack the rectangles using this fractional solution. If the resulting packing is not feasible, we can add a simple post-processing step in which we shift all rectangles beginning with the leftmost, bottommost infeasible rectangle R_i (position (x_i, y_i)), such that x_i is a multiple of $1/m$. This shifting is possible since all rectangles which are positioned left of R_i start at a feasible position and have a width that is a multiple of $1/m$. Repeating this shifting step for each infeasible rectangle leads to a feasible packing.

3.4.4.1 Adapting and Sorting the Configurations

Before we start packing the rectangles, we sort and modify the configurations inside each snapshot (see Figure 3.6). The objective is to make sure that on the one hand no rectangles from $L^{>1/2}$ get split and, on the other hand that the fragmentation of the slots reserved for the low-narrow rectangles is *limited*. In each snapshot, we first sort all configurations based on the number of slots reserved for rectangles from $L^{>1/2}$. After this sorting, all configurations reserved for rectangles from $L^{>1/2}$ of the same height appear next to each other. Note that the sorting is well-defined, since in each configuration at most one subset of contiguous slots is reserved for rectangles from $L^{>1/2}$, and they cannot appear on top of each other (height $> (1+2\varepsilon')/2$). Furthermore, we need to modify the configurations such that the slots reserved for rectangles from $L^{>1/2}$ of the same height are the same. In each snapshot all configurations cover the same subset of slots (all slots save the slots occupied by the pre-positioned rectangles from L') and obviously, there is at most one contiguous subset Π' of these covered slots that can contain slots reserved for rectangles from $L^{>1/2}$. Now we modify each configuration that contains a subset $\Pi'' \subseteq \Pi'$ reserved for rectangles from $L^{>1/2}$, by



(a) Original configurations

(b) Configurations sorted by slots for $L^{>1/2}$

(c) Configurations sorted by slots for $L^{>1/2}$ and $L_{\text{lo-na}}$

Figure 3.6: Adapting configurations

shifting Π'' *inside* Π' as far down as possible. The resulting configuration does not change the solution of our LP, since no assumptions about the locations of the reserved slots are made. But now packing the $L^{>1/2}$ rectangles next to each other will not lead to splittings (see Figure 3.6).

To limit the fragmentation of the slots reserved for low-narrow rectangles, we place configurations with the same set SN_i^j of slots reserved for low-narrow rectangles next to each other, but without disturbing the previous sorting.

3.4.4.2 Packing Pre-Positioned Rectangles

Each rectangle $R_i \in L'$ is placed in the slots assigned by function f such that its left side is at distance $t^*_{\alpha_i}$ from the left side of the strip, i.e. the position (lower left corner) for R_i is given by $(t^*_{\alpha_i}, f(R_i)\delta^2)$. In particular, no rectangle from L' is split in this process.

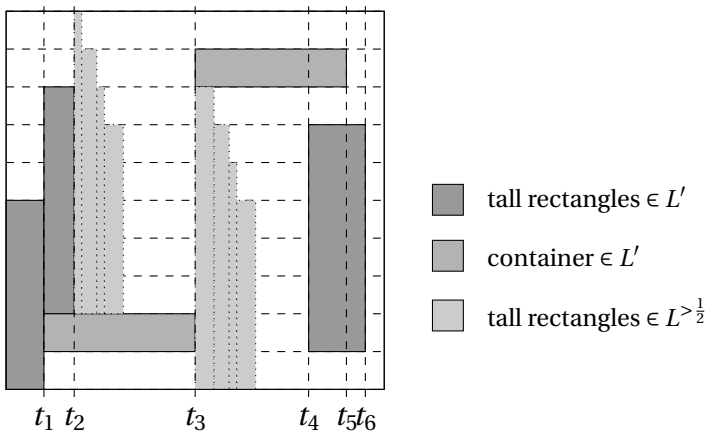


Figure 3.7: Packing of pre-positioned rectangles

The next step is to pack the $L^{>1/2}$ rectangles. Since the solution is feasible, in each snapshot S_j the widths of the configurations that are reserved for the $L^{>1/2}$ rectangles are at least as large as the widths given by vector v^j (see constraint (3.18)). Furthermore, due to the ordering of the configurations within each snapshot, we can simply pack all these rectangles next to one other according to the configurations. Due to the ordering described in the previous subsection (Section 3.4.4.1), the packing of the tall rectangles and the containers has a structure as in Figure 3.7.

3.4.4.3 Tall Rectangles

The next step is to pack the remaining tall rectangles. Let $\mathscr{R}_l = \{R_{l,1}, \ldots, R_{l,n_l}\} = L(l) \setminus L' \subseteq L_{\text{ta}} \setminus L'$ be the rectangles of height $l\delta^2$ for every $l \in I_{\text{ta}} \setminus I^{>1/2}$. Take the first configuration $c_i^j = (\text{SN}_i^j, \Pi_i^j)$ in the above ordering with $x_i^j > 0$ and select for each set $X \in \Pi_i^j$ with $l = |X|$ successively the first not yet completely packed rectangle $R \in \mathscr{R}_l$. These rectangles are packed within the slots X starting at position $x(c_i^j)$ until their total width is at least x_i^j or all rectangles in $\mathscr{R}_l$ are packed. If the total width is greater than x_i^j the last rectangle is split such that the width is exactly x_i^j. Repeating the packing process for each configuration in each snapshot leads to a fractional packing of all tall rectangles, since (3.19) ensures that there is sufficient space reserved for them.

This fractional packing of tall rectangles allows a certain number of tall rectangles to get split. In the following, we show that the number of these split rectangles is bounded and that the total area of these rectangles is at most δ.

The splitting of rectangles is caused by the transition from one subset X of slots to the next as described above. Thus, the number of split rectangles is bounded by the number

of subsets of slots in each configuration times the number of configurations per snapshot times the number of snapshots. The number of subsets of slots in each configuration is bounded by $|M|$ (M corresponds to the set of all slots). For each snapshot there are at most $(2|M|)^{|M|}$ different configurations, since the number of all subsets of the set of slots is $2^{|M|}$ and the number of partitions of M is bounded by $|M|^{|M|}$. In total, this leads to at most $g^*|M|(2|M|)^{|M|}$ divided rectangles. Note that the number of chosen configurations ($x_i^j \neq 0$) is bounded by the number of constraints in the LP. Thus, the number of split rectangles is possibly much lower. Since $g^* \leq g \leq 6(K+2\delta^{-3})$ (see Equation (3.15)), the number of split rectangles is bounded by

$$g^*|M|(2|M|)^{|M|} \leq 6(K+2\delta^{-3})|M|(2|M|)^{|M|}. \tag{3.22}$$

With this bound for the number of split tall rectangles, we only have to choose the constant K such that the total area of split tall rectangles is bounded by 2δ, which will be done in the next subsection.

3.4.4.4 Choosing Constant K

In order to choose a constant K such that the total area of the split tall rectangles is bounded, we use a slightly modified version of a result by Jansen and Porkolab [30, Lemma 2.5].

Lemma 3.4.6 ([30])**.** *Suppose $d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$ is a sequence of real numbers and $D = \sum_{j=1}^n d_j$. Let p, q be nonnegative integers, $\alpha > 0$, and assume that $n > (\lceil \frac{1}{\alpha} \rceil p + 1)(q+1)^{\lceil \frac{1}{\alpha} \rceil}$. Then, there exists an integer $k = k(p,q,\alpha)$ such that*

$$d_k + \cdots + d_{k+p+qk-1} \leq \alpha D$$

and

$$k \leq (q+1)^{\lceil \frac{1}{\alpha} \rceil - 1} + p\left(1 + (q+1) + \cdots + (q+1)^{\lceil \frac{1}{\alpha} \rceil - 2}\right). \tag{3.23}$$

Proof. Decompose the sum $d_1 + \cdots + d_n$ into blocks $B_0 = d_1 + \cdots + d_{f(1)-1}, B_1 = d_{f(1)} + \cdots + d_{f(2)-1}, \ldots B_i = d_{f(i)} + \cdots + d_{f(i+1)-1}$, where the function f is defined recursively by the following equation:

$$f(0) = 1, \quad f(i+1) = f(i) + p + q \cdot f(i). \tag{3.24}$$

Since $\sum_{j=1}^n d_j = D$, at most $\lceil \frac{1}{\alpha} \rceil - 1$ blocks are larger in size than $\alpha \cdot D$. Now let i be the smallest integer for which $B_i \leq \alpha D$. Then $i \leq \lceil \frac{1}{\alpha} \rceil - 1$, and $B_i = d_{f(i)} + \cdots + d_{f(i+1)-1} \leq \alpha \cdot D$. This implies that there is an index $k \leq f(i)$ such that $d_k + \cdots + d_{k+p+qk-1} \leq \alpha \cdot D$. It follows

from (3.24) that

$$f(i) = (q+1)^i + p\left(1 + (q+1) + \cdots + (q+1)^{i-1}\right), \tag{3.25}$$

which along with the bound on i implies (3.23). □

We choose $d_j = w_j \cdot h_j$ for each $R_j \in L_{\text{ta}}$ (sorted by non-increasing area), and define $\alpha := \delta$, and $p := 6|M|2\delta^{-3}(2|M|)^{|M|}$, and $q := 6|M|(2|M|)^{|M|}$. Then $D \leq 1+2\delta$, since $\mathrm{A}(L) \leq 1+2\delta$ (see Section 3.4.1.5). If $|L_{\text{ta}}| \leq (\lceil \frac{1}{\alpha} \rceil p + 1)(q+1)^{\lceil \frac{1}{\alpha} \rceil}$, we can add *dummy* rectangles with area 0. Note that the algorithm would also work without this modification, since the number of tall rectangles would be constant in this case and thus all tall rectangles could be pre-positioned. However, this would make the following proofs more complicated.

Lemma 3.4.6 yields that there exists a constant K such that for each set $\hat{L} \subseteq L_{\text{ta}} \setminus \tilde{L}$ with $|\hat{L}| \leq p + qK$, the total area of $\hat{L}$ is bounded by $|\hat{L}| \leq \alpha \cdot D \leq \delta(1+2\delta) = \delta + 2\delta^2 \leq 2\delta$, if $\tilde{L} \subseteq L_{\text{ta}}$ contains the K rectangles with largest area. Furthermore,

$$K \leq (1+2\delta^{-3})(6|M|(2|M|)^{|M|})^{\frac{1}{\delta}-1}, \tag{3.26}$$

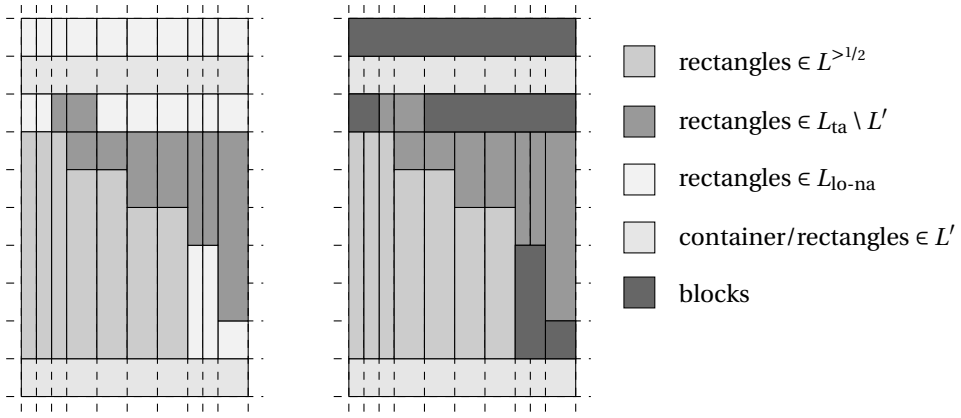
since

$$\begin{aligned}
k &\leq (q+1)^{\lceil \frac{1}{\alpha} \rceil - 1} + p\left(1 + (q+1) + \cdots + (q+1)^{\lceil \frac{1}{\alpha} \rceil - 2}\right) \\
&= (q+1)^{\lceil \frac{1}{\alpha} \rceil - 1} + p\frac{(q+1)^{\lceil \frac{1}{\alpha} \rceil - 1} - 1}{(q+1) - 1} \\
&\leq (q+1)^{\lceil \frac{1}{\alpha} \rceil - 1} + \frac{p}{q}(q+1)^{\lceil \frac{1}{\alpha} \rceil - 1} \\
&= (1 + \frac{p}{q})(q+1)^{\lceil \frac{1}{\alpha} \rceil - 1} \\
&= (1 + \frac{6|M|2\delta^{-3}(2|M|)^{|M|}}{6|M|(2|M|)^{|M|}})(6|M|(2|M|)^{|M|})^{\frac{1}{\delta}-1} \\
&= (1+2\delta^{-3})(6|M|(2|M|)^{|M|})^{\frac{1}{\delta}-1}.
\end{aligned}$$

Since the number of split rectangles is at most

$$\begin{aligned}
g|M|(2|M|)^{|M|} &\leq 6|M|(K + 2\delta^{-3})(2|M|)^{|M|} \\
&= 6|M|2\delta^{-3}(2|M|)^{|M|} + 6|M|(2|M|)^{|M|}K \\
&= p + qK
\end{aligned}$$

and the K rectangles with largest profit (L'_{ta}) are not split (see Section 3.4.4.2), Lemma 3.4.6 yields that the total area of split rectangles is at most 2δ.



