Useful Structures and How to Find Them

Hardness and Approximation Results for Various Variants of the Parallel Task Scheduling Problem

M.Sc. Malin Rau

Dissertation
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel
eingereicht im Jahr 2019

1. Gutachter:	Prof. Dr. Klaus Jansen Christian-Albrechts-Universität zu Kiel Kiel
2. Gutachter:	Prof. Dr. Nicole Megow Universität Bremen Bremen
3. Gutachter:	Prof. Dr. Grégory Mounié Université Grenoble Alpes Grenoble

Datum der mündlichen Prüfung: 24.5.2019

Zusammenfassung

In dieser Thesis untersuchen wir das Problem Parallel Task Scheduling und einige seiner Varianten. Dieses Problem und seine Variationen haben vielfältige Anwendungen in Theorie und Praxis. Beispielsweise treten sie als Teilprobleme in höherdimensionalen Problemen auf. Im Problem Parallel Task Scheduling erhalten wir eine Menge von Jobs und eine Menge identischer Maschinen. Jeder Job ist ein paralleler Task, d. h. er benötigt eine feste Anzahl der identischen Maschinen, um bearbeitet zu werden. Ein Schedule ordnet den Jobs die Maschinen zu, auf denen sie bearbeitet werden sollen, sowie einen festen Startzeitpunkt der Bearbeitung. Der Schedule ist gültig, wenn zu jedem Zeitpunkt jede Maschine höchstens einen Job bearbeitet. Beim Strip Packing Problem sind die identischen Maschinen in einer totalen Ordnung angeordnet und Jobs können nur benachbarte Maschinen in Bezug auf diese Ordnung nutzen. In dem Single Resource Constraint Scheduling Problem gibt es eine zusätzliche Einschränkung, wie viele Jobs gleichzeitig verarbeitet werden können. Für die genannten Varianten des Parallel Task Scheduling Problems betrachten wir eine Erweiterung, bei der die Maschinen in identische Cluster gruppiert sind. Bei der Bearbeitung eines Jobs dürfen in diesem Modell nur Maschinen aus einem Cluster genutzt werden.

Für all diese Probleme schließen wir Lücken zwischen Nichtapproximierbarkeit und Algorithmen. Für Parallel Task Scheduling zeigen wir, dass es stark NP-vollständig ist, wenn genau 4 Maschinen gegeben sind. Vorher war ein pseudopolynomieller Algorithmus für bis zu 3 Maschinen bekannt, sowie dass dieses Problem stark NP-vollständig ist für 5 oder mehr Maschinen. Für Strip Packing zeigen wir, dass es keinen pseudopolynomiellen Algorithmus gibt, der eine Güte besser als $5/4$ besitzt und geben einen pseudopolynomiellen Algorithmus mit Güte $(5/4 + \varepsilon)$ an. Für Single Resource Constraint Scheduling ist die bestmögliche Güte eine $3/2$-Approximation und wir präsentieren eine $(3/2 + \varepsilon)$-Approximation. Für die Erweiterung auf identische Cluster gibt es keine Approximation

mit Güte besser als 2. Vor unseren Untersuchungen waren bereits Algorithmen mit Güte 2 bekannt, die jedoch gigantische Worst-Case Laufzeiten haben. Wir geben für alle drei Varianten 2-Approximationen mit linearer Laufzeit an, sofern mindestens drei Cluster gegeben sind. Schlussendlich betrachten wir noch Scheduling auf Identischen Maschinen mit Setup Zeiten. Wir entwickeln für drei untersuche Varianten dieses Problems jeweils einen EPTAS, wobei ein EPTAS das beste ist, auf das man hoffen kann, es sei denn es gilt $P = NP$.

Abstract

In this thesis, we consider the Parallel Task Scheduling problem and several variants. This problem and its variations have diverse applications in theory and practice; for example, they appear as sub-problems in higher dimensional problems. In the Parallel Task Scheduling problem, we are given a set of jobs and a set of identical machines. Each job is a parallel task; i.e., it needs a fixed number of identical machines to be processed. A schedule assigns to each job a set of machines it is processed on and a starting time. It is feasible if at each point in time each machine processes at most one job. In a variant of this problem, called Strip Packing, the identical machines are arranged in a total order, and jobs can only allocate neighboring machines with regard to this total order. In this case, we speak of Contiguous Parallel Task Scheduling as well. In another variant, called Single Resource Constraint Scheduling, we are given an additional constraint on how many jobs can be processed at the same time. For these variants of the Parallel Task Scheduling problem, we consider an extension, where the set of machines is grouped into identical clusters. When scheduling a job, we are allowed to allocate machines from only one cluster to process the job.

For all these considered problems, we close some gaps between inapproximation or hardness result and the best possible algorithm. For Parallel Task Scheduling we prove that it is strongly NP-hard if we are given precisely 4 machines. Before it was known that it is strongly NP-hard if we are given at least 5 machines, and there was an (exact) pseudo-polynomial time algorithm for up to 3 machines. For Strip Packing, we present an algorithm with approximation ratio $(5/4 + \varepsilon)$ and prove that there is no approximation with ratio less than $5/4$ unless $\mathrm{P} = \mathrm{NP}$. Concerning Single Resource Constraint Scheduling, it is not possible to find an algorithm with ratio smaller than $3/2$, unless $\mathrm{P} = \mathrm{NP}$, and we present an algorithm with ratio $(3/2 + \varepsilon)$. For the extensions to identical clusters, there can be no approximation algorithm with a ratio smaller than 2 unless $\mathrm{P} = \mathrm{NP}$.

For the extensions of Strip Packing and Parallel Task Scheduling there are 2-approximations already, but they have a huge worst case running time. We present 2-approximations that have a linear running time for the extensions of Strip Packing, Parallel Task Scheduling, and Single Resource Constraint Scheduling for the case that at least three clusters are present and greatly improve the running time for two clusters. Finally, we consider three variants of Scheduling on Identical Machines with setup times. We present EPTAS results for all of them which is the best one can hope for since these problems are strongly NP-complete.

Acknowledgements

I sincerely thank my advisor Klaus Jansen for introducing me to the world of optimization in general and scheduling in particular, for his support and for his insights.

Furthermore, I am very grateful to all my colleagues at the university Alexandra Lassota, Sebastian Berndt, Marten Maack, Lars Rohwedder, Max Deppert, Kilian Grage, Felix Land, Kati Land, Marcin Pal, Ute Iaquinto, Kim-Manuel Klein, and Maren Kaluza for exciting and fun discussions in our lunch breaks. I had a really good time working with you. Special thanks go to Sebastian Berndt, Max Deppert, Alexandra Lassota, Marten Maack, and Bastian Schulz for proofreading parts of this thesis and for helpful discussions on its presentation.

Finally, I want to thank all my family and friends. I thank my parents Kerstin and Stefan for their everlasting support, advise and love. I deeply thank Bastian Schulz for his optimism, motivation and support.

Contents

List of Acronyms

BP Bin Packing

PTS Parallel Task Scheduling

MTS Moldable Task Scheduling

SP Strip Packing

SPR Strip Packing With Rotations

CTS Contiguous Task Scheduling

CMTS Contiguous Moldable Task Scheduling

RCS Resource Constraint Scheduling

SRCS Single Resource Constraint Scheduling

MSRCS Moldable Single Resource Constraint Scheduling

BPCC Bin Packing With Cardinality Constraint

SPCC Strip Packing With Cardinality Constraint

kKP Knapsack With Cardinality Constraint

UkKP Unbounded Knapsack With Cardinality Constraint

MCS Multiple Cluster Scheduling

MSP Multiple Strip Packing

SRCMCS Single Resource Constraint Multiple Cluster Scheduling

MMRS Max-Min-Resource-Sharing

SIM Scheduling on Identical Machines

ETH Exponential Time Hypothesis

Chapter 1

Introduction

This thesis examines problems related to the *Parallel Task Scheduling* problem. Assume you are a leader of a company and you have a set of projects that you would like to realize. For each project, you need a fixed number of staff members that depends on the project. These staff members will deal with its realization for some (previously known) time and cannot work on any other project during this time. (Obviously, your assessment of each project is perfect and you know the exact time it will need to be finished.) However, since there are a lot of projects you want to realize, not all the projects can be worked on at the same time. As the leader of the company, you would like to assign the projects to staff members such that the last finished project will be finished as early as possible.

It is easy to think of various variants of this problem that arise naturally. For example, it might be possible to speed up a given project by assigning more people to it. Another alteration is that all your projects need a project leader. However, not all of your staff members qualify for this position, which adds an additional constraint to the number of projects that can be worked on at the same time. We call these modifications *Moldable Task Scheduling* and *Single Resource Constraint Scheduling* respectively. In another modification, your company has multiple locations around the world and projects can only be executed by teams whose members all work at the same place. This variant of the problem corresponds to a problem called *Multiple Cluster Scheduling*. Finally, we consider a variant, where we have a restriction in the set of people that can work together. For example, your team could be an international team and which members speak different languages. If two people have to work together on a project and they speak different languages, they need a particular third person in the project team who can play the role of a translator. Hence, there are restrictions on the set

of people that can be chosen for a project, while all people have the same skills concerning the realization of the project. This thesis will consider a very restricted variant of this modification. Namely, we will consider a variant where all people are arranged in a total order. To assign two people to a project, all the people in between, with regard to this order, have to work on the same project as well. This restricted variant of the Parallel Task Scheduling is called *Strip Packing*.

All these problems have in common that we do not aim to find an arbitrary solution but one that is good regarding a certain quality-measurement. These kinds of problems are called optimization problems. In our example, this quality measurement is the total time we need to finish the projects with the given staff. In all the variants of Parallel Task Scheduling considered in this thesis, we will optimize the solution with regard to this quality measurement, called the makespan of the schedule. Since Parallel Task Scheduling and the considered variants are all NP-hard, we cannot hope to find solutions that are optimal with regard to the quality measurement. Therefore, we will consider approximation algorithms.

Given an instance I of an optimization problem, we define $\mathrm{OPT}(I)$ as the value of the objective of the optimization problem, e.g., the time we need to finish all the projects in our example. For an algorithm A, we denote by $A(I)$ the value of the solution generated by it. We say an approximation algorithm A has an (*absolute*) approximation ratio α if for each instance I of the problem it holds that $A(I) \leqslant \alpha \cdot \mathrm{OPT}(I)$. If an algorithm A has an approximation ratio of α, we say its result is an α-approximation and sometimes call the algorithm itself α-approximation.

A family of algorithms consisting of algorithms with an approximation ratio $(1+\varepsilon)$ is called polynomial time approximation scheme (PTAS), and a PTAS whose running time is bounded by a polynomial in both the input length $\mathrm{SIZE}(I)$ and $1/\varepsilon$ is called fully polynomial (FPTAS). If the running time of the PTAS is bounded by $(\mathrm{SIZE}(I))^{\mathcal{O}(1)} \cdot f(1/\varepsilon)$, where f is an arbitrary function, we say the running time is *efficient* and call it an efficient PTAS or EPTAS. If the running time of the approximation scheme is not polynomial but pseudo-polynomial, we denote it as pseudo-PTAS or PPTAS. An algorithm A has an *asymptotic* approximation ratio α if there is a constant c such that $A(I) \leqslant \alpha \cdot \mathrm{OPT}(I) + c$ and we denote a polynomial time approximation scheme with respect to the asymptotic approximation

ratio as an A(F)PTAS.

We will present hardness results and algorithmic results for problems that are variants of the Parallel Task Scheduling. As a returning pattern, the algorithms described in this thesis exploit the feature that optimal solutions can be transformed to be well structured. The general approach used to design algorithms in this thesis can be summarized as follows: First, we consider an optimal solution and prove that this solution can be transformed such that it has a well defined simple structure. The algorithm itself guesses this structure and verifies this guess using dynamic or linear programming approaches.

1.1 Contributions and Organization

In Chapter 2, we will introduce the definitions and notations used in this thesis. Each of the problem definitions is supplemented by an overview of the work related to it. This introduction of the problems is followed by a chapter that summarizes some techniques that come in handy when defining algorithms for the considered problems.

In the following chapters, we present the new algorithmic and hardness results. In Chapter 4, we present a hardness result for Parallel Task Scheduling. We prove that the problem is strongly NP-hard when we are given exactly 4 machines. Previously it was known that the problem is solvable in pseudo-polynomial time if the number of machines is at most 3 and that it is strongly NP-hard for more than 5 machines. We close this gap between hardness and solvability.

As a second result in this chapter, we prove that it is strongly NP-hard to solve Strip Packing with an approximation ratio better than $5/4$. Before it was known that there is no pseudo-polynomial algorithm with a ratio better than $12/11$ and the best algorithm was a $4/3+\varepsilon$-approximation.

In Chapter 5, we present a pseudo-polynomial algorithm for the Strip Packing problem. This algorithm closes the gap between inapproximability result and best algorithm since it has an approximation ratio of at most $5/4+\varepsilon$. This algorithm combines techniques used for the $4/3+\varepsilon$-approximation with new techniques to shift and reorder the items. The structure-result proven in this chapter applies to other problem settings

as well, and thus we were able to extend the algorithm to *Strip Packing With Rotations* and *Contiguous Moldable Task Scheduling* which both are generalizations of Strip Packing.

Chapter 6 presents an AFPTAS for the Single Resource Constraint Scheduling problem. In this AFPTAS the additive term is improved to $\mathcal{O}((\log(1/\varepsilon)/\varepsilon)p_{\max})$, whereas previous results for the underlying problem of *Bin Packing With Cardinality Constraint* only could achieve an additive term that was exponentially dependent on $1/\varepsilon$.

In Chapter 7, we present an algorithm for Single Resource Constraint Scheduling that has an absolute approximation ratio of $3/2 + \varepsilon$. This improves the previous best algorithm with an approximation ratio of 2 and nearly matches the lower bound of $3/2$ for this problem. Furthermore, this chapter contains an AEPTAS for Single Resource Constraint Scheduling that has an additive term of $p_{\max}$ only and a running time which is doubly exponential in $1/\varepsilon$.