(a) Sorted configurations

(b) Reserved space combined to blocks

Figure 3.8: Blocks

3.4.4.5 Packing the Low-Narrow Rectangles

The next step is to pack the subset $L_{\text{lo-na}}$ of the low-narrow rectangles that are not assigned to the containers. Due to the ordering, configurations c_i^j with the same set SN_i^j of slots reserved for low-narrow rectangles are adjacent (if possible). In the following, we combine adjacent reserved slots into blocks. Then, if we pack the $L_{\text{lo-na}}$ rectangles only into blocks of width at least $4\delta^{s-3}$, we can pack almost all rectangles, i.e. the remaining rectangles have a total area of at most δ. To be more specific, in each snapshot we define blocks $B_1, \ldots, B_l$ by combining all adjacent subsets $Y \subseteq M$ reserved for low-narrow rectangles $L_{\text{lo-na}}$ that occur in adjacent configurations (see Figure 3.8). For example assume that a set of adjacent slots $Y \subseteq M$ occurs in adjacent configurations $c_i^j, c_i^{j+1}, c_i^{j+2}$, that is $Y \in \text{SN}_i^j = \text{SN}_i^{j+1} = \text{SN}_i^{j+2}$. Then we combine these reserved regions into a block B_k with width $x_i^j + x_i^{j+1} + x_i^{j+2}$ and height $|Y|\delta^2$. Then each block is a rectangular region and the height of each block is a multiple of δ^2.

Let B be a block with height $d\delta^2$ and width b. We select low-narrow rectangles to be packed into this block by adding rectangles to a set S until the total area of S is at least $d\delta^2 b$. Since each small rectangle has area at most δ^{2s} the total area of S is bounded by $d\delta^2 b + \delta^{2s}$. We pack the small rectangles into the block using the NFDH (Next Fit Decreasing Height) algorithm introduced by Coffman et al. [12]. We pack in each block with width at least $4\delta^{s-3}$ a subset $S' \subseteq S$ with $\text{A}(S') \geq \text{A}(S) - \delta\,\text{A}(S)$ (see Figure 3.9), since

$$\text{A}(S') \geq \sum_{i=2}^{n^{(S)}} h_i(b - \delta^s)$$

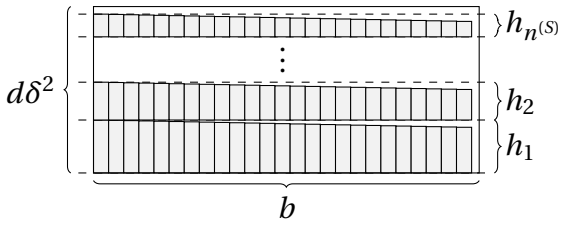


Figure 3.9: NFDH for packing blocks

$$
\begin{aligned}
&\geq (b-\delta^s)\underbrace{\sum_{i=1}^{n^{(S)}} h_i}_{\geq d\delta^2-\delta^s} - \underbrace{h_1}_{\leq \delta^s} \\
&\geq (b-\delta^s)\big((d\delta^2-\delta^s)-\delta^s\big) \\
&= (b-\delta^s)\big(d\delta^2-2\delta^s\big) \\
&= d\delta^2 b - b2\delta^s - d\delta^2\delta^s + 2\delta^{2s} \\
&= \underbrace{d\delta^2 b+\delta^{2s}}_{=\mathrm{A}(S)} + \delta^{2s} - b2\delta^s - d\delta^{s+2} \\
&= \mathrm{A}(S) + \delta^{2s} - (b2\delta^s + d\delta^{s+2}) \qquad (3.27) \\
&\geq \mathrm{A}(S) - (2+\underbrace{d\delta^2}_{\leq 1+2\delta})\delta^s \\
&\geq \mathrm{A}(S) - (\underbrace{2+1+2\delta}_{\leq 4})\delta^s \\
&\geq \mathrm{A}(S) - 4\delta^s \\
&\geq \mathrm{A}(S) - \delta(\underbrace{\delta^2}_{\leq d\delta^2}\underbrace{4\delta^{s-3}}_{\leq b}) \\
&\geq \mathrm{A}(S) - \delta\,\mathrm{A}(S),
\end{aligned}
$$

where h_i denotes the height of ith level generated by NFDH and $n^{(S)}$ denotes the total number of levels generated by NFDH.

To take care of the discarded small rectangle (while partitioning the low-narrow rectangles, see Section 3.4.3), we add this rectangle to one set S without changing the bound $\delta\,\mathrm{A}(S)$ for the total area of the unpacked rectangles (see Equation (3.27)).

Thus, after packing all blocks, the total area of the unpacked low-narrow rectangles is bounded by $\delta\,\mathrm{A}(L_{\text{lo-na}}) \leq \delta$.

Area lost by discarding small blocks. While packing the small boxes we discarded all blocks with width smaller than $4\delta^{s-3}$. However, the area lost by discarding these blocks

is bounded by δ.

Lemma 3.4.7. *The total area of all blocks with width $< 4\delta^{s-3}$ is bounded by δ.*

Proof. Let us first note that there are at most $g^*(\frac{1}{2\delta^2}+1)2^{|M|}|M|$ blocks; this bound holds because for a fixed subset of free slots, there are at most $\frac{|M|}{2} \leq |M|$ blocks. Furthermore, there are at most $2^{|M|}$ different subsets of the free slots. (Remember that we combined subsets of free slots if the sets SN_i^j are equal for adjacent configurations.)

Due to the sorting of the configurations (see Section 3.4.4.1), configurations with equal sets SN_i^j are adjacent for each snapshot and each height class. Thus, the above bound for the number of blocks holds for each snapshot ($\leq g^*$) and for each height class ($\leq \frac{1}{2\delta^2}+1$). In total the number of blocks is bounded by

$$g^*(\frac{1}{2\delta^2}+1)2^{|M|}|M|.$$

Even if we assume that all blocks have width smaller than $4\delta^{s-3}$ and that each block has height 1, the total area of the discarded blocks is bounded by $(4\delta^{s-3})g^*(\frac{1}{2\delta^2}+1)2^{|M|}|M| \leq \delta$.

It holds that

$$\begin{aligned}
K &\leq (1+2\delta^{-3})(6|M|(2|M|)^{|M|})^{\frac{1}{\delta}-1} \\
&\leq 3\delta^{-3}(6|M|(2|M|)^{|M|})^{\frac{1}{\delta}-1} \\
&\leq 3\delta^{-3}\left(6\frac{1+2\delta}{\delta^2}\left(2\frac{1+2\delta}{\delta^2}\right)^{\frac{1+2\delta}{\delta^2}}\right)^{\frac{1}{\delta}-1} \\
&\leq 3\delta^{-3}\left(6\frac{2}{\delta^2}\left(2\frac{2}{\delta^2}\right)^{\frac{2}{\delta^2}}\right)^{\frac{1}{\delta}-1} \\
&= 3\delta^{-3}\left(\frac{12}{\delta^2}\left(\frac{4}{\delta^2}\right)^{\frac{2}{\delta^2}}\right)^{\frac{1}{\delta}}\left(\frac{12}{\delta^2}\left(\frac{4}{\delta^2}\right)^{\frac{2}{\delta^2}}\right)^{-1} \\
&\leq 3\delta^{-3}2^{\frac{4}{\delta}}\delta^{-\frac{2}{\delta}}\left(\frac{2^2}{\delta^2}\right)^{\frac{2}{\delta^3}}2^{-3}\delta^2\left(\frac{2^2}{\delta^2}\right)^{-\frac{2}{\delta^2}} \\
&= 3\delta^{2-3-\frac{2}{\delta}}2^{\frac{4}{\delta}-3}\left(\frac{2}{\delta}\right)^{\frac{4}{\delta^3}-\frac{4}{\delta^2}} \\
&= 3\delta^{-1-\frac{2}{\delta}-\frac{4}{\delta^3}+\frac{4}{\delta^2}}2^{\frac{4}{\delta}-3+\frac{4}{\delta^3}-\frac{4}{\delta^2}} \\
&= 3\delta^{-(\overbrace{\frac{\delta^3+2\delta^2+4-4\delta}{\delta^3}}^{\leq 4,\ \text{since } 4\delta\geq\delta^3+2\delta^2})}2^{\overbrace{\frac{4\delta^2-3\delta^3+4-4\delta}{\delta^3}}^{\leq 4}} \\
&\leq 3\delta^{-\frac{4}{\delta^3}}2^{\frac{4}{\delta^3}}
\end{aligned}$$

and thus

$$\begin{aligned}4(K+2\delta^{-3}) &\leq 4(3\delta^{-\frac{4}{\delta^3}}2^{\frac{4}{\delta^3}}+2\delta^{-3})\\ &\leq 4(4\delta^{-\frac{4}{\delta^3}}2^{\frac{4}{\delta^3}})\\ &= \delta^{-\frac{4}{\delta^3}}2^{\frac{4}{\delta^3}+4}.\end{aligned}$$

Using these inequalities we conclude

$$\begin{aligned}4\delta^{s-3}g^*(\frac{1}{2\delta^2}+1)2^{|M|}|M| &\overset{\text{def } M,g^*}{\leq} 4\delta^{s-3}(4(K+2\delta^{-3}))(\frac{1}{2\delta^2}+1)2^{\frac{1+2\delta}{\delta^2}}\frac{1+2\delta}{\delta^2}\\ &\overset{\delta\leq\frac{1}{4}}{\leq} 4\delta^{s-3}(\delta^{-\frac{4}{\delta^3}}2^{\frac{4}{\delta^3}+4})(\frac{1}{\delta^2})2^{\frac{2}{\delta^2}}\frac{2}{\delta^2}\\ &= 2^3\delta^{s-3-\frac{4}{\delta^3}-2-2}2^{\frac{4}{\delta^3}+\frac{2}{\delta^2}+4}\\ &= \delta^{s-(7+\frac{4}{\delta^3})}2^{7+\frac{4}{\delta^3}+\frac{2}{\delta^2}}\\ &= \delta^{s-(\frac{\overbrace{7\delta^3+4}^{\leq 5}}{\delta^3})}2^{\frac{\overbrace{7\delta^3+4+2\delta}^{\leq 5}}{\delta^3}}\\ &\leq \delta^{s-\frac{5}{\delta^3}}2^{\frac{5}{\delta^3}}\\ &\overset{\text{def } s}{=} \delta^{\frac{8}{\delta^3}-\frac{5}{\delta^3}}2^{\frac{5}{\delta^3}}\\ &= \delta^{\frac{3}{\delta^3}}2^{\frac{5}{\delta^3}}\\ &= \delta^{\frac{1}{\delta^3}}\delta^{\frac{2}{\delta^3}}2^{\frac{5}{\delta^3}}\\ &\overset{\delta\leq\frac{1}{14}\leq 2^{-3}}{\leq} \delta^{\frac{1}{\delta^3}}2^{-\frac{6}{\delta^3}}2^{\frac{5}{\delta^3}}\\ &\leq \underbrace{\delta^{\frac{1}{\delta^3}}}_{\leq\delta}\underbrace{2^{-\frac{1}{\delta^3}}}_{\leq 1}\\ &\leq \delta.\end{aligned}$$

□

3.4.4.6 Packing Containers

In the following, we describe how to pack the remaining low-narrow rectangles $L^C_{\text{lo-na}}$ and the low-wide rectangles $L_{\text{lo-wi}}$ into the containers.

Assume that we have chosen the *right* set of containers $\mathscr{C}$, that is, the set of containers corresponds to the set induced by an optimal packing. If we have not chosen the *right* set, packing the remaining low-narrow rectangles $L^C_{\text{lo-na}}$ and the low-wide rectangles $L_{\text{lo-wi}}$ into the containers might not be possible. In this case, we restart with another set of containers.

Unfortunately, some low-wide rectangles might intersect two containers. To ensure that

all low-wide rectangles are packable into containers, we increase the height of each container by δ^s. This is sufficient, since the height of all low rectangles is bounded by δ^s. In the following lemma, we prove that the mKR algorithm (see Section 3.3) can be used to pack nearly all rectangles into the containers.

Lemma 3.4.8. *Nearly all rectangles from $L_{lo\text{-}wi}$ and $L^C_{lo\text{-}na}$ can be packed into the containers, i.e. the total area of unpacked rectangles is bounded by δ.*

Proof. It is sufficient to prove that the assumptions (A1) – (A6) of Theorem 3.3.1 are fullfilled.

Let $L_{\text{wi}} := L_{\text{lo-wi}}, L_{\text{sm}} := L^C_{\text{lo-na}}$. Since we increased the height of all containers by δ^s, all rectangles from $L_{\text{lo-wi}}$ are packable inside the containers; consequently, (A1) holds.
Furthermore, $L^C_{\text{lo-na}}$ was chosen such that

$$\mathrm{A}(L^C_{\text{lo-na}}) + \mathrm{A}(L_{\text{lo-wi}}) \leq \mathrm{A}(\mathscr{C}) \qquad \text{(see Equation (3.16))}.$$

Thus, Assumption (A2) is fulfilled. Assumption (A3) is fullfilled since the width of each rectangle in $L_{\text{lo-wi}}$ is at least δ.

To show (A4) – (A6), it is sufficient to show (A6), since $h^s_{\max}, h^w_{\max}, w^s_{\max} \leq \gamma = \delta^s$ and the right hand side of (A6) is the strictest. It holds that

$$\begin{aligned}
h^s_{\max} \leq \delta^s = \delta^{\frac{8}{\delta^3}} &= \delta^2 \cdot \delta^2 \delta^3 \underbrace{\delta^{\frac{1}{\delta^3}}}_{\leq 1} \\
&\overset{\delta \leq \frac{1}{7}}{\leq} \frac{\delta}{7} \cdot \frac{\delta^3}{7 \cdot 7} \\
&\leq \frac{\delta}{7} \min\left\{ \frac{\delta^3}{4 \cdot 2}, \frac{\delta}{4 \cdot 7} \delta \right\} \\
&\overset{|\mathscr{C}| \leq \frac{2}{\delta^3}, w^w_{\min} \geq \delta}{\leq} \frac{\delta}{7} \min\left\{ \frac{1}{4|\mathscr{C}|}, \frac{\delta}{4 \cdot 7} w^w_{\min} \right\}.
\end{aligned}$$

□

Hence, the mKR algorithm allows us to pack almost all of the low-narrow rectangles from $L^C_{\text{lo-na}}$ and $L_{\text{lo-wi}}$, in particular the total area of the unpacked rectangles is bounded by δ. But since we increased the height of the containers, some of the low rectangles might intersect with other rectangles. However, we can move all intersecting rectangles to the top of the strip at the cost of an increase in height by at most $2\delta^s \cdot (1+2\delta)/\delta^2 \leq \delta$.

3.4.5 Analysis of the Algorithm

In the following, we summarize the approximation algorithm for the non-malleable, contiguous case, $P\text{poly}|\text{line}_j|C_{\max}$ (see Algorithm 1 for pseudo-code).

We first guess (enumerate) the length of the optimal schedule. The next step in our algorithm is to create a gap between tall and low rectangles and between wide and narrow rectangles. In this process, we discard all *middle-sized* rectangles. The total area of these rectangles is bounded by $\varepsilon' =: A_1$ (see Section 3.4.1.2). In the following, we accept a slightly increased height of the solution by rounding and shifting all tall rectangles. This additional height is bounded by $2\delta =: h_1$, as was shown in Section 3.4.1.3.

Then we guess (enumerate) the set of containers and we guess/enumerate a slot assignment and a snapshot assignment for L', where L' contains all containers and a subset of the tall rectangles (or all tall rectangles if $|L_{\text{ta}}| \leq K$). With a dynamic program we construct a set V of all distinguishable assignments of rectangles from $L^{>1/2}$ to snapshots. After choosing one assignment $v \in V$ we set up a LP. If the LP has no solution, we try the next combination of containers, slot assignment, snapshot assignment and vector $v \in V$. Otherwise, we start the actual packing of the rectangles beginning with the subset of tall rectangles and containers L' and the rectangles from $L^{>1/2}$. All of these rectangles can be packed (see Sections 3.4.4.2, 3.4.4.3). The remaining tall rectangles are packed fractionally according to the solution of the LP. By removing all split tall rectangles, we discard rectangles with total area bounded by $2\delta =: A_2$ (see Section 3.4.4.4).

For packing low-narrow rectangles into the space reserved for them by the LP, we use an approximation algorithm. The low-narrow rectangles that are not packed by this algorithm and the discarded blocks have total area bounded by $2\delta =: A_3$ (see Section 3.4.4.5).

The next step is to pack the rectangles assigned to the containers (see Section 3.4.4.6). Note that this step is not always successful if we have chosen the *wrong* set of containers. In case of failure, we try another set of containers. We increased the height of each container such that all rectangles fit into the containers. Shifting all intersecting rectangles to the top of the strip increases its height by at most $\delta =: h_2$, since there are at most $(1+2\delta)/\delta^2$ slots and the height of each intersected rectangle is bounded by δ^s. Furthermore, rectangles with total area bounded by $\delta := A_4$ are not packed by mKR.

In total, we packed almost all rectangles into a strip of height $1 + h_1 + h_2 = 1 + 3\delta$. We add the discarded rectangles using the NFDH algorithm by Coffman et al. [12]. This leads to an additional strip with height bounded by

$$2 \cdot (A_1 + A_2 + A_3 + A_4) + h_3 \leq 2\varepsilon' + 9\delta + \frac{(1+2\varepsilon')}{2},$$

Algorithm 1: Algorithm for $P\text{poly}|\text{size}_j|C_{\max}$

Input: Set of jobs $L = \{R_i \mid i \in \{1,\dots,n\}\}$, and precision ε
Output: A schedule S with $h(S) \leq (1.5+\varepsilon)\,\text{OPT}$

```
/* see Section 3.4.1.1 */
```
Let v be the height of the solution generated by 2-approximation
foreach $v^* \in \{(1+0\varepsilon)\frac{v}{2}, (1+1\varepsilon)\frac{v}{2}, \dots, (1+\lceil\frac{1}{\varepsilon}\rceil\varepsilon)\frac{v}{2}\}$ **do**
  `/* see Section 3.4.1.2 */`
  Set ε' such that $\varepsilon' = \frac{1}{2a}$ for an integer a and such that $\varepsilon' \leq \frac{\varepsilon}{14}$
  Find δ
  Set $\gamma \leftarrow \delta^s$
  Partition L into $L^{>1/2}, L_{\text{ta}}, L_{\text{lo}}, L_{\text{wi}}, L_{\text{na}}$
  `/* see Section 3.4.1.3 */`
  Round up h_i to the next multiple of δ^2 for all $R_i \in L_{\text{ta}}$
  Set $M \leftarrow \frac{1+2\delta}{\delta^2}$
  Set $K \leftarrow (1+2\delta^{-3})(6|M|(2|M|)^{|M|})^{\frac{1}{\delta}-1}$
  Let $L_K \subseteq L_{\text{ta}} \setminus L^{>1/2}$ be the subset of K tall rectangles with largest area
  `/* see Section 3.4.2 */`
  foreach *choice of containers* $\mathscr{C}$ **do**
    if $L_{lo\text{-}wi}$ *are nearly packable into* $\mathscr{C}$ **then**
      Set $L' \leftarrow \mathscr{C} \cup L_K$
      foreach *slot assignment* f **do**
        foreach *snapshot assignment* α, β **do**
          Calculate V by dynamic program `/* see Section 3.4.2.3 */`
          foreach $v \in V$ **do**
            Solve LP `/* see Section 3.4.3 */`
            if *LP has a solution* **then**
              `/* see Section 3.4.4 */`
              Adapt and Sort configurations `/* 3.4.4.1 */`
              Pack pre-positioned rectangles `/* 3.4.4.2 */`
              Pack low-narrow rectangles `/* 3.4.4.5 */`
              Pack containers (if possible) `/* 3.4.4.6 */`
              Save solution (if it exists)
Choose schedule with minimal length

where $h_3 \leq {}^{(1+2\varepsilon')}/_2$ is the height of the tallest rectangle among all discarded rectangles. Thus, all rectangles can be packed into a strip with height bounded by

$$\begin{aligned}(1+3\delta) + (2\varepsilon' + 9\delta + {}^{(1+2\varepsilon')}/_2) &\leq 1 + \frac{1}{2} + 3\varepsilon' + 12\delta \\ &\leq 1 + \frac{1}{2} + 15\varepsilon' \leq 1 + \frac{1}{2} + \varepsilon.\end{aligned}$$

As already mentioned in Section 3.4.1.1 (see Equation (3.1)), rescaling yields Theorem 3.1.1.

The running time of the algorithm is in $\mathcal{O}(n^{f(\frac{1}{\varepsilon})})$ for some (super-exponential) function f.

3.5 Non-Contiguous Parallel Job Scheduling

In this section, we study the problem $P\text{poly}|\text{size}_j|C_{\max}$. In this problem, the indices of the machines allotted to a job are not required to be contiguous. In the following, we construct a polynomial time approximation scheme (PTAS) for this case. First we show the existence of a nearly optimal schedule with simple structure. Therefore, we guess the length of an optimal schedule and scale the instance such that the height of an optimal solution for the scaled instance is bounded by 1 (see Section 3.4.1.1). Instead of the 2-approximation algorithm for the strip packing problem, we use a 2-approximation algorithm presented by Garey & Graham [25] for resource-constrained scheduling. We partition the jobs into tall, low-narrow, and low-wide jobs. Again, we reduce the search space by rounding and shifting the tall jobs in the same manner as before (see Section 3.4.1.3).

The actual algorithm for this case works as follows. We use a dynamic program to find a distribution of the tall jobs among the slots. Then we schedule the tall jobs according to the distribution in a canonical way. The remaining space is merged into one container per slot. We schedule the low jobs by packing them into these containers using the mKR algorithm. Then, creating a feasible schedule can be done by a simple greedy algorithm. In contrast to the previous, case we do not guess the structure of the containers. The structure is given automatically after assigning the tall jobs.

Again we use the notations job/rectangle and schedule/packing synonymously, although rectangles might be misleading in this case, since horizontal fragmentation is allowed; the height of a rectangle h_i corresponds to the length (processing time) p_i of a job (i.e. $p_i = h_i$) and the width w_i of a rectangle corresponds to the number of required machines q_i of a job divided by m (i.e. $w_i = {}^{q_i}/_m$).

3.5.1 Simple Structure

Again, the first step is to show the existence of a nearly optimal solution with simple structure.

3.5.1.1 Bounded Height

Since we want to divide the solution into a constant number of *slots*, we need to know the height of an optimal solution, at least up to the required accuracy ε. By using the 2-approximation algorithm by Garey & Graham [25], we can find a solution with height $\nu \leq 2 \cdot \mathrm{OPT}$, where OPT is the height of an optimal solution. Again, there exists a value

$$\nu^* \in \{(1+0\varepsilon)\nu/2, (1+1\varepsilon)\nu/2, \ldots, (1+\lceil 1/\varepsilon \rceil \varepsilon)\nu/2\}$$

such that $\mathrm{OPT} \leq \nu^* \leq (1+\varepsilon)\,\mathrm{OPT}$. Therefore, we only have to consider $\lceil 1/\varepsilon \rceil + 1$ different candidates to find the right one. For simplicity, we divide the height of each job by ν^* such that the height $\mathrm{OPT}' := \frac{\mathrm{OPT}}{\nu^*}$ of an optimal solution for the scaled instance satisfies

$$1-\varepsilon < \frac{1-\varepsilon}{1-\varepsilon^2} = \frac{1}{1+\varepsilon} = \frac{\mathrm{OPT}}{(1+\varepsilon)\,\mathrm{OPT}} \leq \frac{\mathrm{OPT}}{\nu^*} \leq 1.$$

3.5.1.2 Creating a Gap

Again we create a gap in size between tall and low and between wide and narrow jobs by discarding *middle-sized* jobs. These discarded jobs will be scheduled in a post-processing step.