The Chapter 8 extends the problems mentioned above to identical clusters. In the cluster variant none of the described problems can be approximated better than 2, unless P = NP, and we present linear time 2-approximations for the cluster variants of the problems Parallel Task Scheduling, Strip Packing, and Single Resource Constraint Scheduling when given at least 3 clusters. For the cluster variants of Parallel Task Scheduling and Strip Packing, we can find a 2-approximation as well for one or two clusters, but the running time is no longer linear. While for Single Resource Constraint Scheduling and less than 3 clusters, we present a $2 + \varepsilon$-approximation which is linear in the number of jobs.

In the final Chapter 9, we consider setup variants of the problem *Scheduling on Identical Machines*. More precisely, we consider a variant where the jobs are partitioned into setup classes and variants where we are allowed to split the processing time of a job. These variants are called splittable and preemptive model. Both contain Scheduling on Identical Machines as a special case and therefore are strongly NP-hard. Hence, we cannot hope for FPTAS results. On the other hand, the currently best algorithms for the considered variants are a PTAS for the setup class model, a $3/2$ and a $(4/3 + \varepsilon)$-approximation for the splittable and preemptive model respectively. We present EPTAS results for all three of these variants.

1.2 Publications

Parts of the results in this thesis where published in the following articles:

Journal Publications

[48] Sören Henning, Klaus Jansen, Malin Rau, and Lars Schmarje. "Complexity and inapproximability results for parallel task scheduling and strip packing". In: *Theory of Computing Systems* (2019). ISSN: 1433-0490. DOI: 10.1007/s00224-019-09910-6. URL: https://doi.org/10.1007/s00224-019-09910-6.

[57] Klaus Jansen, Marten Maack, and Malin Rau. "Approximation schemes for machine scheduling with resource (in-)dependent processing times". In: *ACM Trans. Algorithms* 15.3 (2019), 31:1–31:28. DOI: 10.1145/3302250.

[62] Klaus Jansen and Malin Rau. "Improved approximation for two dimensional strip packing with polynomial bounded width". In: *Theoretical Computer Science* 789 (2019), pp. 34–49. DOI: 10.1016/j.tcs.2019.04.002.

Conference Proceedings

[47] Sören Henning, Klaus Jansen, Malin Rau, and Lars Schmarje. "Complexity and inapproximability results for parallel task scheduling and strip packing". In: *Computer Science - Theory and Applications - 13th International Computer Science Symposium in Russia, CSR 2018, Moscow, Russia, June 6-10, 2018, Proceedings*. 2018, pp. 169–180. DOI: 10.1007/978-3-319-90530-3_15.

[52] Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. "Empowering the configuration-ip - new PTAS results for scheduling with setups times". In: *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*. 2019, 44:1–44:19. DOI: 10.4230/LIPIcs.ITCS.2019.44. URL: https://doi.org/10.4230/LIPIcs.ITCS.2019.44.

[58] Klaus Jansen, Marten Maack, and Malin Rau. "Approximation schemes for machine scheduling with resource (in-)dependent processing times". In: *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*. 2016, pp. 1526–1542. DOI: 10.1137/1.9781611974331.ch104.

[60] Klaus Jansen and Malin Rau. "Closing the gap for pseudo-polynomial strip packing". In: *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany.* 2019, 62:1–62:14. DOI: 10.4230/LIPIcs.ESA.2019.62.

[63] Klaus Jansen and Malin Rau. "Improved approximation for two dimensional strip packing with polynomial bounded width". In: *WALCOM: Algorithms and Computation*. Vol. 10167 of LNCS. 2017, pp. 409–420. DOI: 10.1007/978-3-319-53925-6_32.

[64] Klaus Jansen and Malin Rau. "Linear time algorithms for multiple cluster scheduling and multiple strip packing". In: *Euro-Par 2019: Parallel Processing - 25th International Conference on Parallel and Distributed Computing, Göttingen, Germany, August 26-30, 2019, Proceedings*. 2019, pp. 103–116. DOI: 10.1007/978-3-030-29400-7_8.

The full versions of the extended abstracts [60] and [64] can be found on arXiv.org, see [65] and [61].

Declaration on the Own Contribution For each of the publications [48, 61, 63, 65], I contributed most of the ideas and wrote the manuscript. In the publication [58], I contributed most of the ideas to the parts presented in this thesis (see Section 6.2 to 6.4). Furthermore, I wrote the corresponding parts contained in Section 6.2 to 6.4 in this theses and parts of the introduction. In the publication [52], I played a decisive role in the development of basic ideas for the ILP for non-preemptive jobs. Additionally, I contributed ideas to the extension to splittable and preemptive jobs, and dealt with the details and further optimization of the algorithm. Moreover, I created a first sketch for the EPTAS for non-preemptive jobs.

Chapter 2

Overview on the Considered Problems

This section gives an overview on the problems discussed in this thesis and the work related to them. However, we introduce some notation first. For any integer n, we denote the set $\{1, \dots, n\}$ by $[n]$; we write $\log(\cdot)$ for the logarithm with basis 2. Furthermore, for any two sets X, Y, we write Y^X for the set of functions $f : X \to Y$. If X is finite, we say that Y is indexed by X and sometimes denote the function value of f for the argument $x \in X$ by f_x. Furthermore, we will denote the input size of a given instance I by $\text{size}(I)$.

2.1 Parallel Task Scheduling

The first problem that we consider is Parallel Task Scheduling (PTS). In this problem setting, we are given a set $\mathcal{J}$ of n jobs and m identical machines. Each job $j \in \mathcal{J}$ has a processing time $p(j) \in \mathbb{N}$ and requires $m(j) \in \mathbb{N}$ machines.

A schedule S is given by two functions $\sigma : \mathcal{J} \to \mathbb{N}$ and $\rho : \mathcal{J} \to 2^{[m]}$. The function σ maps each job to a start point in the schedule, while ρ maps each job to the set of machines it is processed on. We say a machine i contains a job $j \in \mathcal{J}$ if $i \in \rho(j)$. A schedule $S = (\sigma, \rho)$ is feasible if each machine processes at most one job at a time and each job is processed on the required number of machines. However, if we find a function σ such that

$$\forall t \in \mathbb{N} : \sum_{j \in \mathcal{J} : t \in [\sigma(j), \sigma(j)+p(j))} m(j) \leqslant m,$$

it is always possible to find a function ρ such that the schedule $S = (\sigma, \rho)$

is feasible, which was proven in [71].

The objective is to find a feasible schedule S minimizing the makespan $T := \max_{j \in \mathcal{J}}(\sigma(j) + p(j))$. In the three-field-notation the problem Parallel Task Scheduling is denoted as $P|size_j|C_{\max}$. If the number m of machines is constant, e. g., $m = 3$, we write $P3|size_j|C_{\max}$.

Related Problems In the above description, each job, after started, has to be processed until it is finished. A widely studied alteration is the case that we are allowed to stop the processing of a job and resume its processing at another time. This stopping and resuming comes, depending of the model, at no cost. We differentiate two cases of this alteration. In the first case, we are allowed to process two parts of the same job simultaneously. We call the jobs of this alteration *splittable*. In the other case, a simultaneously processing of two parts of the same job is forbidden. The jobs in this model are called *preemptive* jobs. Note that on the first view these extensions might not seem very natural for Parallel Task Scheduling since the jobs are already parallel jobs; however, a look from this perspective comes in handy when designing algorithms for this problem.

A generalization of the Parallel Task Scheduling is the problem Moldable Task Scheduling (MTS). In this problem setting the jobs do not have a fix machine requirement and a fix processing time. Instead, each job $j \in \mathcal{J}$ has given a set of possible machine requirements $D_j \subseteq \{1, \ldots, m\}$ and a processing time function $p_j : D_j \to \mathbb{N}$ that depends on the assigned number of machines, i.e., for a given number of machines $m_i \in D_j$ the job has a processing time $p_j(m_i)$.

Related Work In 1989, Du and Leung [27] proved Parallel Task Scheduling $Pm|size_j|C_{\max}$ to be strongly NP-complete for all $m \geqslant 5$, while it is solvable by a pseudo-polynomial algorithm for all $m \leqslant 3$. Amoura et al. [6], as well as Jansen and Porkolab [59], presented a PTAS for the case that m is a constant. If m is polynomially bounded by the number of jobs, a PTAS still exists as proven by Jansen and Thöle [69]. Nevertheless, if m is arbitrarily large, the problem gets harder. Since it contains bin packing as a special case, there is no polynomial algorithm with approximation ratio smaller than $\frac{3}{2}$ for Parallel Task Scheduling unless P = NP. For a

more detailed overview on the hardness of several problems related to Parallel Task Scheduling we refer to [26]. Parallel Task Scheduling with arbitrarily large m has been widely studied [34, 105, 83, 32]. The algorithm with the best known absolute approximation ratio of $\frac{3}{2}+\varepsilon$ was presented by Jansen [50]. For work on the moldable case, we refer to the next section.

Open Problems In contrast to the following problems, Parallel Task Scheduling is very well understood and there are only a few open questions left. Besides improving the running time of the existing algorithms whose approximation ratios match the lower bounds, an open question that remained was whether the problem is strongly NP-hard when given precisely four machines.

Open Problem 2.1. Is $P4|size_j|C_{\max}$ solvable in pseudo-polynomial time?

Results In this thesis, we give a negative answer to Open Problem 2.1 by proving that $P4|size_j|C_{\max}$ is strongly NP-hard, see Chapter 4 Theorem 4.1. Furthermore, in Section 8.3, we present an AEPTAS with additive term $p_{\max}$ that has an improved running time compared to the $(3/2+\varepsilon)$-approximation in [50]. Using the same techniques as in [50] for so-called huge jobs, the techniques used in this AEPTAS imply a $(3/2+\varepsilon)$-approximation with this improved running time as well.

2.2 Strip Packing

In the Strip Packing (SP) problem, we have to pack a set $\mathcal{I}$ of rectangular items into a given strip with width $W \in \mathbb{N}$ and infinite height. Each item $i \in \mathcal{I}$ has a width $w(i) \in \mathbb{N}_{\leqslant W}$ and a height $h(i) \in \mathbb{N}$. The area of an item $i \in \mathcal{I}$ is defined as $\text{area}(i) := h(i) \cdot w(i)$ and the area of a set of items $\mathcal{I}' \subseteq \mathcal{I}$ is defined as $\text{area}(\mathcal{I}') := \sum_{i \in \mathcal{I}'} h(i) \cdot w(i)$.

A packing of the items is given by a mapping $\sigma : I \to \mathbb{Q}_{\leqslant W} \times \mathbb{Q}, i \mapsto (x_i, y_i)$, which assigns the lower-left corner of an item $i \in I$ to a position $\sigma(i) = (x_i, y_i)$ in the strip. An inner point of $i \in \mathcal{I}$ (with respect to a packing σ) is a point from the set $\text{inn}(i) := \{(x,y) \in \mathbb{R} \times \mathbb{R} | x_i < x < x_i + w(i), y_i < y < y_i + h(i)\}$. We say two items $i, j \in \mathcal{I}$ overlap if they

share an inner point (i.e., $\text{inn}(i) \cap \text{inn}(j) \neq \emptyset$). A packing is feasible if no two items overlap and if $x_i + w(i) \leqslant W$ for all $i \in I$. The objective of the Strip Packing problem is to find a feasible packing σ with minimal height $h(\sigma) := \max\{y_i + h(i) | i \in \mathcal{I}, \sigma(i) = (x_i, y_i)\}$.

If all item sizes are integral, we can transform each feasible packing to a packing where all positions are integral without enlarging the packing height.This can be done by shifting all items downwards until they touch the upper border of an item or the bottom of the strip. Now all y-coordinates of the items are integral since each is given by the sum of some item heights, which are integral. The same can be done for the x-coordinate by shifting all items to the left as far as possible. Therefore, we can assume that each packing that we consider has the form $\sigma : I \to \mathbb{N}_0 \times \mathbb{N}_0$. problem, we will denote this minimal packing height with $\text{OPT}(I)$ and dismiss the I if the instance is clear from the context.

Related Problems A broadly studied variant is the version of Strip Packing where we are allowed to rotate the items by 90-degrees such that the height of the item becomes the width of the item. Furthermore, variants have been considered, where arbitrary rotations of the items are allowed. However, in this thesis, we will only consider the variant which allows 90 degree rotations and in the following will call this problem Strip Packing With Rotations (SPR).

Note that Strip Packing can be seen as a scheduling problem. We are given $m := W$ machines, which are arranged in a total order, and a set of jobs $\mathcal{J} := \mathcal{I}$ such that each job $i \in \mathcal{I}$ has a processing time $p(i) := h(i)$ and requires $m(i) := w(i)$ machines that are contiguous with respect to the total order of the machines. Since this definition is quite similar to Parallel Task Scheduling, we also speak of Contiguous Task Scheduling (CTS).

Because of the similarities to Parallel Task Scheduling, it is natural to consider the same alterations and generalizations; i.e., we can consider splittable and preemptive jobs and the generalization to moldable jobs. In the setting of moldable jobs, called Contiguous Moldable Task Scheduling (CMTS), we are given a set of jobs $\mathcal{J}$ and a set of m machines. Each job $j \in \mathcal{J}$ can be scheduled on different numbers of machines given by $M_j \subseteq \{1, \ldots, m\}$. Depending on the number of machines $i \in M_j$, each job $j \in \mathcal{J}$ has a specific processing time $p_j(i) \in \mathbb{N}$.

A schedule S is given by three functions: $\sigma : \mathcal{J} \to \mathbb{N}$ which maps each job $j \in \mathcal{J}$ to a starting time $\sigma(j)$; $\rho : \mathcal{J} \to \{1, \dots, m\}$ which maps each job $j \in \mathcal{J}$ to the number of processors $\rho(j) \in M_j$ it is processed on; and $\varphi : \mathcal{J} \to \{1, \dots, m\}$ which maps each job $j \in \mathcal{J}$ to the first machine it is processed on. The job $j \in \mathcal{J}$ will use the machines $\varphi(j)$ to $\varphi(j) + \rho(j) - 1$ contiguously. A schedule $S = (\sigma, \rho, \varphi)$ is feasible if each machine processes at most one job at a time and its makespan is defined by $\max_{j \in \mathcal{J}} \sigma(j) + p_j(\rho(j))$. The objective is to find a feasible schedule, which minimizes the makespan.