Let ε denote the requested accuracy and let $\varepsilon' \leq \varepsilon/9$ be the largest value of the form $\varepsilon' = 1/a$ for some integer value a. Let $\sigma_0 := 1, \sigma_1 := \varepsilon'$, and $\sigma_k := \sigma_{k-1}^3/(4\cdot 7^2)$ for all $k \geq 2$. Define

$$L_k := \left\{ R_i \in L \;\middle|\; h_i \in (\sigma_k, \sigma_{k-1}] \text{ or } w_i \in (\sigma_k, \sigma_{k-1}] \right\}.$$

By Lemma 3.4.1 there exists $k \in \{2, \ldots, 2/\varepsilon' + 1\}$ such that $A(L_k) \leq \varepsilon' A(L)$. Choose the smallest value $k \in \{2, \ldots, 2/\varepsilon' + 1\}$ with this property, and let $\delta := \sigma_{k-1}$, and $\gamma := \sigma_k = \delta^3/(4\cdot 7^2)$. Define

$$\begin{aligned}
L_{\text{ta}} &:= \left\{ R_i \in L \;\middle|\; h_i > \delta \right\} && \text{tall jobs} \\
L_{\text{lo-wi}} &:= \left\{ R_i \in L \;\middle|\; h_i \leq \gamma, w_i > \delta \right\} && \text{low-wide jobs} \\
L_{\text{lo-na}} &:= \left\{ R_i \in L \;\middle|\; h_i \leq \gamma, w_i \leq \gamma \right\} && \text{low-narrow jobs.}
\end{aligned}$$

In the following, we present an algorithm to schedule $L_{\text{ta}}, L_{\text{lo-wi}}$, and $L_{\text{lo-na}}$. The remaining

rectangles will be scheduled in a post-processing step using a greedy algorithm. This is possible since

$$\mathrm{A}(L \setminus (L_{\text{ta}} \cup L_{\text{lo-wi}} \cup L_{\text{lo-na}})) = \mathrm{A}(L_k) \leq \varepsilon'.$$

3.5.1.3 Shifting and Rounding

Analogous to the previous case, we can round up the height of the jobs to the next multiple of δ^2 and shift the positions of the tall jobs in an optimal schedule up to the next multiple of δ^2. Again this modifications increases the height of the schedule by at most 2δ.

Lemma 3.5.1. *Let S be an optimal schedule, $\mathrm{h}(S) \leq 1$, for all jobs L. At the cost of an increase in height of at most 2δ we can round up all tall jobs to the nearest multiple of δ^2 and we can shift the jobs such that the start time of all jobs is a multiple of δ^2.*

Proof. Analogous to Lemma 3.4.3 (Section 3.4.1.3). □

3.5.2 Dynamic Program for Tall Jobs

Draw horizontal lines spaced by a distance δ^2 across the schedule starting with the x-axis as first such line. Note that, due to the rounding and shifting the lower side of each tall job corresponds to one of these horizontal lines. We say that job R_i is in slot i if its starting time (in a given schedule) corresponds to the ith horizontal line. Let

$$I_{\text{S}} := \left\{1, \ldots, \frac{(1+2\delta)}{\delta^2}\right\}$$

denote the set of slots and let

$$q := |I_{\text{S}}| = \frac{(1+2\delta)}{\delta^2} \leq \frac{2}{\delta^2}. \tag{3.28}$$

Since we cannot enumerate all possible assignments of tall jobs to slots in polynomial time, we use a dynamic programming approach similar to the approach in Section 3.4.2.3. Again, due to the height bound of 1 and the rounding, we can partition the set of tall jobs L_{ta} into a constant number of height classes $L(i)$ with

$$L(i) := \left\{R_j \in L_{\text{ta}} \;\middle|\; h_j = i \cdot \delta^2\right\} \qquad \text{for all } i \in I_{\text{ta}} := \left\{\frac{1}{\delta} + i \;\middle|\; i \in \mathbb{N} : 1 \leq i \leq \frac{1-\delta}{\delta^2}\right\}.$$

Lemma 3.5.2. *The set of all tall rectangles can be partitioned into a constant number of subsets:*

$$L_{ta} = \dot{\bigcup_{i \in I_{ta}}} L(i). \tag{3.29}$$

In particular, the number of partitions of L_{ta} is $|I_{ta}| = \frac{1-\delta}{\delta^2} \leq \frac{1}{\delta^2}$.

Proof. Analogous to Lemma 3.4.4 (Section 3.4.1.3). □

Consider an optimal schedule for a scaled, shifted and rounded instance. We can define a vector $v^i = (v^i_1, \ldots, v^i_q)$ for each height $i \in I_{\text{ta}}$ such that each entry $v^i_j \cdot m$ denotes the total width of all tall jobs with height $i \cdot \delta^2$ in slot $j \in I_{\text{S}}$. Obviously,

$$\sum_{i \in I_{\text{ta}}} \sum_{j \in I_{\text{S}}} v^i_j \frac{1}{m} = \sum_{R_i \in L_{\text{ta}}} w_i \leq 1 \tag{3.30}$$

(otherwise the schedule is not feasible) and thus we have

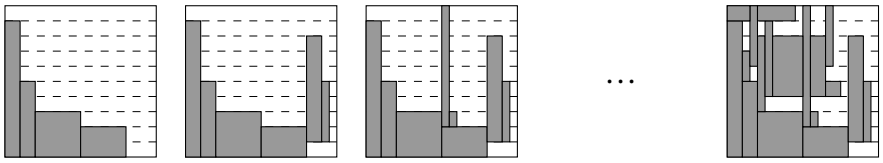
$$v^i_j \leq m \tag{3.31}$$

for each $i \in I_{\text{ta}}$, $j \in I_{\text{S}}$.

Furthermore, the value of every entry v^i_j is integral, since the width of each job is a multiple of $1/m$. Thus, a rough upper bound for the number of feasible vectors for each height $i\delta^2$ ($i \in I_{\text{ta}}$) is given by $(m+1)^q$ since there are $q = |I_{\text{S}}|$ components (one for each slot) and for every component we have $v^i_j \in \{0, \ldots, m\}$. This number is polynomial in n, since m is polynomial in n and q is a constant (see Equation (3.28)).

With a dynamic programming approach we can compute a list of all feasible vectors satisfying (3.30) and (3.31). The algorithm to calculate all feasible vectors for a given height class $i \in I_{\text{ta}}$ works as follows. Assume that $L(i) = \{R_1, \ldots, R_{k_i}\}$. Starting with a set $V^i := \{(0, \ldots, 0)\}$ containing only the null vector, in step $l \in \{1, \ldots, k_i\}$ we replace each vector $v \in V^i$ with all vectors that can be generated by adding $\gamma_l := w_l \cdot m$ to one of its components. After each step remove all duplicate vectors. Note that for height class i only the slots $\{1, \ldots, \frac{(1+2\delta)}{\delta^2} - i + 1\}$ have to be considered, since jobs belonging to $L(i)$ can be only in these slots without exceeding the height bound of $1 + 2\delta$. Since the number of different vectors is bounded by $(m+1)^q$ and $q \leq 2/\delta^2$ (see Equation 3.28), we can show that the running time of the algorithm is polynomially bounded in m with similar arguments as in Section 3.4.2.3.

After repeating this computation for each height class $L(i)$, again we can build the direct product of all sets of vectors $V := \bigtimes_{i \in I_{\text{ta}}} V^i$ and again there is an element $v \in V$ that cor-



(a) First slot (b) Second slot (c) Third slot (d) (e) All slots

Figure 3.10: Canonical packing for tall rectangles

responds to the vectors of widths induced by an optimal solution for the scaled instance. Again we use the direct product notation only for convenience. Analogous to the first case, we extend our dynamic program such that for each component of each vector a set of associated jobs is stored. Let $L(v_j^i)$ be this set of jobs associated with v_j^i.

Define $\varphi_j(i) := \max\{1, j-i+2\}$. Then, $\varphi_j(i)$ is the index of the lowest slot, such that a rectangle of height $i\delta^2$ starting in slot $\varphi_j(i)$ intersectsslot j, since

$$(\varphi_j(i)-1)\delta^2 + i\delta^2 \geq ((j-i+2)-1)\delta^2 + i\delta^2 = (i+1)\delta^2.$$

We call an element $v \in V$ *feasible*, if for each slot the total width of all tall jobs intersecting this slot (including all tall jobs in this slot) is not greater than 1. That is,

$$\sum_{i \in L_{\text{ta}}} \sum_{k=\varphi_j(i)}^{k \leq j} v_k^i \leq m \qquad \text{for all } j \in I_{\text{S}}. \tag{3.32}$$

Obviously, the vector induced by a scaled, rounded, and shifted optimal solution is feasible.

3.5.3 Canonical Packing for Tall Jobs

In the following, we present a canonical way to schedule all tall jobs (see Figure 3.10).

Lemma 3.5.3. *Let $v = (v^1, \ldots, v^{|I_{ta}|}) \in V$ be a feasible vector. There exists a canonical schedule for all tall jobs.*

Proof. Let $v = (v^1, \ldots, v^{|I_{\text{ta}}|}) \in V$ be a feasible vector. The algorithm starts with the first slot ($j = 1$) and schedules/packs left aligned all tall jobs $L(v_1^i), i \in I_{\text{ta}}$ into this slot (see Figure 3.10a). Obviously, this is possible, since the vector is feasible, i.e.

$$\sum_{i \in L_{\text{ta}}} \sum_{k=\varphi_j(i)}^{k \leq 1} v_k^i = \sum_{i \in I_{\text{ta}}} v_1^i \leq m.$$

Now assume that we have scheduled all slots prior to slot j. Since v is feasible, the free space in slot j is sufficient to (fractionally) pack all jobs assigned by $v_j^i, i \in I_{\text{ta}}$. Note that all jobs scheduled in previous slots and intersecting the current slot are accounted for in Equation (3.32). Furthermore, the free space in this slot is also free in all following slots (see Figure 3.10c). This allows us to (fractionally) pack all jobs $L(v_j^i)$ ($i \in I_{\text{ta}}$) left aligned into slot j. □

3.5.4 Packing Low Jobs

Given a feasible vector $v \in V$, the total width for each slot that is not occupied by tall jobs can be computed. Let

$$w_j^f := 1 - \left(\sum_{i \in L_{\text{ta}}} \sum_{k=\varphi_j(i)}^{k \le j} \frac{v_k^i}{m} \right) \qquad \text{for each } j \in I_{\text{S}}$$

denote the total width of the free space for slot j and define for each slot j a container C_j of width w_j^f and height δ^2.

Consider an optimal schedule of all jobs (after scaling, rounding and shifting). In this optimal schedule some low jobs might intersect the horizontal lines that form the borders of the slots. Since the height of all low jobs is bounded by γ, increasing the height of the containers to $\delta^2 + \gamma$ ensures that all low jobs are packable inside the containers. Analogous to Lemma 3.4.8, almost all low jobs can be packed into the containers using the mKR algorithm. The total area of the discarded jobs is bounded by δ. Note that the discarded jobs will be packed in a post-processing step.

The next step is to *schedule* the containers. Since we increased the height of the containers in order to ensure that all low-wide rectangles can be packed, the first step is to decrease the height again and remove all overlapping jobs. Since the height of each low job is bounded by γ, the total area of all overlapping jobs is at most

$$2\gamma \frac{1+2\delta}{\delta^2} \le 2\frac{\delta^3}{4 \cdot 7^2} \frac{1+2\delta}{\delta^2} \le \frac{\delta + 2\delta^2}{3} \le \frac{\delta + 2\delta}{3} = \delta.$$

Thus, the total area of discarded jobs is at most 2δ. Altough now the height of all containers corresponds to the height of the free space in each slot, in general it is still not possible to schedule the container without fragmentation. Therefore, we split the containers into slices of width $1/m$. This is feasible since the jobs need not be schedule on contiguous machines. Due to the definition of the containers the width of each container corresponds exactly to the free space of each slot. This allows us to add the slices successively, left-

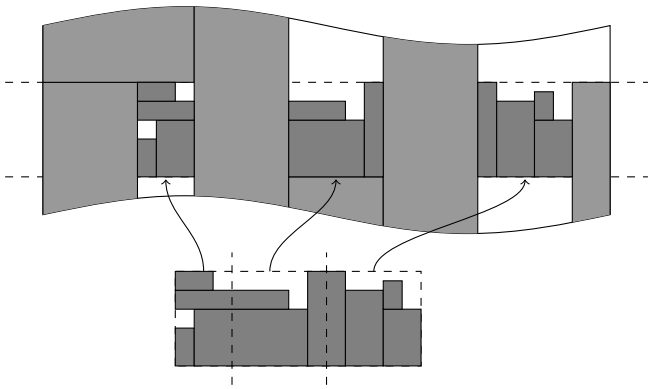


Figure 3.11: Split container to pack slots

aligned to the free space of each corresponding slot (see Figure 3.11). Note that packing the low rectangles into the containers might not be possible, if we have chosen the *wrong* vector v or scaling factor v^*.

3.5.5 Analysis

In the following, we summarize the algorithm for $P\text{poly}|\text{size}_j|C_{\max}$ (see Algorithm 2 for pseudo-code).

While creating the gap, we discard jobs with total area bounded by $\varepsilon' := A_1$ (see Section 3.5.1.2). Due to the shifting and rounding of the tall jobs we increased the height of the resulting schedule to $1+2\delta =: h_1$ (see Section 3.5.1.3). In order to ensure that all low-wide jobs fit into the containers, we increased the height of all containers by γ. This led to overlapping jobs with total area bounded by $\delta =: A_2$ (see Section 3.5.4). Furthermore, we discarded all jobs that were not packed by the modified Kenyon and Rémila algorithm, these discarded jobs have total area $\delta =: A_3$ (see Section 3.5.4). Note that this step is not always successful if we have chosen the *wrong* vector v or scaling factor v^*. In case of failure, we try another combination. Overall the resulting schedule has height

$$h_1 = 1+2\delta$$

and we discarded jobs with total area bounded by

$$A_1 + A_2 + A_3 = \varepsilon' + \delta + \delta.$$

In a post-processing step we pack all discarded jobs on top of the schedule. For this step we use the NFDH algorithm. Due to the fact that all discarded jobs have height bounded

Algorithm 2: Algorithm for $P\text{poly}|\text{size}_j|C_{\max}$

Input: Set of jobs $L = \{R_i \mid i \in \{1,\dots,n\}\}$, and precision ε
Output: A schedule S with $h(S) \leq (1+\varepsilon)\,\text{OPT}$

```
/* see Section 3.5.1.1 */
```
Let v be the height of the solution generated by 2-approximation
foreach $v^* \in \{(1+0\varepsilon)\frac{v}{2}, (1+1\varepsilon)\frac{v}{2}, \dots, (1+\lceil\frac{1}{\varepsilon}\rceil\varepsilon)\frac{v}{2}\}$ **do**
    `/* see Section 3.5.1.2 */`
    Set ε' such that $\varepsilon' = \frac{1}{a}$ for an integer a and such that $\varepsilon' \leq \frac{\varepsilon}{8}$
    Find δ
    Set $\gamma \leftarrow \frac{\delta^3}{4 \cdot 7^2}$
    Partition L into $L_{\text{ta}}, L_{\text{lo-wi}}, L_{\text{lo-na}}$
    `/* see Section 3.5.1.3 */`
    Round up h_i to the next multiple of δ^2 for all $R_i \in L_{\text{ta}}$
    Calculate V by dynamic program `/* see Section 3.5.2 */`
    foreach $v \in V$ **do**
        if v *is feasible* **then**
            Pack L_{ta} in a canonical way `/* see Section 3.5.3 */`
            Pack $L_{\text{lo-wi}} \cup L_{\text{lo-na}}$ (if possible) `/* see Section 3.5.4 */`
            Save solution (if it exists)
Choose schedule with minimal length

by δ, this results in an additional height of at most

$$2(\varepsilon' + 2\delta) + \delta = 2\varepsilon' + 5\delta.$$

Thus, the height of the resulting schedule is bounded by

$$(1 + 2\delta) + (2\varepsilon' + 5\delta) \overset{\delta \leq \varepsilon'}{\leq} 1 + 9\varepsilon' \overset{\varepsilon' \leq \frac{\varepsilon}{9}}{\leq} 1 + \varepsilon.$$

Since we scaled the instance in Section 3.5.1.1, the last step is to multiply the length of the schedule by v^*:

$$\begin{aligned} v^*(1+\varepsilon) &\leq (1+\varepsilon)\,\mathrm{OPT}(1+\varepsilon) = (1 + 2\varepsilon + \varepsilon^2)\,\mathrm{OPT} \\ &\leq (1+3\varepsilon)\,\mathrm{OPT}. \end{aligned}$$

This proves Theorem 3.1.2. The running time of the algorithm is in $\mathcal{O}(n^{f(\frac{1}{\varepsilon})})$ for some exponential function f.

3.6 Malleable Parallel Job Scheduling

In the following, we extend both algorithms for scheduling malleable jobs. We denote the malleable versions of Ppoly|line$_j$|$C_{\max}$ and Ppoly|size$_j$|$C_{\max}$ by Ppoly|fnct_line$_j$|$C_{\max}$ and Ppoly|fnct$_j$|$C_{\max}$, respectively. Instead of a fixed pair consisting of the number of required processors and the execution time, in this setting each job J_j is associated with a function $p_j : \{1, \ldots, m\} \to \mathbb{Q}^+$ that gives the execution time $p_j(\ell)$ of J_j in terms of the number ℓ of processors assigned to J_j.

We present a dynamic program that generates a polynomial number of assignments of jobs to the number of processors they use. If we have chosen an assignment, we use the corresponding non-malleable algorithm to find a nearly optimal solution. Iterating over all assignments generated by the dynamic program allows us to find a nearly optimal solution. In the following, we assume that $\varepsilon \leq 1/2$, where ε is the required accuracy.

3.6.1 Simple Structure

Before we present the dynamic program, we show again that an optimal solution can be transformed into a nearly optimal solution with simpler structure.

3.6.1.1 Bounded Height

Analogous to the non-malleable cases, we can find a schedule with length $\nu \leq 2 \cdot \mathrm{OPT}$ by using the 2-approximation algorithm by Ludwig & Tiwari [54] (for both cases contiguous and non-contiguous), where OPT is the length of an optimal solution.

Let $C := \{(1+0\varepsilon)\nu/2, (1+1\varepsilon)\nu/2, \ldots, (1+\lceil 1/\varepsilon \rceil \varepsilon)\nu/2\}$. Then there exists a value $\nu^* \in C$ such that $\mathrm{OPT} \leq \nu^* \leq (1+\varepsilon)\,\mathrm{OPT}$. Obviously, we only have to consider $\lceil 1/\varepsilon \rceil + 1$ different candidates to find this value.

Again, we scale the execution times of all jobs by ν^*. But since we do not know the number of processors assigned to each job, the scaling of the execution time of each job is done by components, that is, for each job J_j and for each number of processors l we scale the value of $p_j(\ell)$.

3.6.1.2 Partitioning

In order to simplify a given schedule, we are going to partition the set of jobs into tall jobs, middle-sized jobs and small jobs as before. But since we have no knowledge about the *size* of the jobs in the optimal solution, we cannot calculate δ, γ in advance. We have to enumerate all possible values for δ, γ. This is feasible since there is only a constant number of candidates (see Sections 3.4.1.2 and 3.5.1.2).

- In the contiguous case we have $2/\varepsilon'$ candidates σ_k with $\sigma_0 := 1, \sigma_1 := \varepsilon'$, and $\sigma_k := (\sigma_{k-1})^{8/\sigma_{k-1}^3}$ for all $k \geq 2$; we enumerate all values $k \in \{2, \ldots, 2/\varepsilon' + 1\}$ and set $\delta := \sigma_{k-1}$ and $\gamma := \sigma_k = \delta^{8/\delta^3}$.

- In the non-contiguous case we have $2/\varepsilon'$ candidates σ_k with $\sigma_0 := 1, \sigma_1 := \varepsilon'$, and $\sigma_k := \sigma_{k-1}^3/(4 \cdot 7^2)$ for all $k \geq 2$; we enumerate all values $k \in \{2, \ldots, 2/\varepsilon' + 1\}$ and set $\delta := \sigma_{k-1}$ and $\gamma := \sigma_k = \delta^3/(4 \cdot 7^2)$.

Even knowing δ and γ we cannot partition the set of rectangles at this point, since we do not know how many processors will be assigned to each job. However, if we have fixed the number of processors ℓ assigned to a job R_j, we will call R_j

$$\begin{aligned} \textbf{tall} &\quad \text{if } p_j(\ell) > \delta \\ \textbf{low} &\quad \text{if } p_j(\ell) \leq \gamma, \\ \textbf{wide} &\quad \text{if } \ell > \delta m, \text{ and} \\ \textbf{narrow} &\quad \text{if } \ell \leq \gamma m. \end{aligned}$$

The following lemma shows that among these $2/\varepsilon'$ candidates there is a least one allowing the gap creation as before, i.e. the set of discarded jobs has small total area.

Lemma 3.6.1. *Let S be an arbitrary schedule for L. Then there exists a pair δ, γ among all candidates such that the area*

$$A(L_k) \leq \varepsilon' A(L)$$

with $L_k := \{R_i \in L \mid \gamma < r_i < \delta \quad or \quad \gamma < p_i(r_i) < \delta\}$.

Proof. Analogous to the proof of Lemma 3.4.1. □

3.6.1.3 Rounding and Shifting

Let S be an optimal schedule for a given instance. In particular in this schedule for each job, the number of assigned processors is fixed and thus, the processing time is known.

Lemma 3.6.2. *There exists a schedule S with nearly optimal length and simpler structure. That is:*

(a) $h(S) \leq (1+3\delta)\,OPT$, *where* $h(S)$ *denotes the length of schedule* S.

(b) For each tall job $R_i \in L_{ta}$ the start time is a multiple of δ^2 and the processing time / height of each tall job can be rounded up to the next multiple of δ^2.

(c) For all other rectangles, $R_i \in L \setminus L_{ta}$, the processing time can be rounded up to the next multiple of γ/n.

Proof. Let S^* be an optimal schedule. We can modify this schedule basically in the same way as in the non-malleable cases, since in schedule S^* the number of processors assigned to each job is fixed and thus the processing time is also fixed.

Rounding up the processing time of all non-tall jobs R_i (i.e. jobs with processing time $\leq \delta$) to the next multiple of γ/n increases the height by at most

$$n \cdot \frac{\gamma}{n} = \gamma.$$

Analogous to the proof of Lemma 3.4.3, we can shift the starting time and round the processing time of each tall job after scaling the schedule by $(1+2\delta)$. Since $\gamma \leq \varepsilon \leq 1/2$, this leads to an increased height of at most

$$(1+2\delta)(1+\gamma) = 1+\gamma+2\delta+2\delta\gamma \leq 1+3\delta.$$

□

Thus, we can round up the running time of all jobs at the cost of at most 3δ. Again, since we do not know the number of processors assigned to each job, the rounding of the execution time of each job is done by components, that is, for each job R_i and for each number of processors ℓ we round up the value of $p_i(\ell)$; the value is rounded up to the next multiple of δ^2 if $p_i(\ell) > \delta$ and to the next multiple of γ/n otherwise. In the following, we denote the scaled and rounded execution times by $\tilde{p}_j(\ell)$. Note that $\tilde{p}_j(\ell) \leq \delta$ iff $p_j(\ell) \leq \delta$, since δ is a multiple of γ and thus δ is a multiple of γ/n. Furthermore, $\tilde{p}_j(\ell) \leq \gamma$ iff $p_j(\ell) \leq \gamma$, since γ is a multiple of γ/n.