Contiguous Moldable Task Scheduling is a true generalization of Strip Packing as it contains Strip Packing (and Strip Packing With Rotations) as a special case: We define the number of machines m as the width of the strip W and for each item $i \in \mathcal{I}$ we introduce one job i with $M_i := \{w(i)\}$ and processing time $p_i(w(i)) = h(i)$ (or introduce one job i with $M_i := \{w(i), h(i)\}$ and processing times $p_i(w(i)) = h(i)$ and $p_i(h(i)) = w(i)$ respectively).

Furthermore, we consider another variant related to Strip Packing, called Multiple Strip Packing, where we have more than one strip and have to minimize the packing height over all the given strips. However, we refer to Section 2.4 for a more detailed overview on this variant of the problem.

Related Work Strip Packing is an important NP-hard problem which has been studied since 1980 (Baker et al. [8]). It arises naturally in many settings as scheduling or cutting stock problems in industrial manufacturing (e.g, cutting rectangular pieces out of a sheet of material as cloth or wood). Recently, it also has been applied to practical problems as electricity allocation and peak demand reductions in smart-grids [73, 94, 102].

In a series of papers [7, 8, 15, 22, 39, 45, 67, 77, 96, 99, 100, 101] algorithms with improved approximation ratios have been presented and $5/3 + \varepsilon$ is the best absolute approximation ratio achieved so far by an algorithm presented by Harren, Jansen, Prädel, and van Stee [44]. On the other hand, by a reduction from the Partition Problem, one can see that it is not possible to find an algorithm with approximation ratio better than $3/2$ unless P = NP.

Two of these algorithms will come in handy later, when we discuss a

new algorithm for Strip Packing or Parallel Task Scheduling. The first is the famous NFDH-Algorithm studied by Coffman et al. [22], which finds a packing with the properties from the following (slightly adapted) lemma.

Theorem 2.2 (See [22]). *For any list L ordered by nonincreasing height it holds that*

$$\mathrm{NFDH}(L) \leqslant 2W(L)/m + p_{max} \leqslant 2 \cdot \mathrm{OPT}(L) + p_{max}.$$

The other algorithm is Steinberg's Algorithm, which we will use to bound the height of an optimal packing from above. It has the following properties:

Theorem 2.3 (See [100]). *Let $w_{\max} := \max_{i \in \mathcal{I}} w(i)$ and $h_{\max} := \max_{i \in \mathcal{I}} h(i)$. If the following inequalities hold,*

$$w_{\max} \leqslant W,\ h_{\max} \leqslant H,\ 2 \cdot \mathrm{area}(\mathcal{I}) \leqslant WH - (2w_{\max} - W)_+(2h_{\max} - H)_+,$$

(where $x_+ := \max\{x, 0\}$) then the items $\mathcal{I}$ can be placed inside a rectangle Q with width W and height H.

In contrast to absolute approximation ratios, asymptotic approximation ratios can get better than $3/2$ and they have been improved in a series of papers [7, 22, 39]. The first asymptotic fully polynomial approximation scheme (in short AFPTAS) was presented by Kenyon and Rémila [77]. It has an additive term of $\mathcal{O}(h_{\max}/\varepsilon^2)$, where $h_{\max}$ is the largest occurring item height. The additive term was improved by Sviridenko [101] and Bougeret et al. [15] to $\mathcal{O}((\log(1/\varepsilon)/\varepsilon)h_{\max})$ simultaneously. Moreover, the running time of this algorithm was improved by Jansen and Kraft [54] to be in $\mathcal{O}(n \log(1/\varepsilon) + \log(1/\varepsilon)^4/\varepsilon^5)$. Furthermore, Jansen and Solis-Oba [67] presented an asymptotic PTAS with an additive term $h_{\max}$ at the expense of the running time, which bounded by $\mathcal{O}(n \log(n)) \cdot n^{1/\varepsilon^{2^{\mathcal{O}(1/\varepsilon)}}}$ operations. Asymptotic algorithms are useful when the maximal occurring item height is small compared to the optimum. However, if the maximal occurring height equals the optimum, the listed algorithms have an approximation ratio of $(2 + \varepsilon)$ or even worse for the AFPTAS results. This motivates the search for algorithms with better approximation ratios in expense of the processing time.

The Partition Problem is solvable in pseudo-polynomial time. Therefore, the lower bound of $3/2$ for absolute approximation ratios does not hold for

pseudo-polynomial algorithms where we allow W to appear polynomially in the running time. The best approximation ratio has been improved step by step [69, 91, 33, 63] with $4/3 + \varepsilon$ as the best absolute approximation ratio achieved so far [33, 63]. On the other hand, we cannot approximate arbitrary in this scenario. Adamaszek et al. [1] proved a lower bound of $12/11$ if $P \neq NP$.

Furthermore, other interesting facts about Strip Packing have been noted. There are differences in the size of the optimal solutions of the same instances for Contiguous Task Scheduling, i.e., Strip Packing, and the closely related Parallel Task Scheduling. Turek et al. [105] first drew attention to these differences and Bladek at al. [10] studied them intensively. Furthermore, the differences in the pseudo-polynomial absolute approximation ratio are notable. While for the contiguous case, we have a lower bound of $5/4$ if P $\neq$ NP, in the non-contiguous case there is a pseudo-PTAS [69].

Strip Packing With Rotations has been studied in the following papers [31, 88, 68, 67]. Algorithms for Strip Packing (without rotations) using the area of the items to prove their ratio, e.g., NFDH, FFDH [8] or Steinberg's algorithm [100], work for Strip Packing With Rotations as well. Furthermore, algorithms using 2D Knapsack with area maximization as a subroutine can also be extended to Strip Packing With Rotations. On the other hand, the lower bounds of $3/2$ for polynomial and $5/4$ for pseudo-polynomial approximation ratios hold for Strip Packing With Rotations as well, unless P $=$ NP.

Next we will consider work related to the generalization Contiguous Moldable Task Scheduling. However, since this problem is also similar to Moldable Task Scheduling and the results for these problems depend on each other we consider research related to this topic as well. Turek, Wolf, and Yu [105] presented an algorithm that assigns jobs to numbers of processors and then schedules the fix instance with known algorithms for these scenarios. This algorithm achieves a 2-approximation for the non-contiguous case and a 2.5-approximation for the contiguous case, using Sleator's algorithm [99] as a subroutine. Furthermore, they pointed out that for improved approximations for the fixed processor instances the algorithm achieves better approximation ratios. More precisely, if the algorithm uses Steinberg's Algorithm [100] as a subroutine instead (which

was not known when the paper was published), it has an approximation ratio of 2 for the contiguous case. The running time of these algorithms was improved by Ludwig and Tiwari [83] from $\mathcal{O}(mn \cdot L(m,n))$ to $\mathcal{O}(mn + L(m,n))$, where $\mathcal{O}(L(m,n))$ is the running time of the used subroutine for the fixed machine instance.

A well-studied variant of these two problems concerns the running time function. We say the running time function is monotonic if the work of the job, i.e., $p_j(m_i) \cdot m_i$, is non-increasing when the number of machines decreases. There is a pseudo-polynomial algorithm with ratio $(3/2 + \varepsilon)$ by Mounié et al. [90] for monotonic non-contiguous moldable jobs. Jansen and Thöle [69] extended the $(3/2 + \varepsilon)$-approximation to non-monotonic contiguous moldable jobs. Furthermore, they presented a pseudo-PTAS for non-monotonic non-contiguous moldable jobs. Together with the FPTAS from [56] for the case $m \geqslant 8n/\varepsilon$, this delivers a PTAS for the case of monotonic non-contiguous jobs. Additionally, the running time of the algorithm by Mounié et al. [90] is improved to be nearly linear by Jansen and Land [56]. A polynomial $(3/2 + \varepsilon)$-approximation algorithm for non-monotonic non-contiguous jobs was presented by Jansen [50], which is arbitrary close to the best possible algorithm for this case unless P = NP.

Open Questions Regarding Strip Packing, there are several open questions. The most interesting are the questions about the best possible absolute approximation ratios of polynomial time and pseud polynomial time algorithms. For polynomial time algorithms, we know that we cannot find an algorithm with approximation ratio better than $3/2$ unless P = NP. On the other hand, the best approximation ratio achieved so far is $5/3 + \varepsilon$.

Open Problem 2.4. Close the gap between the lower bound of $3/2$ and the best known ratio $5/3 + \varepsilon$ for polynomial time algorithms for Strip Packing.

Regarding pseudo-polynomial algorithms, we know that we cannot find an algorithm with approximation ratio better than $12/11$ unless P = NP. On the other hand, the best pseudo-polynomial algorithm so far has a ratio of $4/3 + \varepsilon$, see Figure 2.1 for an overview on the results for this scenario. Hence, we state the second open question for Strip Packing as follows.

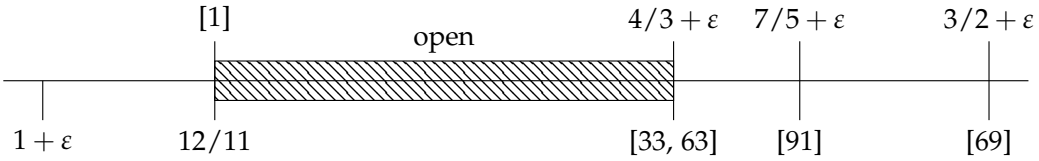


Figure 2.1. Overview on the upper and lower bounds for pseudo-polynomial approximations.

Open Problem 2.5. Close the gap between the lower bound of $12/11$ for pseudo-polynomial time algorithms for Strip Packing and the best known ratio $4/3 + \varepsilon$.

Regarding asymptotic approximation ratios, we can see a discrepancy between algorithms which have a small running time, e. g., the AFPTAS in [54], which have a large additive term of $\mathcal{O}((\log(1/\varepsilon)/\varepsilon)h_{\max})$, and algorithms which have a large running time as the algorithm in [67] but have a small additive term of $h_{\max}$. Note that it is not possible to find an algorithm with approximation guarantee better than $1 + h_{\max}$, which can be seen by the same proof, which shows that it is not possible to approximate Strip Packing with ratio better than $3/2$. Hence, we do not hope to improve the additive term or ratio of the algorithm in [67]. Instead the question that arises is the following:

Open Problem 2.6. What is the best possible running time for an APTAS for Strip Packing with additive term $h_{\max}$, and what is the best additive term that we can achieve with an AFPTAS?

Results In this thesis, we managed to solve Open Problem 2.5 with an exception for a neglectable small ε. Namely, we first improve the lower bound for pseudo-polynomial algorithms to $5/4$, see Chapter 4 Theorem 4.2. This result was published in [47]. Afterward, we first improved the ratio from $7/5 + \varepsilon$ to $4/3 + \varepsilon$ in [63], which was published simultaneously with [33]. Later, we managed to improve this result to a $5/4 + \varepsilon$-approximation, see [61]. In Chapter 5 we present a summary of the results in both papers which lead to the $(5/4 + \varepsilon)$-approximation. Note that the mentioned results (lower and upper bounds) also hold for Strip Packing With Rotations and Contiguous Moldable Task Scheduling.

Furthermore, we considered Open Problem 2.6. We were not able to fully answer this question. However, we have proven that there is an AEPTAS with additive term $h_{\max}$ for Strip Packing, which is a running time class below an APTAS. This result can be found in Chapter 8 Theorem 8.2 and is a requirement to find an improved algorithm for Multiple Strip Packing.

2.3 (Single) Resource Constraint Scheduling

Another problem we consider is the Single Resource Constraint Scheduling (SRCS) problem. In this problem setting, we are given a set $\mathcal{J}$ of n jobs, $m \in \mathbb{N}$ identical parallel machines and a discrete renewable resource $R \in \mathbb{N}$. Each job $j \in \mathcal{J}$ has a processing time $p(j) \in \mathbb{N}$ and a resource requirement $r(j) \in \mathbb{N}$ that has to be met in order to execute the job.

A schedule of these jobs is given by a mapping $\tau : \mathcal{J} \to \mathbb{N}_{\geqslant 0}$ from jobs to starting times. It is *feasible*, if at each point in time $t \in \mathbb{N}$ there are enough machines to schedule the jobs and the total resource requirement of jobs scheduled at t does not exceed the resource limit R, i.e.:

$$\forall t \in \mathbb{N} : \sum_{j : t \in [\tau(j), \tau(j)+p(j))} r_j \leqslant R \tag{2.1}$$

$$\forall t \in \mathbb{N} : |\{j \in \mathcal{J} | t \in [\tau(j), \tau(j) + p(j))\}| \leqslant m \tag{2.2}$$

The objective is to find a feasible schedule $\tau : \mathcal{J} \to \mathbb{N}_{\geqslant 0}$ minimizing the makespan $M := \max_{j \in \mathcal{J}}(\tau(j) + p(j))$.

Related problems There are several natural extensions and alterations to this problem that have been studied. A natural extension to this problem is the Resource Constraint Scheduling (RCS) problem where we are given m identical machines and instead of one, we are given s distinct resource limits $R_1, \ldots, R_s$ such that each job requires an amount of each of the s distinct resources.

Furthermore, the variant of Single Resource Constraint Scheduling, where the resource has to be allocated contiguously, is equivalent to the problem Strip Packing With Cardinality Constraint. Moreover, in the case that we have less jobs than machines, i.e., $n \leqslant m$, Single Resource

Constraint Scheduling cn be reduced to Parallel Task Scheduling as follows: If the number of machines m in Single Resource Constraint Scheduling is larger than n, we can never schedule more than m jobs at the same time, and this constraint is trivially fulfilled in each schedule. We then consider the resource requirement in Single Resource Constraint Scheduling as the required number of machines in Parallel Task Scheduling.

Another variant of this problem is a moldable variant where the processing time depends on the number of resources allocated by the job. More precisely, for each job $j \in \mathcal{J}$ we are given a set $D_j \subseteq \{0, \ldots, R\}$ of resource requirements allowed to be allocated by the job and a processing time function $p(j, r)$ that denotes for each $r \in D_j$ the processing time of job j if it allocates r resources. We call this variant Moldable Single Resource Constraint Scheduling (MSRCS) but it has also been referred to as Parallel Machine Scheduling With Resource Dependent Processing Times [76] or Scheduling With Resource Dependent Processing Times [58].