3.6.1.4 Container

In the following, we show that there exists a constant number of widths such that the width of all low-wide rectangles can be rounded up to one of these widths.

Lemma 3.6.3. *Let S be a nearly optimal schedule (scaled, rounded, shifted). There exists a packing $\hat{S}$ and a vector of width $(b_1, \ldots, b_{\hat{m}})$ such that*

(a) $\mathrm{h}(\hat{S}) \leq \mathrm{h}(S) + \delta$

(b) the total area of discarded rectangles is bounded by 2δ, and

(c) the width of each low-wide rectangle can be rounded up to a width b_i for some $i \in \{1, \ldots, \hat{m}\}$.

Proof. Consider a nearly optimal schedule (scaled, rounded, shifted) S. We define slots as in Section 3.4.1.4 or 3.5.4. Draw horizontal lines spaced by a distance δ^2 across the schedule. Due to the rounding and shifting, the lower and upper sides of the tall jobs lie along two of these lines. These lines split the schedule into at most $(1+3\delta)/\delta^2$ horizontal rectangular regions that we call *slots*. The definition of containers depends on the scheduling problem we are considering.

In the contiguous case, we define a container as a rectangular region inside a slot whose left boundary is either the right side of a tall rectangle or the left side of the strip, and whose right boundary is either the left side of a tall rectangle or the right side of the schedule. We consider only containers that contain at least one low-wide rectangle (see Section 3.4.1.4).

In the non-contiguous case, we define a container a little bit differently. Let w_j^f denote the total width for each slot j that is not occupied by tall rectangles. We define for each slot j a container C_j of width w_j^f and height δ^2 (see Section 3.5.4).

As we have shown in Section 3.4.4.6 (contiguous case) and in Section 3.5.4 (non-contiguous case) we can pack almost all low-wide rectangles using the mKR algorithm. The

mKR algorithm stacks all low-wide rectangles on top of each other and divides the stack into $\hat{m}$ groups (see Section 3.3.1, Figure 3.1). The width of each wide rectangle is rounded to the width of the widest rectangle of the corresponding group. During repacking, rectangles with total area at most δ are discarded. Furthermore, we have to discard overlapping rectangles with total area bounded by δ (see Sections 3.4.4.6 and 3.5.4).

Thus,

- the total area of all discarded rectangles is bounded by 2δ, and
- there is only a constant number of different widths among the low-wide rectangles.

□

3.6.1.5 Induced Vector

Assume that we have a scaled, rounded, and shifted solution according to Sections 3.6.1.1, 3.6.1.2, 3.6.1.3, 3.6.1.4. Let $g := 1/\delta^2$, $q := (1+3\delta)/\delta^2$ and let $\hat{m}$ denote the number of groups constructed in the mKR algorithm (see Section 3.3.1). For this schedule with simpler structure we can define a vector

$$v = (v_1^t, \ldots, v_g^t, v_1^w, \ldots, v_{\hat{m}}^w, v^s, v^d)$$

with the following semantics:

- $v_i^t = (v_{i,1}^t, \ldots, v_{i,q}^t)$ is a vector and $v_{i,j}^t \cdot \frac{1}{m}$ denotes the total width of all tall jobs with height $i\delta^2$ in slot j for each $i \in \{1,\ldots,g\}, j \in \{1,\ldots,q\}$; e.g. $v_{\frac{1}{\delta}+1,2}^t = 5$ means that the sum of the widths of all jobs with height $(\frac{1}{\delta}+1)\cdot\delta^2 = \delta+\delta^2$ in slot 2 is $\frac{5}{m}$. Note that $v_{i,j}^t = 0$ for all $i \leq \frac{1}{\delta}$, since all tall jobs have height $> \delta = \frac{1}{\delta}\cdot\delta^2$. Furthermore, $v_{i,j}^t = 0$ for all $j > \frac{(1+3\delta)}{\delta^2} - i + 1$, since jobs of height $i\delta^2$ would exceed the height $(1+3\delta)$ if placed in such slots j.
- $v_j^w \cdot \frac{\gamma}{n}$ denotes the total height of all wide jobs belonging to group j as constructed by the mKR algorithm for each $j \in \{1,\ldots,\hat{m}\}$; e.g. $v_2^w = 5$ means that the total height of all wide jobs R_i with width $b_2 \leq w_i < b_3$ (or $b_2 = w_i$ if $b_2 = b_3$) is $5 \cdot \frac{\gamma}{n}$.
- $v^s \cdot \frac{\gamma}{nm}$ denotes total area of all small jobs; e.g. $v^s = 5$ means that the total area of all small jobs is $5 \cdot \frac{\gamma}{nm}$.
- $v^d \cdot \frac{\gamma}{nm}$ denotes total area of all discarded jobs; e.g. $v^d = 5$ means that the total area of all discarded jobs is $5 \cdot \frac{\gamma}{nm}$.

Due to the rounding and normalization, all components must have discrete values and the value of each component is bounded. We have:

- $v^t_{i,j} \in \{0,\dots,\frac{(1+3\delta)}{\delta}m\}$ for each $i \in \{1,\dots,g\}$, $j \in \{1,\dots,q\}$; the value is integral, since the width of each job is a multiple of $\frac{1}{m}$; the value is bounded by $\frac{(1+3\delta)}{\delta}m$, since otherwise the total area of all jobs with height $i\delta^2$ is

$$\underbrace{i\delta^2}_{>\delta} \cdot \underbrace{v^t_{i,j}}_{>\frac{(1+3\delta)}{\delta}m} \cdot \frac{1}{m} > \delta\frac{(1+3\delta)}{\delta}m\frac{1}{m} = 1+3\delta.$$

- $v^w_i \in \{0,\dots,\frac{(1+3\delta)}{\delta}\frac{n}{\gamma}\}$ for each $i \in \{1,\dots,\hat{m}\}$; the value is integral, since the height of all non-tall jobs is a multiple of $\frac{\gamma}{n}$; the value is bounded by $\frac{(1+3\delta)}{\delta}\frac{n}{\gamma}$, since otherwise the total area of all jobs in group i is at least

$$\underbrace{b_{i+1}}_{>\delta}\frac{(1+3\delta)}{\delta}\frac{n}{\gamma}\frac{\gamma}{n} > 1+3\delta.$$

- $v^s \in \{0,\dots,(1+3\delta)\frac{(mn)}{\gamma}\}$; the value is integral, since the height of all non-tall jobs is a multiple of $\frac{\gamma}{n}$ and the width of all jobs is a multiple of $\frac{1}{m}$; the value is bounded by $(1+3\delta)\frac{(mn)}{\gamma}$, since otherwise the total area of all small jobs exceeds

$$(1+3\delta)\frac{(mn)}{\gamma}\cdot\frac{\gamma}{nm} = 1+3\delta.$$

- $v^d \in \{0,\dots,3\varepsilon'\frac{nm}{\gamma}\}$; the value is integral, since the height of all discarded (non-tall) jobs is a multiple of $\frac{\gamma}{n}$ and the width of all jobs is a multiple of $\frac{1}{m}$; the value is bounded by $3\varepsilon'\frac{nm}{\gamma}$, since otherwise the total area of all discarded jobs exceeds

$$3\varepsilon'\frac{nm}{\gamma}\cdot\frac{\gamma}{nm} = 3\varepsilon'.$$

  However, the total area of all discarded jobs is at most $3\varepsilon'$.

Thus in total, the number of different vectors is bounded by

$$\left(\frac{(1+3\delta)}{\delta}m+1\right)^{gq}\left(\frac{(1+3\delta)}{\delta}\frac{n}{\gamma}+1\right)^{\hat{m}}\left((1+3\delta)\frac{nm}{\gamma}+1\right)\left(\frac{3\varepsilon'\cdot nm}{\gamma}+1\right) \in \mathcal{O}(m^{gq+2}n^{\hat{m}+2}) \tag{3.33}$$

and

$$gq = \frac{1}{\delta^2}\frac{(1+3\delta)}{\delta^2} = \frac{1+3\delta}{\delta^4},$$
$$\hat{m} \in \mathcal{O}(\frac{1}{\delta^2}).$$

3.6.2 Dynamic Program

In the following, we assume that we have chosen a vector $b := (b_1, \ldots, b_{\hat{m}})$ such that b_i denotes the width of group i (as introduced by the mKR algorithm). Let $b_{\hat{m}+1} := \delta$.

The dynamic program works as follows. We start with a set $V^0 := \{(0,\ldots,0)\}$ containing only the null vector. Then we iterate over the set of jobs $L = \{J_1, \ldots, J_n\}$ and generate in each step i a new set V^i of vectors by replacing each vector from V^{i-1} with all vectors that can be generated by *adding* J_i to each component, if feasible. To be more specific let $v = (v_1^t, \ldots, v_g^t, v_1^w, \ldots, v_{\hat{m}}^w, v^s, v^d) \in V^{i-1}$. To generate new vectors we try to *add* J_i to each component in turn. This *adding* is done as follows.

v_k^t For each $j \in \{1,\ldots,q\}$ and for each $k \in \{1,\ldots,g\}$ with $k\delta^2 > \delta$ let ℓ be the minimal number of processors such that $\tilde{p}_i(\ell) = k\delta^2$.

If ℓ exists, define $\hat{v}_k^t := (v_{k,1}^t, \ldots, v_{k,j-1}^t, v_{k,j}^t + \ell, v_{k,j+1}^t, v_{k,q}^t)$ and $v' := (v_1^t, \ldots, v_{k-1}^t, \hat{v}_k^t, v_{k+1}^t, \ldots, v_g^t, v_1^w, \ldots, v_{\hat{m}}^w, v^s, v^d)$ and add v' to V^i. If ℓ is not existing, we continue with the next component.

v_k^w For each $k \in \{1,\ldots,\hat{m}\}$ with $b_k \neq b_{k+1}$ we choose $\ell \in (m \cdot b_{k+1}, m \cdot b_k]$ such that $\tilde{p}_i(\ell)$ is minimal; if $b_k = b_{k+1}$ we choose $\ell := b_k m$.

If $\tilde{p}_i(\ell) \leq \gamma$ (i.e. J_i is a low job, furthermore J_i is a wide job, since $b_k \geq b_{\hat{m}+1} = \delta$), define $v' := (v_1^t, \ldots, v_g^t, v_1^w, \ldots, v_{k-1}^w, v_k^w + \frac{\tilde{p}_i(\ell)n}{\gamma}, v_{k+1}^w, \ldots, v_{\hat{m}}^w, v^s, v^d)$ and add v' to V^i. If $\tilde{p}_i(\ell) > \gamma$, we continue with the next component.

v^s Choose $\ell \in \{1,\ldots,\gamma m\}$ such that $\tilde{p}_i(\ell) \leq \gamma$ and such the area of J_i, $A_\ell(J_i) = {}^{\ell}/_{m} \cdot \tilde{p}_i(\ell)$, is minimal.

If such ℓ exists, define $v' := (v_1^t, \ldots, v_g^t, v_1^w, \ldots, v_{\hat{m}}^w, v^s + A_\ell(J_i) \cdot \frac{nm}{\gamma}, v^d)$ and add v' to V^i.

v^d Choose $\ell \in \{(\gamma m)+1,\ldots,\delta m\}$ such that $\gamma < \tilde{p}_i(\ell) \leq \delta$ and such the area of J_i, $A_\ell(J_i) = {}^{\ell}/_{m} \cdot \tilde{p}_i(\ell)$, is minimal.

If such ℓ exists, define $v' := (v_1^t, \ldots, v_g^t, v_1^w, \ldots, v_{\hat{m}}^w, v^s, v^d + A_\ell(J_i) \cdot \frac{nm}{\gamma})$ and add v' to V^i.

If during an *adding step* a vector is generated that contains a component with value exceeding the above mentioned bounds, this vector is discarded. Since V^i is a set, no duplicate vectors (vectors that have the same components) occur. Consequently, the number of all vectors in V^i is polynomially bounded,

$$|V^i| \in \mathcal{O}(m^{gq+2} n^{\hat{m}+2}), \qquad \text{see Equation (3.33).}$$

Again, we extend the dynamic program such that for each component of each vector a set of associated jobs with fixed number of processors is stored. This increases the space needed to store the vectors, but it is still polynomial in n. Due to this extension, we can create a list of *non-malleable* jobs based on a vector $v \in V$. We denote this distinct list by $L(v)$.