This problem contains Bin Packing With Cardinality Constraint as a special case. In this problem setting, we are given a set of items $\mathcal{I}$ which each a height $h(i) \leqslant 1$ with $h(j) \in \mathbb{Q}$ and a number k. The objective is to place the items inside as few as possible bins with capacity 1 such that each bin contains at most k items. Given such an instance of Bin Packing With Cardinality Constraint, we define $R := 1$, $r(i) := h(j)$ and scale the values such that all of them are integers. Finally, we define $p(j) = 1$. Minimizing the makespan of the resulting instance of Single Resource Constraint Scheduling corresponds to minimizing the number of bins.

Related Work The first result for Resource Constraint Scheduling was presented in 1975 by Garey and Graham [34]. Given m identical machines and s distinct resource limits such that each job requires an amount of each of the s distinct resources, they have shown that the greedy list algorithm delivers a schedule of length at most $(s + 2 - (2s + 1)/m)\mathrm{OPT}$. This gives an absolute approximation ratio of $(3 - 3/m)$ for the case of $s = 1$ which corresponds to Single Resource Constraint Scheduling. In the same year Garey and Johnson [35] showed that this general scheduling problem is NP-complete even if just one resource is given, i.e., $s = 1$. Lately, Niemeier and Wiese [92] presented a $(2 + \varepsilon)$-approximation for Single Resource Constraint Scheduling, and this is the best known ratio so far. Since this

problem contains Bin Packing With Cardinality Constraint as a special case, there is no algorithm with an approximation guarantee better than $3/2$ for this problem unless P $=$ NP. For Bin Packing With Cardinality Constraint Epstein and Levin [30] presented an AFPTAS.

For Moldable Single Resource Constraint Scheduling the first result was achieved by Grigoriev et al. [42], who studied the unrelated machines variant in which the processing times depend on the machine as well as on the resource assignment. They achieved a 3.75-approximation algorithm. This was improved to $3.5 + \varepsilon$ by Kellerer [76] for the identical machine version. Grigoriev et al. [43] presented a $(3 + \varepsilon)$-approximation for a version of the problem where the jobs are preassigned to machines that also works when the processing time functions are encoded more succinctly. Rather recently, Kling et al. [78] considered a related problem, where for each job j a resource value $\rho_{j,t}$ has to be chosen for each time step t it is processed in. Furthermore, each job j has a resource requirement r_j and a processing volume $p(j)$, and if j receives a resource value of $\rho_{j,t}$ at time step t, exactly $\min_{1,\rho_{j,t}/r_j}$ units of its processing volume are finished during t. They provide a $(2 + 1/(m-2))$-approximation for this case and show that the problem is NP-hard.

Open Problems The problem Single Resource Constraint Scheduling cannot be approximated better than $3/2$ unless P $=$ NP. On the other hand, the best algorithm so far has an approximation ratio of $2 + \varepsilon$. This motivates the following open problem.

Open Problem 2.7. Close the gap between the lower bound of $3/2$ for Single Resource Constraint Scheduling and the best known ratio $2 + \varepsilon$.

For the problem Moldable Single Resource Constraint Scheduling, we can state a similar open problem since the so far best algorithm has an approximation ratio of $3.5 + \varepsilon$, while the lower bound of $3/2$ holds for this generalization as well.

Open Problem 2.8. Close the gap between the lower bound of $3/2$ for Moldable Single Resource Constraint Scheduling and the best known ratio $3.5 + \varepsilon$.

Results In this thesis, we solve the Open Problem 2.7 with exception for a negligibly small ε by presenting an algorithm for Single Resource Constraint Scheduling with a ratio of $3/2 + \varepsilon$. On the other hand, we considered asymptotic algorithms for Single Resource Constraint Scheduling and present an AFPTAS with additive term $\mathcal{O}(\log(1/\varepsilon)/\varepsilon)p_{\max}$ and an AEPTAS with additive term $p_{\max}$. Note that this result directly improves the additive term of the AFPTAS for Bin Packing With Cardinality Constraint by Epstein and Levin [30], which has an additive terms that is exponential in $1/\varepsilon$. While the AFPTAS can be found in Chapter 6, the other results can be found in Chapter 7.

2.4 Identical Clusters

The final problem setting we consider is an extension of the problems Parallel Task Scheduling (PTS), Strip Packing (SP), and Single Resource Constraint Scheduling to identical clusters. While the set of jobs or items respectively remains the same in these settings, the set of machines differ. We are given N identical clusters, each with the properties of the set of machines as before. For the extension of Parallel Task Scheduling (PTS) and Strip Packing (SP), each cluster contains m identical machines. For the extension of Single Resource Constraint Scheduling, each cluster contains m identical machines and R renewable resources. The jobs are allowed to be processed (contiguously) on the machines (and resources) of one cluster only, i.e, they are not allowed to allocate machines (or resources) from more than one cluster. The objective is to find a schedule with minimal makespan.

This extension of Parallel Task Scheduling is called Multiple Cluster Scheduling (MCS), the extension of Strip Packing is called Multiple Strip Packing (MSP), and last the extension of Single Resource Constraint Scheduling is called Single Resource Constraint Multiple Cluster Scheduling (SRCMCS).

Related problems We can extend all problems related to Parallel Task Scheduling to the cluster version of this problem. Hence, we can apply the same variations as to Parallel Task Scheduling or Strip Packing (SP). For

example, we can consider the moldable, splittable, or preemptive versions of these problems.

Related Work Zhuk [109] proved that Multiple Cluster Scheduling and Multiple Strip Packing cannot be approximated better than 2 unless $P = NP$. There is an algorithm by Ye, Han and Zhang [107] which finds a $(2+\varepsilon)$-approximation to the optimal solution for each instance of Multiple Cluster Scheduling or Multiple Strip Packing. This algorithm needs to solve an EPTAS for Scheduling On Identical Machines as a subroutine. The algorithm with the best running time for this problem is currently given by [66] and it is bounded by $2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))} + \text{poly}(n)$. As a result, the running time of the algorithm from Ye, Han and Zhang [107] is bounded by $2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))} + \text{poly}(n)$, using [66] and corresponding 2-approximation algorithms for Parallel Task Scheduling , e.g., the List-Scheduling algorithm by Garay and Graham [34], and Strip Packing, e.g., Steinberg's Algorithm [100]. For Multiple Cluster Scheduling, the approximation ratio of $(2+\varepsilon)$ was improved by Jansen and Trystram [70] to an algorithm with an approximation ratio of 2 and it has a worst case running time of $\Omega(n^{256})$ since it uses an algorithm with running time $n^{\Omega(1/\varepsilon^{1/\varepsilon})}$ using the constant $\varepsilon = 1/4$ as a subroutine. Furthermore, for Multiple Strip Packing there is an algorithm by [14] that has a ratio of 2 as well. The worst case running time of this algorithm is of the form $\Omega(n^{256})$ as well, for the same reasons.

However, since the worst-case running time for these algorithms with an approximation ratio close to or exactly 2 is this enormous, work has been done to improve the runtime at the expense of the approximation ratio. There is a faster algorithm by Bougeret et al. [13] which guarantees an approximation ratio of $5/2$ and has a running time of $\mathcal{O}(\log(np_{\max}) \cdot n \cdot (N + \log(n)))$. Note that the Multifit algorithm for Scheduling on Identical Machines has an approximation ratio of $13/11$ and a running time of at most $\mathcal{O}(n\log(n) + n\log(N)\log(p(\mathcal{J})/N))$, see [108]. Hence using this algorithm as a subroutine in [107], we find a $26/11 \approx 2.364$ approximation in $\mathcal{O}(\text{size}(I)^2 \log(\text{size}(I)))$ operations for each instance I. In [16] Bougeret et al. present an algorithm with approximation ratio $7/3$ and running time $\mathcal{O}(\log(np_{\max})N(n + \log(n)))$. Furthermore, they present a fast algorithm

with approximation ratio 2 and the same running time for the case that the job with the largest machine requirement needs less than $m/2$ machines.

Multiple Cluster Scheduling has also been studied for the case that clusters are not identical, i.e., they do not need to have the same number of machines. It is still NP-hard to approximate this problem better than 2, see [109]. Furthermore, it was proven in [98] and [103] that the List-Schedule cannot guarantee a constant approximation ratio for this problem.

The first algorithm was presented by Tchernykh et al. [103] and has an approximation ratio of 10. This ratio was improved to a 3-approximation by Schwiegelshohn et al. [98], which is given by an online non-clairvoyant algorithm where the processing times are not known beforehand. Later, the algorithm was extended by Tchernykh et al. [104] to the case where jobs have release dates changing the approximation ratio to $2e + 1$. This ratio was improved by Bougeret et al. [12] who developed an algorithm with approximation ratio 2.5 for this case. This algorithm needs the constraint that the largest machine requirement of a job is smaller than the smallest number of machines available in any given cluster. This ratio was improved by Dutot et al. [28] by presenting an algorithm with approximation ratio $(2 + \varepsilon)$. The currently best algorithm for this problem matches the lower bound of 2 [70], but has an enormous running time of $\Omega(n^{256})$.

Best to our knowledge the problem Single Resource Constraint Multiple Cluster Scheduling has not been studied before.

Open Problems Since for Multiple Cluster Scheduling and Multiple Strip Packing algorithms with best possible ratio of 2 are already known, the remaining open question concerns the running time of these algorithms.

Open Problem 2.9. What is the best possible running time for an algorithm for Multiple Cluster Scheduling or Multiple Strip Packing with approximation ratio of 2?

Results We consider the Open Problem 2.9 and find an answer for all the instances where $N \geqslant 3$ by presenting algorithms for Multiple Cluster Scheduling, Multiple Strip Packing, and Single Resource Constraint Multiple Cluster Scheduling with an approximation ratio 2 and running

time $\mathcal{O}(n)$. Furthermore, for $N \in \{1,2\}$, we present 2-approximations for Multiple Cluster Scheduling and Multiple Strip Packing with running time $\mathcal{O}(n\log(n))$ and $\mathcal{O}(n\log^2(n)/\log(\log(n)))$ respectively which correspond to the running times of the 2-approximations in [34] and [100]. Note that the running times for the case $N \in \{1,2\}$ can be improved, when improving the running times of the algorithms in [34] and [100]. On the other hand for Single Resource Constraint Multiple Cluster Scheduling, we present a $(2+\varepsilon)$-approximation with running time $\mathcal{O}(n) \cdot \mathcal{O}_\varepsilon(1)$.

Furthermore, we consider a truly fast algorithm for Multiple Cluster Scheduling since the above mentioned algorithms all hide large constants in the $\mathcal{O}$-notation. The running time of this fast algorithm is dominated by the time it needs to sort the jobs by the number of required machines. It finds a $9/4$-approximation for all instances where N is dividable by three and the ratio converges to $9/4$ for increasing other cluster numbers. All the mentioned algorithms can be found in Chapter 8.

2.5 Setup Times

The final considered problem is not a variant of the Parallel Task Scheduling Problem. Instead, we consider variants of the problem Scheduling on Identical Machines, i.e., $P||C_{\max}$ in the three field notation. In this problem setting, we are given a set $\mathcal{J}$ of n jobs and m identical machines. Each job has a processing time $p(j) \in \mathbb{Q}_{>0}$. A schedule of these jobs assigns each job to a machine it will be processed on; i.e., a schedule is given by a function $\sigma : \mathcal{J} \to [m]$. The objective is to find a schedule σ that minimizes the makespan $C_{\max}(\sigma) := \max_{i\in[m]} \sum_{j\in\sigma^{-1}(i)} p(j)$.

In this thesis, we will consider a variant where the jobs need some setup time before they can be processed. In the first variant, the set $\mathcal{J}$ is partitioned into K sets called classes. Before processing some jobs from a given class, we need a setup time, which can be seen as the time needed to adjust the machine to the following tasks. After this setup is finished, any number of jobs from this class can be processed without needing the setup time again until the machine is adjusted to another class using another setup time. For each job $j \in \mathcal{J}$, we denote its setup class by $k_j \in [K]$ and the setup time of a class $k \in [K]$ as $s(k) \in \mathbb{Q}_{>0}$. As for Scheduling on

Identical Machines a schedule of the jobs $\mathcal{J}$ is defined as a mapping from jobs to machines $\sigma : \mathcal{J} \to [m]$. The jobs assigned to each machine can be scheduled in batches comprised of the jobs of the same class assigned to the machine without overlaps and gaps. Hence, the makespan is given by $C_{\max} = \max_{i\in[m]} \sum_{j\in\sigma^{-1}(i)} p(j) + \sum_{k\in\{k_j \mid j\in\sigma^{-1}(i)\}} s(k)$.

Similar as for Parallel Task Scheduling, we consider a splittable and a preemptive model for this problem as well. In the splittable model, each job can be stopped and restarted at any time. It is even allowed to process multiple parts of this job at the same time. However, before processing any part of the job, we need to process the setup time first. More precisely for each job $j \in \mathcal{J}$, we are given a setup time $s(j) \in \mathbb{Q}_{>0}$ that depends on the job (and no longer on a job class). A schedule of these jobs differs from a schedule as seen before. A job $j \in \mathcal{J}$ can be split into multiple parts. Hence in a schedule, we need to find the number of pieces $\kappa(j) \in \mathbb{N}$ and the fractions of processing times $\lambda_j : [\kappa(j)] \to (0, 1]$ into which it is partitioned such that $\sum_{k\in[\kappa(j)]} \lambda_j(k) = 1$ for each job $j \in \mathcal{J}$. Meaning that the k-th part for $k \in [\kappa(j)]$ has processing time $\lambda_j(k)p(j)$. Finally we need to assign the job peaces to the machines: $\sigma : \mathcal{J}' \to [m]$. Hence, in this case, the output is of the form $(\kappa, \lambda, \sigma)$ and the makespan can be defined as $C_{\max} = \max_{i\in[m]} \sum_{(j,k)\in\sigma^{-1}(i)} s(j) + \lambda_j(k)p(j)$.