In contrast to the previous dynamic programs, V might not contain a vector corresponding to the vector induced by a nearly optimal solution. However, we show in the following lemma that there is a vector $v \in V$ that is nearly optimal.

Lemma 3.6.4. *Let S be an optimal schedule for the scaled instance (i.e. $\mathrm{h}(S) \leq 1$). Then there exists $v \in V$ such that*

$$\mathrm{OPT}(L(v)) \leq (1+\tilde{\varepsilon}),$$

where $\mathrm{OPT}(L(v))$ denotes an optimal solution for $L(v)$ and $\tilde{\varepsilon}$ is some constant depending on ε'.

Proof. Let S be an optimal schedule and let S' be the nearly optimal schedule with simpler structure (i.e. rounded and repacked) and let L^d be the set of discarded jobs. Then $\mathrm{h}(S') \leq (1+3\delta)\,\mathrm{h}(S)$ and $\mathrm{A}(L^d) \leq 2\delta + \varepsilon'$ (see Sections 3.6.1.3, 3.6.1.4). Note that in S' the number of processors for each job is fixed and for each tall job the slot it is scheduled in is given.

Create v' as follows. We add each job basically in the same way as in the dynamic program. In distinction to the dynamic program, the component a job is added to is determined by S'. The resulting vector v' might differ from the induced vector, since in each adding step the number of assigned processors might be different to the number assigned by S'. The following discrepancies might occur. For each case we argue why the upper bound for an optimal solution for $L(v')$ still holds.

tall jobs The number of assigned processors might be smaller in v' than in S'. Obviously, this does not affect the bound.

low-wide jobs The number of processors assigned is chosen in the same interval, but the height is minimized. If we repack the container with the mKR algorithm, we round

up the widths of these jobs to the upper bound of the interval. Consequently, only the height of the job is affecting the solution, but the height is equal or even lower.

small/discarded jobs The number of processors might be different, but the area of the job is minimized. Since only area arguments are used for small or discarded jobs, the bound is not affected.

The dynamic program includes the same adding steps as we used for generating v'. Thus, V contains a vector v that is equivalent to v', that is, the values of all components are equal.

For the problem $P\text{poly}|\text{fnct}_j|C_{\max}$ this is sufficient, since it is possible to schedule the tall jobs fractionally and the other jobs are scheduled using only area arguments (small or discarded jobs) or height arguments (low-wide jobs). Thus, in this case

$$S(v) \leq (1+3\delta)\,\mathrm{h}(S)$$

and

$$L^d(v) \leq 2\delta + \varepsilon',$$

where $S(v)$ is the schedule with simpler structure for $L(v)$ and $L^d(v)$ is the set of discarded jobs. We add the discarded jobs using the NFDH algorithm. This leads to an additional strip of height bounded by

$$2(2\delta+\varepsilon') + \delta,$$

since the height of all discarded jobs is bounded by δ. Thus in total, we have

$$\begin{aligned}\mathrm{OPT}(L(v)) &\leq (1+3\delta)\,\mathrm{h}(S) + 5\delta + 2\varepsilon' \\ &\leq 1 + 2\varepsilon' + 8\delta \\ &\leq 1 + \tilde{\varepsilon}\end{aligned}$$

for $\tilde{\varepsilon} := 2\varepsilon' + 8\delta$.

Unfortunately, for $P\text{poly}|\text{fnct\_line}_j|C_{\max}$ this is not sufficient. Since in v might be jobs which are wider than the jobs in an optimal solution, there might not be a feasible solution for v although it is equivalent to a nearly optimal solution.

Therefore, we guess a subset $L_{(K)}$ with K jobs (where K is the same constant as in Section 3.4.4.4). For each of these K jobs, we try all numbers of processors such that this job is tall. In the following, we consider only vectors $v \in V$ where all other tall jobs have area smaller than the jobs in $L_{(K)}$. This ensures that the jobs $L_{(K)}$ are chosen in the corresponding non-malleable algorithm for pre-positioning (see Section 3.4.2). Since we try all pos-

sibilities, we find the set corresponding to the nearly optimal solution eventually. For the remaining jobs we have the same arguments as for the Ppoly|fnct$_j$|$C_{\max}$ case. The (remaining) tall jobs are packed fractionally. The upper bound for the low-wide jobs depends only on the height of the jobs and the bound for the small and the discarded jobs depends only on area arguments. If we annotate the jobs from $L_{(K)}$ in v with the guessed number of processors, we have the same result as for the previous case, i.e.

$$\mathrm{OPT}(L(v)) \leq 1 + \tilde{\varepsilon}. \qquad \square$$

3.6.3 The Algorithm

Using Lemma 3.6.4 we get the following theorem.

Theorem 3.6.5.

(a) For every $\varepsilon > 0$ there exists an algorithm A for every instance I of Ppoly|fnct_line$_j$|$C_{\max}$ such that

$$A(I) \leq (1.5 + \varepsilon)\,\mathrm{OPT}(I)$$

holds and the running time is polynomial in n, where $A(I)$ is the length of the schedule for instance I generated by algorithm A and $\mathrm{OPT}(I)$ is the length of an optimal schedule for instance I.

(b) For every $\varepsilon > 0$ there exists an algorithm A for every instance I of Ppoly|fnct$_j$|$C_{\max}$ such that

$$A(I) \leq (1 + \varepsilon)\,\mathrm{OPT}(I)$$

holds and the running time is polynomial in n, where $A(I)$ is the length of the schedule for instance I generated by algorithm A and $\mathrm{OPT}(I)$ is the length of an optimal schedule for instance I.

Proof. Since we use the algorithms for Ppoly|size$_j$|$C_{\max}$ (Algorithm 2) and Ppoly|line$_j$|$C_{\max}$ (Algorithm 1), respectively, it remains to show that there exists an algorithm to find a proper assignment of a number of required processors to each job.

Lemma 3.6.4 shows that among all vectors generated by the dynamic program is at least one which allows a nearly optimal solution. Thus, after enumerating all possibilities (for pseudo-code see Algorithm 3), the algorithm for Ppoly|line$_j$|$C_{\max}$ or Ppoly|size$_j$|$C_{\max}$ finds a nearly optimal solution. $\square$

Algorithm 3: Algorithm for the malleable case

Input: Set of jobs $L = \{R_i \mid i \in \{1,\dots,n\}\}$, and precision ε

Output: A schedule S with

$$\mathrm{h}(S) \leq \begin{cases} (1.5+\varepsilon)\,\mathrm{OPT} & \text{contiguous case, } P\mathrm{poly}|\mathrm{fnct\_line}_j|C_{\max} \\ (1+\varepsilon)\,\mathrm{OPT} & \text{non-contiguous case, } P\mathrm{poly}|\mathrm{fnct}_j|C_{\max} \end{cases}$$

```
/* see Section 3.6.1.1 */
```

Let v be the height of the solution generated by 2-approximation

foreach $v^* \in \{(1+0\varepsilon)\frac{v}{2}, (1+1\varepsilon)\frac{v}{2}, \dots, (1+\lceil\frac{1}{\varepsilon}\rceil\varepsilon)\frac{v}{2}\}$ **do**

```
/* see Section 3.6.1.2 */
```

foreach $k \in \{2,\dots,\frac{2}{\varepsilon'}\}$ **do**

Set $\delta \leftarrow \sigma_{k-1}$

Set $\gamma \leftarrow \sigma_k$

```
/* see Section 3.6.1.3 */
```

Define $\tilde{p}_i(\ell)$ for each $R_i \in L$

foreach *Vector of widths* $(b_1,\dots,b_{\hat{m}})$ **do**

Calculate V by dynamic program `/* see Section 3.6.2 */`

foreach $L_{(K)}$ `/* Problem` $P\mathrm{poly}|\mathrm{fnct\_line}_j|C_{\max}$ `only */` **do**

foreach $v \in V$ **do**

Use Algorithm 1 or 2 to find a solution

Save solution

Choose schedule with minimal length

3.7 Conclusion

In this chapter, we have shown that the problem of scheduling parallel jobs can be solved within $(1+\varepsilon')$ of the optimum, if we restrict the instances such that the number of machines is polynomially bounded in the number of jobs, $P\text{poly}|\text{size}_j|C_{\max}$. Furthermore, we presented an extension to the problem of scheduling malleable jobs, $P\text{poly}|\text{fnct}_j|C_{\max}$. These are in a sense the best results possible, since the problems are NP-hard in the strong sense. However, the running times of the presented algorithms are far from practical, so the obvious question is if the running times can be improved or whether there exists an efficient PTAS (EPTAS), i.e. an algorithm with running time $\mathcal{O}(f(\varepsilon^{-1})n^c)$ for a constant c.

For $P\text{poly}|\text{line}_j|C_{\max}$ and $P\text{poly}|\text{fnct\_line}_j|C_{\max}$ we presented $(1.5+\varepsilon)$ approximation algorithms. The existence of a PTAS is still open for these problems. Thus, it is an interesting question if a lower bound for the approximation ratio can be shown or if algorithms with better approximation ratio exist. Of course, an improvement of the running time would also be interesting.

We assume that the described approach can also be applied to other scheduling problems like resource constrained scheduling.

4 Concluding Remarks

In this thesis, we presented approximation algorithms for packing and scheduling problems.

In Chapter 2, we presented approximation algorithms for the three-dimensional orthogonal knapsack problem (OKP-3). We presented a $(9+\varepsilon)$-approximation algorithm (Theorem 2.2.4) based on a two-dimensional strip packing algorithm in Section 2.2. Furthermore, we have shown that our algorithm can also be used to solve the three-dimensional strip packing problem with absolute approximation ratio 6 (Corollary 2.2.3); this improves the best known algorithm with absolute approximation ratio $45/4 = 11.25$ which follows from [51]. In Section 2.3, we used a stronger relaxation and a refinement of the strip packing algorithm to get a $(8+\varepsilon)$-approximation (Theorem 2.3.4). We generalized the packability criterion (Theorem 2.2.1) for the three-dimensional case in Section 2.4 (Theorem 2.4.1). In order to get a better approximation ratio for *small* boxes, we revised the strip packing approach for *small* boxes in Section 2.5. A $(7+\varepsilon)$-approximation algorithm for OKP-3 was presented in Section 2.6. However, the better approximation ratio is at the cost of a considerably larger running time, which is caused by a large enumeration. In Section 2.7, we analyzed the case when rotations are allowed. We have shown that the $(7+\varepsilon)$-approximation can be modified to yield a $(6+\varepsilon)$-approximation ratio, if 90° rotations around the z-axis are allowed (Theorem 2.7.1), and a $(5+\varepsilon)$-approximation ratio if arbitrary 90° rotations are allowed (Theorem 2.7.2).

It is of interest whether an algorithm with ratio $(6+\varepsilon)$ or less exists for the non-rotational case and whether there are algorithms with better approximation ratio for the rotational cases. Furthermore, we are interested in a reduction of the running time, especially for the $(7+\varepsilon)$-approximation algorithm. In [50] it was proved that it is NP-complete to decide whether a set of squares can be packed into the unit square. However, it is an open problem whether checking the packability of cubes into the unit cube is NP-complete.

In Chapter 3, we presented approximation algorithms for variants of the non-preemptive parallel job scheduling problem in which the number of available machines is polynomially bounded in the number of jobs. After a short introduction in Section 3.1 and an outline of the algorithms in Section 3.2, we presented an algorithm for packing rectangles into a

constant number of bins in Section 3.3. For the case that jobs are allotted to contiguous machines, Ppoly|line$_j$|$C_{\max}$, we presented an $(1.5+\varepsilon)$-approximation algorithm in Section 3.4. In Section 3.5, we presented a PTAS for the non-contiguous case, Ppoly|size$_j$|$C_{\max}$. We generalized both scheduling algorithms to the malleable cases in Section 3.6; we presented a pre-processing step that allows us to assign a number of machines to each job. Subsequent application of the algorithms for the corresponding non-malleable cases yields an $(1.5+\varepsilon)$-approximation algorithm for the contiguous (Ppoly|fnct_line$_j$|$C_{\max}$) and a PTAS for the non-contiguous case (Ppoly|fnct$_j$|$C_{\max}$).