Last, we consider the preemptive model. In this scenario it is no longer allowed to process the same job or its setup time at the same time on more than one machine. Similar as in the splittable model, each job $j \in \mathcal{J}$ has a personal setup $s(j)$, and each job $j \in \mathcal{J}$ can be split into multiple parts. Hence a schedule also defines how the jobs are partitioned. The function $\kappa : \mathcal{J} \to \mathbb{Z}_{>0}$ defines how often each job is partitioned and $\lambda_j : [\kappa(j)] \to (0, 1]$ defines the processing time of each part. Meaning that the k-th part for $k \in [\kappa(j)]$ has processing time $\lambda_j(k)p(j)$. Finally, an assignment $\sigma : \mathcal{J}' \to [m]$ along with starting times $\xi : \mathcal{J}' \to \mathbb{Q}_{>0}$ has to be determined such that any two job parts assigned to the same machine or belonging to the same job do not overlap. More precisely, we have to assure that for each two job parts $(j, k), (j', k') \in \mathcal{J}'$ with $\sigma(j, k) = \sigma(j', k')$ or $j = j'$, we have $\xi(j, k) + s(j) + \lambda_j(k)p(j) \leqslant \xi(j')$ or $\xi(j', k') + s(j') + \lambda_{j'}(k)p_{j'} \leqslant \xi(j)$. A schedule is given by $(\kappa, \lambda, \sigma, \xi)$ and the makespan can be defined as $C_{\max} = \max_{(j,k)\in\mathcal{J}'}(\xi(j, k) + s(j) + \lambda_j(k)p(j))$.

Note that in the preemptive and the setup class model, we can assume

that the number of machines is bounded by the number of jobs: If there are more machines than jobs, placing each job on a private machine yields an optimal schedule in both models and the remaining machines can be ignored. This, however, is not the case in the splittable model.

Related problems Note that in the splittable model it is equivalent if the jobs are partitioned into setup classes or each job has its own set up since we are allowed to process the same job multiple times simultaneously. However, in the preemptive model, it makes a difference since jobs from the same class are allowed to be scheduled in parallel while this is forbidden for parts of the same job.

Furthermore, note that the variant of the preemptive model in which overlap between a job part and setup of the same job is allowed is equivalent to the one presented above. This was pointed out by Schuurmann and Woeginger [97] and can be seen with a simple swapping argument.

Furthermore, note that all these problems contain Scheduling on Identical Machines, i.e., $P||C_{\max}$, as a special case since the setup is not allowed to be split. Hence, for a given instance I of $P||C_{\max}$, we can define an instance of the setup problem for the class, splittable, or preemptive model, by defining for each job in I a setup (class) and a job such that the processing time of the job is near zero and the setup time equates the processing time of the corresponding job in I.

Related Work Scheduling on Identical Machines, i.e., $P||C_{\max}$, is a very well studied problem with a long history. It was first studied by Graham [40], and in a long chain of research [4, 5, 40, 49, 51, 53] improved algorithms have been found. The best algorithm to this point is an EPTAS with running time $2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))} + \text{poly}(n)$ by Jansen and Rohwedder [66]. On the other hand, since $P||C_{\max}$ is strongly NP-complete, it is not possible to find an FPTAS. Furthermore, there is a work by Chen, Jansen, and Zhang [20] which states that an EPTAS with a running time $2^{(1/\varepsilon)^{1-\delta}} + n^{\mathcal{O}(1)}$ implies that the Exponential Time Hypothesis (ETH) fails.

There is extensive literature on scheduling problems with setup times. We highlight a few closely related results and otherwise refer to the surveys [2, 3]. The setup class model was first considered by Mäcker et al. [84] in

the special case that all classes have the same setup time. They designed a 2-approximation and additionally a $3/2+\varepsilon$-approximation for the case that the overall length of the jobs from each class is bounded. Jansen and Land [55] presented a simple 3-approximation with linear running time, a $2+\varepsilon$-approximation, and the aforementioned PTAS for the general setup class model. Furthermore, Chen et al. [19] developed a 5/3-approximation for the splittable model. A generalization of this, in which both setup and processing times are job and machine dependent, has been considered by Correa et al. [23]. They achieve a $(1+\phi)$-approximation, where ϕ denotes the golden ratio, using a newly designed linear programming formulation. Moreover, there are recent results concerning Scheduling on Identical Machines in the splittable model considering the sum of the (weighted) completion times as the objective function, e.g. [95, 24]. For the preemptive model, a PTAS for the special case that all jobs have the same setup time has been developed by Schuurman and Woeginger [97]. The $(4/3+\varepsilon)$-approximation for the general case [97] follows the same approach. Furthermore, a combination of the setup class and the preemptive model has been considered, in which the jobs are scheduled preemptively, but the setup times are class dependent. Monma and Potts [89] presented, among other things, a $(2-1/(\lfloor m/2\rfloor+1))$-approximation for this model, and later Chen [18] achieved improvements for some special cases. Lately Deppert and Jansen [25] developed near linear time algorithms with approximation ratio 3/2 for the setup class and the splittable model as well as for the preemptive model with setup classes.

Open Problems As mentioned, the best algorithm for the class model is a PTAS, whereas the best algorithms for the splittable and preemptive model are 3/2 and $(4/3+\varepsilon)$-approximations, respectively. On the other hand, all these problems contain $P||C_{\max}$ as a special case, and, hence, it is not possible to find an FPTAS, unless P = NP, or even an EPTAS with running time smaller than $2^{(1/\varepsilon)^{1-\delta}}+n^{\mathcal{O}(1)}$ unless the ETH fails. This gap between approximation and inapproximation results motivates the following question.

Open Problem 2.10. What is the best possible approximation ratio and running time for algorithms for scheduling with setup times in the class,

splittable and preemptive model?

Results In Chapter 9, we consider Open Problem 2.10 and answer the question about the approximation ratio by presenting an EPTAS for each of these problems, see Theorem 9.1. The running time of these algorithms is bounded by $2^{\mathcal{O}(1/\varepsilon^3 \log^4 1/\varepsilon)} \cdot n^{\mathcal{O}(1)}$ for the setup class model, by $2^{\mathcal{O}(1/\varepsilon^2 \log^3 1/\varepsilon)} \cdot n^{\mathcal{O}(1)}$ for the splittable model, and last but not least by $2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}} \cdot (n \log(m))^{\mathcal{O}(1)}$ for the preemptive model, which is at least for the setup class and the splittable model quite close to the lower bound.

Chapter 3

Algorithmic Concepts

In this Chapter, we introduce some algorithmic concepts that will return repeatedly when designing algorithms for the considered problems. The general approach used to design algorithms in this thesis can be summarized as follows: First we consider an optimal solution and prove that this solution can be transformed such that it has a well defined structure. To find this structure we generally make a small error in the approximation ratio. Then the algorithm guesses the structure of the optimal solution and verifies this guess. For the smallest feasible structure that was found, the algorithm places the items or jobs inside this structure. This again can provoke a small error compared to the optimal solution. In the analysis of the algorithm we have to ensure that the corresponding total error is small enough compared to the aspired approximation ratio.

3.1 Dual Approximation

Many of the algorithms presented in this thesis follow the dual approximation frame-work introduced by Hochbaum and Shmoys [49]: Instead of solving the minimization version of a problem directly, it suffices to find a procedure that for a given bound T on the objective value either correctly reports that there is no solution with value T or returns a solution with value at most $(1+a\varepsilon)T$ for some constant a. If we have some initial upper bound B for the optimal makespan OPT with $B \leqslant b\text{OPT}$ for some b, we can define an approximation algorithm by trying different values T from the interval $[B/b, B]$ in a binary search fashion, and find a value $\text{OPT} \leqslant T^* \leqslant (1+\mathcal{O}(\varepsilon))\text{OPT}$ after $O(\log b/\varepsilon)$ iterations. By applying this technique, we end up with at $(c+\mathcal{O}(\varepsilon))$ approximation. To find the

desired $(c+\varepsilon)\mathcal{O}$ approximation, we scale ε with a corresponding constant before executing the algorithm.

3.2 Rounding Methods

For simplicity of notation, we will look at Strip Packing from the scheduling perspective in the following. There are two dimensions of the jobs that can be rounded: either we round the processing time of a job or we round its machine or resource requirement. Since we are allowed to make an error in the size of the makespan, but we are not allowed to make an error in the total number of allocated machines or resources, the rounding of these sizes differs drastically, as we describe in the following sections.

Rounding Processing Times – Arithmetic vs Geometric Rounding For the processing times, we consider two kinds of rounding arithmetic and geometric rounding. The first technique reduces the smaller processing times drastically, while there are still many large processing times. On the other hand, geometric rounding reduces the large processing times drastically, while it is unsuitable for small processing times. As a consequence, we will use both rounding techniques when restructuring optimal solutions. We will use arithmetic rounding for small processing times, while the geometric grouping is applied to large processing times.

Lemma 3.1. *Consider an instance of Parallel Task Scheduling, Strip Packing, or Single Resource Constraint Scheduling and let T be a lower or upper bound of the optimal makespan of the instance. With an additive loss of at most εT in the approximation ratio, we can assume that each job has a processing time that is a multiple of $\varepsilon T/n$.*

Proof. For this assumption, we add a processing time of at most $\varepsilon T/n$ to the processing time of each job. Since there are at most n jobs and these jobs are scheduled sequentially in the worst case, this adds at most εT to the optimal makespan of the considered instance. □

Lemma 3.2. *Consider an instance I of Parallel Task Scheduling, Strip Packing, or Single Resource Constraint Scheduling with set of jobs $\mathcal{J}$ and let T be a lower or upper bound of the optimal makespan of the instance.*

At a loss of a factor of at most $1+2\varepsilon$ in the approximation ratio, we can ensure that each job $j \in \mathcal{J}$ with $p(j) \geqslant p_0 T$ and $\varepsilon^l T < p(j) \leqslant \varepsilon^{l-1} T$ for some $l \in \mathbb{N}$ has processing time $p(j) = k_j \varepsilon^{l+1} T$ for $k_j = \lceil p(j)/(\varepsilon^{l+1}T) \rceil \in \{1/\varepsilon + 1, \ldots, 1/\varepsilon^2\}$ and a starting time, which is a multiple of $\varepsilon^{l+1}T$ as well.

The jobs can be rounded in $\mathcal{O}(n)$ or $\mathcal{O}(n \log_\varepsilon(p_0))$ operations, depending on the number of operation needed for calculating the logarithm of a given number.

Proof. Consider an optimal schedule for I. We stretch it by a factor of $1+2\varepsilon$; i. e., each point in time τ in the original schedule corresponds to the point $(1+2\varepsilon)\tau$ in the stretched schedule. Let $j \in \mathcal{J}$ be a job with $p(j) \geqslant p_0 T$ and $\varepsilon^{l-1} T \geqslant p(i) \geqslant \varepsilon^l T$. Furthermore, let $\sigma(j)$ and $\varphi(j) := \sigma(j) + p(j)$ be its start and endpoints, respectively, in the original schedule. We define the stretched start and end points as $\bar{\sigma}(j) := (1+2\varepsilon)\sigma(j)$ and $\bar{\varphi}(j) := (1+2\varepsilon)\varphi(j)$. As a consequence, we have $\bar{\varphi}(j) - \bar{\sigma}(j) = (1+2\varepsilon)(\varphi(j) - \sigma(j)) = (1+2\varepsilon)p(j)$.

Next, we change the start and end times of j in the stretched schedule to $\sigma'(j) := \bar{\sigma}(j) + \varepsilon p(j)$ and $\varphi'(j) := \bar{\varphi}(j) - \varepsilon p(j)$. As a result it holds that $\varphi'(j) - \sigma'(j) = \bar{\varphi}(j) - \bar{\sigma}(j) - 2\varepsilon p(j) = (1+2\varepsilon)p(j) - 2\varepsilon p(j) = p(j)$. We have $p(j) \geqslant \varepsilon^l T$, which implies $\bar{\varphi}(j) - \varphi'(j) = \sigma'(j) - \bar{\sigma}(j) = \varepsilon p(j) \geqslant \varepsilon^{l+1} T$. This ensures that for the interval $[\varphi'(j), \bar{\varphi}(j)]$ there is an integer k_φ such that $k_\varphi \varepsilon^{l+1} T \in [\varphi'(j), \bar{\varphi}(j)]$, analogously there exists an integer k_σ such that $k_\sigma \varepsilon^{l+1} T \in [\bar{\sigma}(j), \sigma'(j)]$. We change the start and end point of job j to $\sigma''(j) := k_\sigma \varepsilon^{l+1} T$ and $\varphi''(j) = k_\varphi \varepsilon^{l+1} T$.

It is possible that $\varphi''(j) - \sigma''(j) > p'(j) := \lceil p(j)/(\varepsilon^{l+1}T) \rceil \varepsilon^{l+1} T$. In this case, we increase $\sigma''(j)$ such that $p'(j) = \varphi''(j) - \sigma''(j)$. Since the rounded job j is scheduled inside the stretched version of itself, we do not create any conflicts with the constraints of the considered scheduling (or packing) problem.

Note that $\lceil p'(j)/(\varepsilon^{l+1}T) \rceil \in \{1/\varepsilon, \ldots, 1/\varepsilon^2\}$ since $\varepsilon^{l-1} T \geqslant p(j) \geqslant \varepsilon^l \mathrm{OPT}$. We round only jobs with processing time larger than $p_0 T$. Therefore, we have at most $\log_\varepsilon(p_0)$ different intervals, which contain at most $1/\varepsilon^2$ sizes resulting in $\mathcal{O}(\log_\varepsilon(p_0)/\varepsilon^2)$ different sizes total. The rounded size of an item can be determined by $p'(j) := \lceil p(j)/(\varepsilon^{l+1}T) \rceil \varepsilon^{l+1} T$ where $l := \lceil \log_\varepsilon(p(j)/T) \rceil$. Assuming $\log(x)$ can be calculated in $\mathcal{O}(1)$, we need at most $\mathcal{O}(n)$ operations for this rounding step. However, if we assume that we need $\mathcal{O}(\log(x))$ operations to calculate $\log(x)$, we need at most

$\mathcal{O}(n \log_\varepsilon(p_0)) = \mathcal{O}(n \log_{1/\varepsilon}(1/p_0)) = \mathcal{O}(n \log(1/p_0)/\varepsilon)$ operations. □

Rounding the machine requirements – Linear and Geometric Grouping The techniques presented in this section were first introduced by Fernandez de la Vega and Lueker for the problem Bin Packing [106] and later extended by Kenyon and Rémila [77] for Strip Packing. The rounding techniques described in this section are usually applied just to jobs with a small processing time or, when interested in an AFPTAS, they are applied to all the jobs.

In the flowing, when speaking of a fractional schedule or fractional scheduled jobs, we refer to the jobs scheduled as splittable jobs. The following Lemma is presented from the view of Parallel Task Scheduling. However, as mentioned before, we can substitute the number of machines m by the number of resources or the with of the strip and jobs with items, to transfer the following lemma to Single Resource Constraint Scheduling and Strip Packing respectively.

Lemma 3.3. *Let $\mathcal{J}$ be a set of jobs, m be the number of machines and $\delta, \rho \in (0,1)$. Furthermore, let $\mathcal{J}_\delta$ be the set of all jobs with machine requirement larger than δm.*

Using linear grouping, we can introduce a new set of jobs $\bar{\mathcal{J}}_\delta$ with the following properties:

- $|\bar{\mathcal{J}}_\delta| \leqslant 1/\rho\delta$
- *Given a (fractional) schedule of the jobs in $\mathcal{J}$ the jobs in $\bar{\mathcal{J}}_\delta$ can be scheduled fractionally in the places of the jobs in $\mathcal{J}_\delta$ and one extra box of width m and height at most $\rho \cdot \text{work}(\mathcal{J}_\delta)/m \leqslant \rho\text{OPT}(\mathcal{J}_\delta)$.*

  This rounding can be done in $\mathcal{O}(|\mathcal{J}_\delta| \log(1/\rho\delta))$.

Proof. In this proof, we assume that $|\mathcal{J}_\delta| \geqslant 1/\delta\rho$ since, otherwise, we do not need to round the items.

The jobs in $\bar{\mathcal{J}}_\delta$ are constructed as follows, see Figure 3.1. First, we stack all the jobs in $\mathcal{J}_\delta$ ordered by size such that the job using the most machines is on the bottom of the stack. In the next step, we introduce segments of height $\rho\delta \cdot p(\mathcal{J}_\delta)$, i. e., we draw horizontal lines at the multiples of $\rho\delta \cdot p(\mathcal{J}_\delta)$ and the segment between the multiples $i \cdot \rho\delta \cdot p(\mathcal{J}_\delta)$ and $(i+1) \cdot \rho\delta \cdot p(\mathcal{J}_\delta)$

is called segment i. For each of these segments we introduce one job with height $\rho\delta \cdot p(\mathcal{J}_\delta)$ and machine requirement of the widest job intersecting this segment. The set of all these jobs is called $\bar{\mathcal{J}}_\delta$. Obviously, we have $|\bar{\mathcal{J}}_\delta| \leqslant 1/\rho\delta$. Note that in the case that one job is the size defining job of more than one segment, we introduce one large job for all the segments together. For the sake of simplicity, however, we assume in the following that such a job does not exist.

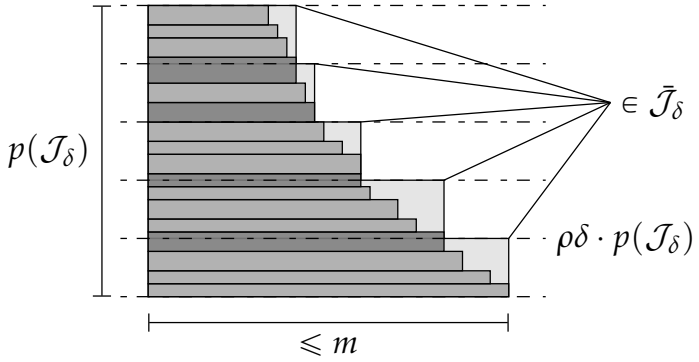


Figure 3.1. Linear grouping for $\mathcal{J}_\delta$

The jobs in $\bar{\mathcal{J}}_\delta$ can be scheduled as a splittable jobs instead of the jobs in $\mathcal{J}_\delta$ and an extra box of width m and height at most $\rho\delta \cdot p(\mathcal{J}_\delta)$. This can be done by scheduling the job for segment i splitted instead of the jobs intersected by segment $(i-1)$. The resulting schedule is feasible since all the jobs in segment $(i-1)$ have a machine requirement which is at least as large as the machine requirement of the rounded job for segment i. The jobs from segment 0 cannot be scheduled instead of other jobs and we schedule the job for this segment inside the extra box of height $\rho\delta \cdot p(\mathcal{J}_\delta)$. Note that $\text{work}(\mathcal{J}_\delta) \geqslant \delta m \cdot p(\mathcal{J}_\delta)$ and hence $\delta \cdot p(\mathcal{J}_\delta) \leqslant \text{work}(\mathcal{J}_\delta)/m \leqslant \text{OPT}_{\mathcal{J}_\delta}$.

Note that sorting the jobs needs $\mathcal{O}(|\mathcal{J}_\delta| \log(|\mathcal{J}_\delta|))$ time, hence to round the jobs in $\mathcal{O}(|\mathcal{J}_\delta| \log(1/\mu\delta))$ we need a smarter idea. To find the rounded jobs, we search only for the $1/\rho\delta$ values for the machine requirement and use a modified Selection-Algorithm to find them. Note that selecting the ith element with respect to any order has a running time of $\mathcal{O}(n)$ [11], where n is the number of considered items.

Consider the following approach: First, we search for the job j_m with the median machine requirement in $\mathcal{O}(|\mathcal{J}_{S,W}|)$. By summing up all the

processing times of jobs with larger machine requirement (in $\mathcal{O}(n)$ time), we can identify the number i of the segment, which the job j_m intersects. If j_m intersects a multiple of $\rho\delta \cdot p(\mathcal{J}_\delta)$, we have found a size defining job and iterate with the both generated subsets. Otherwise, we search for the group size of the group containing this job in $\mathcal{O}(|\mathcal{J}_{S,W}|/2)$ and the group size above this group (if existing) in $\mathcal{O}(|\mathcal{J}_{S,W}|/2)$ as well. We do this by using the ith-Element-Algorithm again but instead of searching for the ith element, we search for the job which intersects $i\rho\delta \cdot p(\mathcal{J}_\delta)$. The set of jobs where we do not know the rounded size is now partitioned into two sets containing at most $|\mathcal{J}_{S,W}|/2$ jobs each. We iterate the process on both sets separately until each group size is found.

Claim 3.4. The described algorithm to find the rounded values has a running time of at most $\mathcal{O}(|\mathcal{J}_\delta| \log(d))$, where d is the number of segments.

We consider the following recurrence equation

$$\begin{aligned} T(n,1) &\leqslant cn \\ T(n,d) &\leqslant T(n/2,d') + T(n/2,d'') + cn, \text{for } d' + d'' < d \end{aligned}$$

where n is the number of considered jobs and $c \in \mathbb{N}$. To find the job with the median machine requirement and the group sizes of the group containing this item as well as the group above, we need $\mathcal{O}(n)$ operations and hence there is a $c \in \mathbb{N}$ with these properties. Afterward the set of jobs is partitioned into two sets such that each set contains at most $n/2$ jobs. The total number of sizes we search for (i. e., $d' + d''$) is reduced by at least one since in this step we find one or two of them (i. e., $d' + d'' < d$). However, the values we search for do not have to be distributed evenly to the sets. Therefore, this recurrence equation represents the running time of the described algorithm adequately.

We claim that $T(n,d) \leqslant cn(\log(d)+1)$. We have $T(n,1) = cn = cn(\log(1)+1)$ and hence the claim is true for $d = 1$. For $d \in \mathbb{N}_{\geqslant 2}$ it follows that

$$\begin{aligned} T(n,d) &\leqslant T(n/2,d') + T(n/2,d'') + cn \\ &\leqslant c(n/2)(\log(d')+1) + c(n/2)(\log(d'')+1) + cn \\ &\leqslant c(n/2) \cdot (\log(d') + 1 + \log(d'') + 1) + cn \\ &= c(n/2)(\log(d'd'')) + 2cn \end{aligned}$$

$$\begin{aligned}&\leqslant c(n/2)(\log((d/2)^2)) + 2cn\\&= cn\log(d) + cn.\end{aligned}$$

Since we have $d = \rho\delta$ and $n = |\mathcal{J}_\delta|$ this concludes the proof of this lemma. □

By using geometric grouping instead of linear grouping, we can reduce the total number of rounded sizes some more, without increasing the error to much.

Lemma 3.5. *Let m be the number of machines or resources given or be the width of the strip. Furthermore, let $\mathcal{J}_\delta$ be the set of all jobs or items with machine requirement or width larger than δm. Furthermore, let $\rho \in (0,1)$.*

Using geometric grouping we can introduce a set of rounded jobs $\bar{\mathcal{J}}_\delta$, where we round the machine requirement of each job $j \in \mathcal{J}_\delta$ to a machine requirement $\bar{m}(j) \geqslant m(j)$. For this set of rounded jobs it holds that $|\{m(j)|j \in \bar{\mathcal{J}}_\delta\}| \leqslant \lceil\log(1/\delta)\rceil/\rho$. Furthermore, the set of rounded jobs can be scheduled as splittable jobs instead of the jobs in $\mathcal{J}_\delta$ and an extra box of width m and height at most $2\rho \cdot \text{work}(\mathcal{J}_\delta)/m \leqslant 2\rho\text{OPT}_{\mathcal{J}_\delta}$.

This rounding can be done in $\mathcal{O}(|\mathcal{J}_\delta|(\log(1/\rho) + \log(1/\delta)))$.

Proof. In this proof, we assume that $|\mathcal{J}_\delta| \geqslant \log(1/\delta)/\rho$ since, otherwise, we do not need to round the items. The difference between geometric grouping and linear grouping is an additional partitioning step prior to the steps of the linear grouping, as described below.

To generate the set $\bar{\mathcal{J}}_\delta$, we first introduce a set $\tilde{\mathcal{J}}_\delta$ and partition the set $\mathcal{J}_\delta$ into the following $\lceil\log(1/\delta)\rceil$ sets $\mathcal{J}_{\delta,i} := \{i \in \mathcal{J}_\delta | m/2^{i+1} \leqslant m(i) < m/2^i\}$. (Jobs j with $m(j) = m$ do not need to be rounded and can be scheduled at the end of the schedule without increasing the makespan). This partition can be done in $\mathcal{O}(n\log(1/\delta))$ time. For each of these sets, we perform the steps of linear grouping with a customized adjustment to the height of the segments per set. For these adjusted heights, we use the fact that it is possible to schedule at least 2^i and at most 2^{i+1} jobs from the set $\mathcal{J}_{\delta,i}$ at the same time.

For each $\mathcal{J}_{\delta,i}$, we stack the contained items in order of decreasing machine requirement and partition this stack into $1/\rho$ segments of size $\rho \cdot p(\mathcal{J}_{\delta,i})$, where $p(\mathcal{J}_{\delta,i})$ is the total processing time of the items in $\mathcal{J}_{\delta,i}$. We

define a new job for each segment which has height $\rho \cdot p(\mathcal{J}_{H,i})$ and width of the widest job intersecting this segment, see Figure 3.1. The widest rounded job will be placed at the end of the schedule inside a new box. Since we are allowed to split this job, we can place at least 2^i of these parts next to each other, we need at most $\rho \cdot p(\mathcal{J}_{\delta,i})/2^i$ additional height to place this item.

To place all the largest rounded items from each set $\mathcal{J}_{\delta,i}$, we introduce a new box for horizontal items. We define the boxes height as $\rho \sum_{i=0}^{\lceil \log(1/\delta) \rceil} p(\mathcal{I}_{H,i})/2^i$. For each $i \in \mathbb{N}$, the total machine requirement of 2^{i+1} jobs from the set $\mathcal{J}_{\delta,i}$ is larger than m and hence

$$\sum_{i=0}^{\lceil \log(1/\delta) \rceil} p(\mathcal{I}_{H,i})/2^{i+1} \leqslant \text{work}(\mathcal{J}_\delta)/m.$$

Therefore, the processing time of the introduced box is bounded by $2\rho \cdot \text{work}(\mathcal{J}_\delta)/m$ and the total number of different item widths is bounded by $\lceil \log(1/\delta) \rceil / \rho$.

Regrading the running time, as seen above in the proof of Lemma 3.3, the size defining items can be found in $\mathcal{O}(|\mathcal{J}_{\delta,i}| \log(1/\rho))$ for each set $\mathcal{J}_{\delta,i}$. Therefore, all the sizes can be found in $\mathcal{O}(\sum_{i=0}^{\lceil \log(1/\delta) \rceil} |\mathcal{J}_{\delta,i}| \log(1/\rho)) = \mathcal{O}(|\mathcal{J}_\delta| \log(1/\rho))$.

To introduce the set $\bar{\mathcal{J}}_\delta$, we again consider each job $j \in \mathcal{J}_\delta$. This job might overlap more than one segment. We define the rounded resource requirement of this job as the resource requirement of the job we introduced for the most narrow segment this job overlaps. Since for each segment we introduced a job with a machine requirement of the widest job intersecting this segment, the rounded resource requirement of this job is at least as large as the original resource requirement. Obviously it is possible to schedule the jobs in $\bar{\mathcal{J}}_\delta$ as splittable jobs instead of the jobs from the set $\tilde{\mathcal{J}}_\delta$ without increasing the makespan since we did not enlarge the machine requirement with regard to the corresponding segment in $\tilde{\mathcal{J}}_\delta$.

□

Sometimes the jobs have the additional condition that they all have the same height. In this case, we use linear grouping as it is applied in the Bin Packing problem.

Lemma 3.6. *Let m be the number of machines or resources given or be the width of the strip. Furthermore, let $\mathcal{J}_\delta$ be the set of all jobs or items with machine requirement or width larger than δm. Furthermore, let $\rho \in (0,1)$. We consider the case that all the jobs in $\mathcal{J}_\delta$ have the same processing time p.*

Using geometric grouping we can introduce a new set of jobs $\bar{\mathcal{J}}_\delta$ width $|\bar{\mathcal{J}}_\delta| = |\mathcal{J}_\delta|$ and $|\{m(j)|j \in \bar{\mathcal{J}}_\delta\}| \leqslant \lceil \log(1/\delta) \rceil / \rho$ such that the jobs in $\bar{\mathcal{J}}_\delta$ all have processing time p and can be scheduled instead of the jobs in $\mathcal{J}_\delta$ and an extra box of width m and height at most $2\rho \cdot \text{work}(\mathcal{J}_\delta))/m + 2p \leqslant 2\rho \text{OPT}_{\mathcal{J}_\delta} + 2p$.

This rounding can be done in $\mathcal{O}(|\mathcal{J}_\delta|(\log(1/\rho) + \log(1/\delta)))$.