The results for the non-contiguous cases are in a sense the best possible results, since the problems are NP-hard in the strong sense. However, the running times of the presented algorithms are far from practical, so the obvious question is if the running times can be improved or whether there exists an efficient PTAS (EPTAS), i.e. an algorithm with running time $\mathcal{O}(f(\varepsilon^{-1})n^c)$ for a constant c.

For the contiguous cases, Ppoly|line$_j$|$C_{\max}$ and Ppoly|fnct_line$_j$|$C_{\max}$, the existence of a PTAS is still open. Thus, it is an interesting question if a lower bound for the approximation ratio can be shown, or if algorithms with better approximation ratios exist. Of course, an improvement of the running time would also be interesting.

We assume that the described approach can also be applied to other scheduling problems like resource constrained scheduling.

Bibliography

[1] Abdel Krim Amoura, Evripidis Bampis, Claire Kenyon, and Yannis Manoussakis. Scheduling independent multiprocessor tasks. *Algorithmica*, 32(2):247–261, 2007.

[2] Brenda S. Baker. A new proof for the first-fit decreasing bin-packing algorithm. *Journal of Algorithms*, 6(1):49–70, 1985.

[3] Brenda S. Baker, Donna J. Brown, and Howard P. Katseff. A 5/4 algorithm for two-dimensional packing. *Journal of Algorithms*, 2(4):348–368, 1981.

[4] Nikhil Bansal, Alberto Caprara, and Maxim Sviridenko. Improved approximation algorithms for multidimensional bin packing problems. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006)*, pages 697–708. IEEE Computer Society, 2006.

[5] Nikhil Bansal, José R. Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: Inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31(1):31–49, 2006.

[6] Nikhil Bansal, Xin Han, Kazuo Iwama, Maxim Sviridenko, and Guochuan 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.

[7] Krishna P. Belkhale and Prithviraj Banerjee. A scheduling algorithm for parallelizable dependent tasks. In *Proceedings of the 5th International Parallel Processing Symposium (IPPS 1991)*, pages 500–506, 1991.

[8] Stefan Bischof and Ernst W. Mayr. On-line scheduling of parallel jobs with runtime restrictions. *Theoretical Computer Science*, 268(1):67–90, 2001.

[9] Jacek Błażewicz, Klaus H. Ecker, Erwin Pesch, Günter Schmidt, and Jan Węglarz. *Handbook on Scheduling: From Theory to Applications (International Handbooks on Information Systems)*. Springer, 2007.

[10] Alberto Caprara. Packing 2-dimensional bins in harmony. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 490–499. IEEE Computer Society, 2002.

[11] Miroslav Chlebík and Janka Chlebíková. Inapproximability results for orthogonal rectangle packing problems with rotations. In Tiziana Calamoneri, Irene Finocchi, and

Giuseppe F. Italiano, editors, *Proceedings of the 6th Italian Conference on Algorithms and Computation (CIAC 2006)*, volume 3998 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 2006.

[12] Edward G. Coffman, Jr., Michael R. Garey, David S. Johnson, and Robert E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

[13] Thomas Decker, Thomas Lücking, and Burkhard Monien. A 5/4-approximation algorithm for scheduling identical malleable tasks. *Theoretical Computer Science*, 361(2):226–240, 2006.

[14] Florian Diedrich. Approximative Algorithmen für Rucksackprobleme. diploma thesis, Institut für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel, 2004.

[15] Florian Diedrich, Rolf Harren, Klaus Jansen, Ralf Thöle, and Henning Thomas. Approximation algorithms for 3d orthogonal knapsack. In *Theory and Applications of Models of Computation*, volume 4484 of *Lecture Notes in Computer Science*, pages 34–45. Springer, 2007.

[16] Florian Diedrich, Rolf Harren, Klaus Jansen, Ralf Thöle, and Henning Thomas. Approximation algorithms for 3d orthogonal knapsack. *Journal of Computer Science and Technology*, 23(5):749–762, 2008.

[17] György Dósa. The tight bound of first fit decreasing bin-packing algorithm is FFD(I) <= 11/9OPT(I) + 6/9. In Bo Chen, Mike Paterson, and Guochuan Zhang, editors, *ESCAPE*, volume 4614 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2007.

[18] Maciej Drozdowski. Scheduling multiprocessor tasks – an overview. *European Journal of Operational Research*, 94(2):215–230, 1996.

[19] Jianzhong Du and Joseph Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, 1989.

[20] Leah Epstein. Two dimensional packing: The power of rotation. In Branislav Rovan and Peter Vojtás, editors, *Symposium on Mathematical Foundations of Computer Science (MFCS 2003)*, volume 2747 of *Lecture Notes in Computer Science*, pages 398–407. Springer, 2003.

[21] Leah Epstein and Rob van Stee. This side up! *ACM Transactions on Algorithms*, 2(2):228–243, 2006.

[22] Anja Feldmann, Ming-Yang Kao, Jiří Sgall, and Shang-Hua Teng. Optimal on-line scheduling of parallel jobs with dependencies. *Journal of Combinatorial Optimization*, 1(4):393–411, 1998.

[23] Anja Feldmann, Jiří Sgall, and Shang-Hua Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science (Special Issue on Dynamic and On-line Algorithms)*, 130(1):49–72, 1994.

[24] Wenceslas Fernandez de la Vega and George S. Lueker. Bin packing can be solved within $1+\epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[25] Michael R. Garey and Ronald L. Graham. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[26] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[27] Rolf Harren. Approximating the orthogonal knapsack problem for hypercubes. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Annual International Colloquium on Automata, Languages and Programming (ICALP 2006)*, volume 4051 of *Lecture Notes in Computer Science*, pages 238–249. Springer, 2006.

[28] Klaus Jansen. Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme. *Algorithmica*, 39(1):59–81, 2004.

[29] Klaus Jansen and Marian Margraf. *Approximative Algorithmen und Nichtapproximierbarkeit*. de Gruyter, 2008.

[30] Klaus Jansen and Lorant Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. *Mathematics of Operations Research*, 26(2):324–338, 2001.

[31] Klaus Jansen and Lorant Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. *Algorithmica*, 32(3):507–520, 2002.

[32] Klaus Jansen and Lorant Porkolab. Computing optimal preemptive schedules for parallel tasks: linear programming approaches. *Mathematical Programming*, 95(3):617–630, 2003.

[33] Klaus Jansen and Roberto Solis-Oba. An asymptotic approximation algorithm for 3d-strip packing. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm (SODA 2006)*, pages 143–152. ACM Press, 2006.

[34] Klaus Jansen and Roberto Solis-Oba. New approximability results for 2-dimensional packing problems. In Ludek Kucera and Antonín Kucera, editors, *Symposium on Mathematical Foundations of Computer Science (MFCS 2007)*, volume 4708 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2007.

[35] Klaus Jansen and Roberto Solis-Oba. A polynomial time approximation scheme for the square packing problem. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Proceedings of the 13th International Conference on Integer Programming and*

Combinatorial Optimization (IPCO 2008), volume 5035 of *Lecture Notes in Computer Science*, pages 184–198. Springer, 2008.

[36] Klaus Jansen and Ralf Thöle. Approximation algorithms for scheduling parallel jobs: Breaking the approximation ratio of 2. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, volume 5125 of *Lecture Notes in Computer Science*, pages 234–245. Springer, 2008.

[37] Klaus Jansen and Ralf Thöle. Approximation algorithms for scheduling parallel jobs: Breaking the approximation ratio of 2. Technischer Bericht 0808, Institut für Informatik, Christian-Albrechts-Universität zu Kiel, September 2008.

[38] Klaus Jansen and Rob van Stee. On strip packing with rotations. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the thirty-seventh annual ACM Symposium on Theory of Computing (STOC 2005)*, pages 755–761. ACM Press, 2005.

[39] Klaus Jansen and Guochuan Zhang. Maximizing the number of packed rectangles. In Torben Hagerup and Jyrki Katajainen, editors, *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT 2004)*, volume 3111 of *Lecture Notes in Computer Science*, pages 362–371. Springer, 2004.

[40] Klaus Jansen and Guochuan Zhang. Maximizing the total profit of rectangles packed into a rectangle. *Algorithmica*, 47(3):323–342, 2007.

[41] Klaus Jansen and Hu Zhang. Scheduling malleable tasks with precedence constraints. In Phillip B. Gibbons and Paul G. Spirakis, editors, *Proceedings of the seventeenth annual ACM symposium on Parallelism in Algorithms and Architectures (SPAA 2005)*, pages 86–95. ACM, 2005.

[42] Klaus Jansen and Hu Zhang. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Transactions on Algorithms (TALG)*, 2(3):416–434, 2006.

[43] Berit Johannes. Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling*, 9(5):433–452, 2006.

[44] David S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973.

[45] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer, 2004.

[46] Claire Kenyon and Eric Rémila. A near optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, 2000.

[47] Eugene L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

[48] Renaud Lepère, Denis Trystram, and Gerhard J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. *International Journal of Foundations of Computer Science (IJFCS)*, 13(4):613–627, 2002.

[49] Joseph Y-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[50] Joseph Y.-T. Leung, Tommy W. Tam, C. S. Wong, Gilbert H. Young, and Francis Y. L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990.

[51] Keqin Li and Kam Hoi Cheng. On three-dimensional packing. *SIAM Journal of Computing*, 19(5):847–867, 1990.

[52] Keqin Li and Kam Hoi Cheng. Heuristic algorithms for on-line packing in three dimensions. *Journal of Algorithms*, 13(4):589–605, 1992.

[53] Rongheng Li and Minyi Yue. The proof of FFD(L) ≤ (11/9)OPT(L) + (7/9). *Chinese Science Bulletin*, 42:1262–1265, 1997.

[54] Walter Ludwig and Prasoon Tiwari. Scheduling malleable and nonmalleable parallel tasks. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1994)*, pages 167–176. ACM Press, 1994.

[55] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, 1990.

[56] Flávio K. Miyazawa and Yoshiko Wakabayashi. Packing problems with orthogonal rotations. In Martin Farach-Colton, editor, *Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN 2004)*, volume 2976 of *Lecture Notes in Computer Science*, pages 359–368. Springer, 2004.

[57] Flávio Keidi Miyazawa and Yoshiko Wakabayashi. An algorithm for the three-dimensional packing problem with asymptotic performance analysis. *Algorithmica*, 18(1):122–144, 1997.

[58] Flávio Keidi Miyazawa and Yoshiko Wakabayashi. Approximation algorithms for the orthogonal z-oriented three-dimensional packing problem. *SIAM Journal on Computing*, 29(3):1008–1029, 1999-2000.

[59] Grégory Mounié, Christophe Rapine, and Denis Trystram. A $\frac{3}{2}$-approximation algorithm for scheduling independent monotonic malleable tasks. *SIAM Journal on Computing*, 37(2):401–412, 2007.

[60] Edwin Naroska and Uwe Schwiegelshohn. On an on-line scheduling problem for parallel jobs. *Information Processing Letters*, 81(6):297–304, 2002.

[61] Ingo Schiermeyer. Reverse-fit: A 2-optimal algorithm for packing rectangles. In Jan van Leeuwen, editor, *Proceedings of the Second Annual European Symposium on Algorithms (ESA 1994)*, volume 855 of *Lecture Notes in Computer Science*, pages 290–299, London, UK, 1994. Springer-Verlag.

[62] Jiří Sgall. On-line scheduling. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms — The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 196–231. Springer-Verlag, 1998.

[63] Daniel D. K. Sleator. A 2.5 times optimal algorithm for packing in two dimensions. *Information Processing Letters*, 10(1):37–40, 1980.

[64] A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.

[65] Henning Thomas. On approximative algorithms for a three-dimensional orthogonal knapsack problem. Bachelorarbeit, Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 2006.

[66] John Turek, Joel L. Wolf, and Philip S. Yu. Approximate algorithms for scheduling parallelizable tasks. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 1992)*, pages 323–332, 1992.

[67] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., 2001.

[68] Deshi Ye and Guochuan Zhang. On-line scheduling mesh jobs with dependencies. *Theoretical Computer Science*, 372(1):94–102, 2007.

List of Figures