Proof. The proof of this lemma works exactly as the proof of the lemma above. The only difference is in choosing the sizes of the segments. Instead of choosing the height $\rho \cdot p(\mathcal{J}_{\delta,i})$ for the segment, the height is chosen such that the segment contains exactly $\lceil \rho \cdot p(\mathcal{J}_{\delta,i})/p \rceil$ jobs. The only exception is the segment containing the most narrow jobs, which only contains the residual jobs. As a result each segment has a height of at most $\rho \cdot p(\mathcal{J}_{\delta,i}) + p$ and therefore the height of the extra box we need to schedule the widest jobs from each set $\mathcal{J}_{\delta,i}$ fractionally is bounded by

$$\sum_{i=0}^{\lceil \log(1/\delta) \rceil} (\rho \cdot p(\mathcal{I}_{H,i}) + p)/2^i \leqslant 2\rho \cdot \text{work}(\mathcal{J}_\delta))/m + 2p.$$

□

3.3 Configuration Linear Program

In this chapter, we will see two variants of Configuration Linear Programs (Configuration LPs) which will appear repeatedly in this thesis and introduce them using the example of Parallel Task Scheduling. In general a configuration is a multiset of jobs, which can either be scheduled at the same time, without violating the constraints given by the problem, or can be scheduled one after another without exceeding the anticipated makespan. However, in the variants of configuration LPs described in this section, we consider configurations that represent a set of jobs that is scheduled at the same time.

Let $\mathcal{J}$ be a set of jobs that we want to schedule using a configuration

LP and m be the number of available machines. A configuration of jobs in $\mathcal{J}$ is a multiset $C := \{a_j : j | j \in \mathcal{J}\}$. We say a configuration has a machine requirement of $m(C) := \sum_{j \in \mathcal{J}} a_j \cdot m(j)$ and define $\mathcal{C}_m$ as the set of all configurations with machine requirement at most m. Note that for Strip Packing we can use the same description of configurations, while for Single Resource Constraint Scheduling the definition of configurations differs in the regard that they have a bound on the machine requirement and on the resource requirements as well.

The first configuration LP to be presented is a minimization version which minimizes the total processing time of all the configurations while scheduling all the jobs as splittable jobs. For each $C \in \mathcal{C}_m$ we introduce one variable X_C which defines the processing time of the configuration C. We call the following linear program $LP_{conf,\min}(\mathcal{J}, m)$:

$$\min \sum_{C \in \mathcal{C}_m} X_C \tag{3.1}$$

$$\sum_{C \in \mathcal{C}_m} X_C \cdot a_{j,C} = p(j) \qquad \forall j \in \mathcal{J} \tag{3.2}$$

$$X_C \geqslant 0 \qquad \forall C \in \mathcal{C}_m \tag{3.3}$$

The sum over all configuration processing times in (3.1) represents the makespan of the schedule generated by a solution to the Linear Program. Equation (3.2) ensures that the sum of all parts where job j is scheduled equates the processing time of this job.

Observation 3.7. An optimal schedule of the jobs $\mathcal{J}$ has a makespan which is at least as large as the optimal solution of this linear program since each schedule can be transformed to a solution for $LP_{conf,\min}(\mathcal{J}, m)$.

This transformation can be done by dividing the schedule at the start and endpoints of the jobs into segments such that during the processing of each segment s the same set of jobs is processed. The set of jobs in segment s defines a configuration C_s and we can set X_{C_s} to the processing time of this segment $p(s)$.

In contrast to the first LP $LP_{conf,\min}$ in the second LP we do not have a minimization problem. Instead, we know the processing time we are allowed to use for the configurations beforehand. Namely, we are given a set of segments $\mathcal{S}$. Each of these segments $s \in \mathcal{S}$ has a processing time

$p(s)$ and during the processing time of this segment there are exactly $m(s)$ machines available. (In the case of Single Resource Constraint Scheduling the available resource $R(s)$ is constant as well during the processing of the segment s.) The linear program we consider, called $LP_{conf}(\mathcal{J}, \mathcal{S})$, has the following form:

$$\sum_{C \in \mathcal{C}_{m(s)}} X_{C,s} = p(s) \qquad \forall s \in \mathcal{S} \tag{3.4}$$

$$\sum_{s \in \mathcal{S}} \sum_{C \in \mathcal{C}_{m(s)}} X_{C,s} a_{j,C} = p(j) \qquad \forall j \in \mathcal{J} \tag{3.5}$$

$$X_{C,s} \geqslant 0 \qquad \forall s \in \mathcal{S}, C \in \mathcal{C}_{m(s)} \tag{3.6}$$

Similar as above we introduce for each segment $s \in \mathcal{S}$ and each configuration $C \in \mathcal{C}_{m(s)}$ a variable $X_{C,s}$ which represents the processing time of the configuration C in the segment s. The first equation (3.4) ensures that the sum of the chosen processing times of the configurations does not exceed the processing time of the segment while the second equation (3.5) ensures that each job in $\mathcal{J}$ is processed completely inside the configurations.

Solving the Configuration LPs approximately In the following we will be interested in solving the above configuration LPs fast while sacrificing some of the accuracy. Let OPT_{LP} be the size of the optimal solution for the first LP $LP_{conf,\min}$, i. e.,

$$OPT_{LP} := \min_{X} \sum_{C \in \mathcal{C}_m} X_C, \; X \in \{\mathbb{R}_+^{|\mathcal{C}_m|} \mid \sum_{C \in \mathcal{C}_m} X_C \cdot a_{j,C} = p(j) \forall j \in \mathcal{J}\}.$$

We are interested in finding a solution $\hat{X}$ with $\sum_{C \in \mathcal{C}_m} \hat{X}_C \leqslant (1+\varepsilon) OPT_{LP}$. While in the second configuration LP, LP_{conf}, we are interested in solving a relaxed version, where we replace the first equation by

$$\sum_{C \in \mathcal{C}_{m(s)}} X_{C,s} \leqslant (1+\varepsilon) p(s) \;\; \forall s \in \mathcal{S}. \tag{3.7}$$

In the following, we describe how to find a solution to the relaxed versions of these Configuration LPs using an algorithm for the Max-Min-Resource-Sharing problem.

In the Max-Min-Resource-Sharing problem considered in this section,

we are given a vector $f : B \to \mathbb{R}_+^M$ of M nonnegative continuous concave functions defined on a nonempty convex compact set B and we search for the value

$$\lambda^* = \max\{\lambda | f(x) \geqslant \lambda \cdot \mathbb{1}_M, x \in B\},$$

where $\mathbb{1}_M$ is the vector of size M with 1 at each position. The problem is called *block-angular* if $B = B^1 \times \ldots \times B^K$ for $K > 1$, where B^k is a non empty convex compact set for each $k = 1, \ldots, K$ and we have $f_m(x) = \sum_{k=1}^K f_m^k(x^k)$, where $f_m^k(x^k)$ are given continuous nonnegative concave functions for $x^k \in B^k$ and $m = 1, \ldots, M$.

There is an algorithm by Grigoriadis et al. [41] that computes an $x \in B$ satisfying $f_j(x) \geqslant (1-\varepsilon)\lambda^*$ for each $j \in \mathcal{J}$ and a given ε. The algorithm iterates the following steps: Given an initial solution $\check{x}$, the algorithm computes a profit vector $g = g(\check{x}) \in \mathbb{R}^M$ for the current value $\check{x}$. After that, an $(1-\varepsilon')$-approximative solution $\hat{x}$ of the problem $\max\{g^T f(x) | x \in B\}$ has to be computed, where ε' depends linearly on ε. This problem is called block-problem, denoted by $\mathcal{ABS}(g, \varepsilon')$, and a solver has to be provided. In the block-angular case we have to find a $(1-\varepsilon')$-approximative solution to $\max\{g^T f^k(x^k) | x^k \in B^k\}$ for each $k = 1, \ldots, K$ independently. In the last step of an iteration, the new value of the vector $\check{x}$ is set to $(1-\tau)\check{x} + \tau\hat{x} \in B$, where $\tau \in (0,1)$ is an appropriate step length. One of these iterations, where we update $\check{x}$, is called a coordination step and the algorithm needs at most $\mathcal{O}(M(\ln(M) + \varepsilon^{-2}))$ of them.

We now consider how we can translate the configuration LPs to this Max-Min-Resource-Sharing problem. For $LP_{conf,\min}(\mathcal{J}, m)$, we define the number of functions as $M := |\mathcal{J}|$. Furthermore, we define

$$B := \{x \in \mathbb{R}^{|\mathcal{C}_m|} | \sum_{C \in \mathcal{C}_m} x_C = 1\}$$

and

$$f_j(x) := \sum_{C \in \mathcal{C}_m} x_C \cdot \frac{a_{C,j}}{p(j)} \quad \forall j \in \mathcal{J}.$$

The resulting block-problem is defined as

$$\max\{g^T f(x) | x \in B\} = \max\{\sum_{C \in \mathcal{C}_m} x_C \cdot \sum_{j \in \mathcal{J}} g_j \frac{a_{C,j}}{p(j)} | x \in B\}.$$

The above problem is maximized if we find the configuration $C' \in \mathcal{C}_m$

that maximizes $\sum_{j\in\mathcal{J}} g_j \frac{a_{C,j}}{p(j)}$ and define $x'_C = 1$ for this configuration and $x_C = 0$ for all the other configurations in $\mathcal{C}_m$.

On the other hand, for $LP_{conf}(\mathcal{J},\mathcal{S})$ we have the block-angular case and define $M := |\mathcal{J}|$, $K := |\mathcal{S}|$,

$$B^s := \{x^s \in \mathbb{R}^{|\mathcal{C}_{m(s)}|} | \sum_{C\in\mathcal{C}_{m(s)}} x^s_C = p(s)\} \quad \forall s \in \mathcal{S}$$

and

$$f^s_j(x) := \sum_{C\in\mathcal{C}_{m(s)}} x^s_C \cdot \frac{a_{C,j}}{p(j)} \quad \forall j \in \mathcal{J}, s \in \mathcal{S}.$$

Similar as above the corresponding block-problems are defined as follows

$$\max\{g^T f^s(x^s) | x \in B\} = \max\{\sum_{C\in\mathcal{C}_{m(s)}} x^s_C \cdot \sum_{j\in\mathcal{J}} g_j \frac{a_{C,j}}{p(j)} | x \in B\}.$$

Again, the above problem is maximized if we find the configuration $C' \in \mathcal{C}_{m(s)}$ that maximizes $\sum_{j\in\mathcal{J}} g_j \frac{a_{C,j}}{p(j)}$ and define $x^s_{C'} = 1$ for this configuration and $x^s_C = 0$ for all the other configurations in $\mathcal{C}_{m(s)}$.

How we can find the solution to the corresponding block problem depends solely on the definition of the set of configurations for both block problems. For Parallel Task Scheduling the only condition to the configurations is that the number of given machines is not exceeded. Hence to find the corresponding configuration it is sufficient to find an approximative solution to the following integer program.

$$\max \sum_{j\in\mathcal{J}} A_j \frac{g_j}{p(j)} \tag{3.8}$$

$$\sum_{j\in\mathcal{J}} A_j \cdot m(j) \leqslant m' \tag{3.9}$$

$$A_j \in \mathbb{N}_0 \tag{3.10}$$

In this integer program the variable A_j represent the multiplicity of job j in the calculated configuration while m' represents the bound for the machines of the configuration. The inequality (3.9) ensues that the chosen configuration does not exceed the given number of machines.

The IP for the above block-problem corresponds to the IP of the Unbounded Knapsack problem and hence can be solved approximately in

$\mathcal{O}(|\mathcal{J}| + (\log(1/\varepsilon))^3/\varepsilon^2)$ operations by the algorithm by Jansen and Kraft in [54].

Observation 3.8. If the linear program $LP_{conf}(\mathcal{J}, \mathcal{S})$ has a feasible solution, it holds for the corresponding block problem that $\lambda^* \geqslant 1$.

Consider a feasible solution x to the linear program $LP_{conf}(\mathcal{J}, \mathcal{S})$. Then obviously we have $x \in B$ for the corresponding block problem. Furthermore, since the second condition of the linear program holds, we have $\sum_{s\in\mathcal{S}} \sum_{C\in\mathcal{C}_{m(s)}} X_{C,s} a_{j,C} = p(j)$ and as a consequence $f_j(x) = 1$ for each $j \in \mathcal{J}$ and henceforth $\lambda^* \geqslant 1$.

Lemma 3.9. *An $(1+\varepsilon)$ approximate solution to $LP_{conf,\min}(\mathcal{J}, m)$ with at most $|\mathcal{J}| + 1$ non zero components can be found in at most $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}) \cdot (P(\mathcal{ABS}) + |\mathcal{J}|^{1.5356}))$ operations, were $P(\mathcal{ABS})$ denotes the time needed to solve the corresponding block problem.*

There is an algorithm that finds a solution to the relaxed version of the linear program $LP_{conf}(\mathcal{J}, \mathcal{S})$ with at most $|\mathcal{S}| + |\mathcal{J}|$ non zero components in at most $\mathcal{O}(|\mathcal{S}| \cdot |\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}) \cdot (P(\mathcal{ABS}) + (|\mathcal{J}| + |\mathcal{S}|)^{1.5356}))$ operations, or decides correctly that there exists no feasible solution.

Proof. To find the corresponding solutions we perform four steps: First we find an approximative solution to the corresponding Max-Min-Resource-Sharing problems, second we scale the solution such that each job is scheduled with a surplus but the relaxed configuration conditions are still fulfilled, third we remove some surplus processing time of jobs from the configurations such that we receive a solution to the linear program and in the final step we transform the solution to a basic solution.

Step 1: Solving the Max-Min-Resource-Sharing problem In the first step we transform the corresponding Configuration LP to a Max-Min-Resource-Sharing problem as seen above. This problem is then solved with precision $\rho := \varepsilon/(1+\varepsilon) \in \Theta(\varepsilon)$ using the algorithm by Grigoriadis et al. [41] in $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}) \cdot (P(\mathcal{ABS}))$ for the minimization version, i. e., $LP_{conf,\min}(\mathcal{J}, m)$, and in $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}) \cdot |\mathcal{S}| \cdot (P(\mathcal{ABS}))$ for $LP_{conf}(\mathcal{J}, \mathcal{S})$.

The algorithm by Grigoriadis et al. [41] needs a start solution where

each job occurs in at least one configuration. One possible choice[1] is the vector x where each configuration containing exactly one of the jobs gets a processing time of $1/|\mathcal{J}|$ or $p(s)/|\mathcal{J}_s|$ respectively, where $\mathcal{J}_s$ is the set of jobs width machine requirement at most $m(s)$. If there exists a job $j \in \mathcal{J}$ with $m(j) > m(s)$ for all $s \in \mathcal{S}$, there is no feasible solution to the corresponding configuration LP. This start solution has at most $|\mathcal{J}|$ or $|\mathcal{S}||\mathcal{J}|$ non zero components respectively.

Let $\hat{x}$ be the solution calculated by the algorithm by Grigoriadis et al. [41]. The algorithm adds at most $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}))$ non zero entries for $LP_{conf,\min}(\mathcal{J}, m)$ and at most $\mathcal{O}(|\mathcal{S}||\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}))$ for $LP_{conf}(\mathcal{J}, \mathcal{S})$ to the non zero components of the start solutions because the block-problem is called at most this number of times, and each call adds at most one new non zero component. Hence the number of non zero components in the vector $\hat{x}$ is bounded by $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}))$ or by $\mathcal{O}(|\mathcal{S}||\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}))$ respectively.

Step 2: Scaling of $\hat{x}$ In this step, we transform the solution for the Max-Min-Resource-Sharing problem $\hat{x}$ to a solution where all jobs are scheduled with a surplus, but the relaxed configuration conditions are still fulfilled.

The considered solution $\hat{x}$ implicates a value

$$\hat{\lambda} := \max\{\lambda | f_j(\hat{x}) \geqslant \lambda, \forall j \in \mathcal{J}\}, = \min\{f_j(\hat{x}) | j \in \mathcal{J}\},$$

where

$$f_j(x) = \sum_{C \in \mathcal{C}_m} X_C a_{j,C} / p(j)$$

or

$$f_j(x) = \sum_{s \in \mathcal{S}} \sum_{C \in \mathcal{C}_{m(s)}} X_{C,s} a_{j,C} / p(j)$$

respectively. The algorithm guarantees that $\hat{\lambda} \geqslant (1 - \rho)\lambda^*$. We define a scaled solution $\tilde{x} := \hat{x}/\hat{\lambda}$.

Consider the first configuration LP, i. e., $LP_{conf,\min}(\mathcal{J}, m)$. In the following, we will prove that the vector $\tilde{x}$ meets our requirements to some degree, but still needs some adjustment.

[1] The running time of the algorithm might be improved by a smarter choice for the start solution. This can be done for example by filling the configurations greedily instead of using just one job per configuration.

Claim 3.10. Let $x \in B$ with $f(x) \geqslant (1-\rho)\lambda^* \mathbb{1}_{|\mathcal{J}|}$ and $\hat{\lambda} := \min\{f_j(x) | j \in \mathcal{J}\}$. It holds that $f(x/\hat{\lambda}) \geqslant \mathbb{1}_{|\mathcal{J}|}$ and

$$\sum_{C \in \mathcal{C}_m} \frac{1}{\hat{\lambda}} x_C \leqslant \frac{1}{(1-\rho)} \mathrm{OPT}_{LP}.$$

Proof. The first part of the claim $f(x/\hat{\lambda}) \geqslant \mathbb{1}_{|\mathcal{J}|}$ is obvious since

$$f_j(x/\hat{\lambda}) = f_j(x)/\hat{\lambda} = f_j(x)/\min\{f_j(x) | j \in \mathcal{J}\} \geqslant 1$$

for all $j \in \mathcal{J}$. To prove the second part, we consider a more general definition of λ^* and B. Let $t \in \mathbb{R}_{\geqslant 0}$ be a target makespan and let

$$B_t := \{x \in \mathbb{R}_{\geqslant 0}^{|\mathcal{C}_m|} | \sum_{C \in \mathcal{C}_m} x_C = t\},$$

$$\lambda_t^* := \max\{\lambda | \exists x \in B_t \forall j \leqslant M : f_j(x) \geqslant \lambda\}.$$

Let $t' \in \mathbb{R}_{\geqslant 0}$. By making use of the linearity of f and $g : B \to \mathbb{R}, x \mapsto \sum_{C \in \mathcal{C}_m} x_C$ it is easy to see that $B_t = \frac{t}{t'} B_{t'}$ and $\lambda_t^* = \frac{t}{t'} \lambda_{t'}^*$. Using $t = 1$, $\lambda_1^* = \lambda^*$, $t' = \mathrm{OPT}_{LP}$, and $\lambda_{\mathrm{OPT}_{LP}}^* = 1$, this yields

$$\sum_{C \in \mathcal{C}_{I,\varepsilon}} \frac{1}{\hat{\lambda}} x_C = \frac{1}{\hat{\lambda}} \leqslant \frac{1}{(1-\rho)} \frac{1}{\lambda^*} \leqslant \frac{1}{1-\rho} \mathrm{OPT}(I, \varepsilon)$$

and concludes the proof of the claim. ◁

As a result we know that $\sum_{C \in \mathcal{C}_m} \tilde{x}_C \leqslant \frac{1}{(1-\rho)} \mathrm{OPT}_{LP} = (1+\varepsilon)\mathrm{OPT}_{LP}$ and hence $\tilde{x}$ is a good enough approximation to the solution of the linear program $LP_{conf,\min}(\mathcal{J}, m)$. On the other hand, for all the equations (3.2), we have $\sum_{C \in \mathcal{C}_m} \tilde{x}_C \cdot a_{j,C} \geqslant p(j)$. Therefore, we need to remove some processing time of of jobs from the configurations. This is done in the next step. In order to do so, we transform the linear program $LP_{conf,\min}(\mathcal{J}, m)$ to a linear program of the form $LP_{conf}(\mathcal{J}, \mathcal{S})$ by adding one segment s to $\mathcal{S}$ and define $p(s) := \sum_{C \in \mathcal{C}_m} \tilde{x}_C$, and $m(s) := m$.

Consider the second configuration LP, i. e., $LP_{conf}(\mathcal{J}, \mathcal{S})$. Since $\hat{\lambda} \geqslant (1-\rho)\lambda^*$ and using Observation 3.8, there is no solution to $LP_{conf}(\mathcal{J}, \mathcal{S})$ if $\hat{\lambda} < (1-\rho)$ and the algorithm terminates. On the other hand if $\hat{\lambda} \geqslant (1-\rho)$,

we have

$$\sum_{C \in \mathcal{C}_{m(s)}} \tilde{x}_{C,s} \leqslant \sum_{C \in \mathcal{C}_{m(s)}} \hat{x}_{C,s}/(1-\rho) = p(s)/(1-\rho) = (1+\varepsilon)p(s) \quad \forall s \in \mathcal{S},$$

using $\rho := \varepsilon/(1+\varepsilon)$ and

$$f_j(\tilde{x}) = \sum_{s \in \mathcal{S}} \sum_{C \in \mathcal{C}_{m(s)}} \tilde{x}_{C,s} a_{j,C}/p(j) = \sum_{s \in \mathcal{S}} \sum_{C \in \mathcal{C}_{m(s)}} \hat{x}_{C,s} a_{j,C}/(p(j)\hat{\lambda}) \geqslant 1.$$

Therefore, $\tilde{x}$ fulfills the relaxed conditions (3.7). However the equations (3.5) are not fulfilled since the left hand side might be larger than the right hand side.

Step 3: Removing surplus procesing time Let us assume we are given a vector $\tilde{x}$ that fulfills the conditions (3.7) for an LP $LP_{conf}(\mathcal{J}, \mathcal{S})$, but it holds that

$$\sum_{s \in \mathcal{S}} \sum_{C \in \mathcal{C}_{m(s)}} \tilde{x}_{C,s} a_{j,C} \geqslant p(j)$$

for all $j \in \mathcal{J}$. By adding at most $|\mathcal{J}|$ new configurations, we will modify $\tilde{x}$ to a vector $\check{x}$, where we have

$$\sum_{s \in \mathcal{S}} \sum_{C \in \mathcal{C}_{m(s)}} \tilde{x}_{C,s} a_{j,C} = p(j) \quad \forall j \in \mathcal{J}$$

and

$$\sum_{C \in \mathcal{C}_{m(s)}} \tilde{x}_{C,s} \leqslant (1+\varepsilon)p(s) \quad \forall s \in \mathcal{S}.$$

Consider each job $j \in \mathcal{J}$ and define the overhead

$$\omega(j) := \sum_{s \in \mathcal{S}} \sum_{C \in \mathcal{C}_{m(s)}} \tilde{x}_{C,s} a_{j,C} - p(j) \geqslant 0.$$

We go through the configurations containing j, i. e., the configurations C with $a_{j,C} \geqslant 0$. If $\tilde{x}_{C,s} a_{j,C} \leqslant \omega(j)$, we replace the configuration C in s by the configuration without this job and update the overhead value, i. e., we consider the configuration C^{-j} with $a_{j',C^{-j}} = a_{j',C}$ for all $j' \neq j \in \mathcal{J}$ as well as $a_{j,C^{-j}} = 0$ and overwrite $\tilde{x}_{C^{-j},s} := \tilde{x}_{C^{-j},s} + \tilde{x}_{C,s}$, $\omega(j) = \omega(j) - a_{j,C}\tilde{x}_{C,s}$, as well as $\tilde{x}_{C,s} = 0$. In this step, we do not introduce an additional non zero component since we replace one non zero component with another. On the other hand, if $\tilde{x}_{C,s} a_{j,C} < \omega(j)$, we reduce the processing time of C

in s by $\omega(j)/a_{j,C}$ and add $\omega(j)/a_{j,C}$ processing time to the configuration C^{-j} in s. This generates at most one new zero component, but now the new vector $\tilde{x}$ fulfills the equation (3.5) for the job j. We call the result of this iterative transformation $\check{x}$. See an overview of these transformation steps in Algorithm 1.

Algorithm 1: Transforming $\tilde{x}$

```
Data: x̃, C, S, J
Result: x̌
foreach j ∈ J do
    ω := Σ_{s∈S} Σ_{C∈C_{m(s)}} x̃_{C,s} a_{j,C} − p(j);
    foreach s ∈ S and C ∈ C_{m(s)} do
        if ω = 0 then break;
        find C^{−j};
        if a_{j,C} ⩾ 0 and x̃_{C,s} a_{j,C} ⩾ ω then
            x̃_{C^{−j},s} := x̃_{C^{−j},s} + x̃_{C,s};
            ω = ω − a_{j,C} x̃_{C,s};
            x̃_{C,s} = 0;
        else if a_{j,C} ⩾ 0 and x̃_{C,s} a_{j,C} < ω then
            p = ω/a_{j,C};
            x̃_{C^{−j},s} := x̃_{C^{−j},s} + p;
            x̃_{C,s} = x̃_{C,s} − p;
            break ;
    end
    return x̃
end
```

Step 4: generating a basic solution The last step of the algorithm is to generate a basic solution. Let $\check{x}$ be the generated solution to $LP_{conf}(\mathcal{J}, \mathcal{S})$. This solution has at most $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}) \cdot |\mathcal{S}|)$ non zero components, and is a solution to a problem of the form $Ax = b, x \geqslant 0$, for $A \in \mathbb{N}^{m \times n}$, $x \in \mathbb{R}^n_{\geqslant 0}$ and $b \in \mathbb{N}^m$. We remove all the non zero entries from x and the corresponding columns from A. Now we know that $m \in \mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}) \cdot |\mathcal{S}|)$ and $n = |\mathcal{J}| + |\mathcal{S}|$. There is an algorithm by Beling and Megiddo [9], which was improved by Ke, Zeng, Han, and Pan [75], which given a start solution

to the above described problem finds a basic solution in $\mathcal{O}(m^{1.5356}n)$. We use this algorithm to transform the generated solution to a basic solution in $\mathcal{O}((|\mathcal{J}| + |\mathcal{S}|)^{1.5356} \cdot (|\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}) \cdot |\mathcal{S}|))$.

Together with the steps done so far the total algorithm has a running time of at most $\mathcal{O}((|\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}) \cdot |\mathcal{S}|) \cdot (P(\mathcal{ABS}) + (|\mathcal{J}| + |\mathcal{S}|)^{1.5356}))$ for the second LP, i. e., $LP_{conf}(\mathcal{J}, \mathcal{S})$ and a running time of at most $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \varepsilon^{-2}) \cdot (P(\mathcal{ABS}) + |\mathcal{J}|^{1.5356}))$ for $LP_{conf,\min}(\mathcal{J}, m)$. To find the basic solution for $LP_{conf,\min}(\mathcal{J}, m)$, we had introduced one segment and hence the number of non zero components in the solution for this linear program is bounded by $|\mathcal{J}| + 1$.

□

Chapter 4

Hardness Results for Parallel Task Scheduling and Strip Packing

In this chapter, we study Parallel Task Scheduling $Pm|size_j|C_{max}$ for a constant number of machines. This problem is known to be strongly NP-complete for each $m \geqslant 5$, while it is solvable in pseudo-polynomial time for each $m \leqslant 3$. We give a positive answer to the long-standing open question whether this problem is strongly NP-complete for $m = 4$.

As a second result, we improve the lower bound of $\frac{12}{11}$ for approximating pseudo-polynomial Strip Packing to $\frac{5}{4}$. Since the best known approximation algorithm for this problem has a ratio of $\frac{4}{3} + \varepsilon$, this result narrows the gap between approximation ratio and inapproximability result. Both results are proved by a reduction from the strongly NP-complete problem 3-Partition.

An extended abstract of the results in this chapter has first been published in CSR [47] and a full version was accepted by Theory of Computing Systems [48].

4.1 Results

In the Parallel Task Scheduling (PTS) problem denoted as $P|size_j|C_{max}$ in the three-field-notation, a set of jobs $\mathcal{J}$ has to be scheduled on m machines minimizing the makespan C_{max}. Each job $j \in \mathcal{J}$ has a processing time $p(j) \in \mathbb{N}$ and requires $m(j) \in \mathbb{N}$ machines. A schedule S is given by two functions $\sigma : \mathcal{J} \to \mathbb{N}$ and $\rho : \mathcal{J} \to 2^{\{1,\dots,m\}}$. The function σ maps each

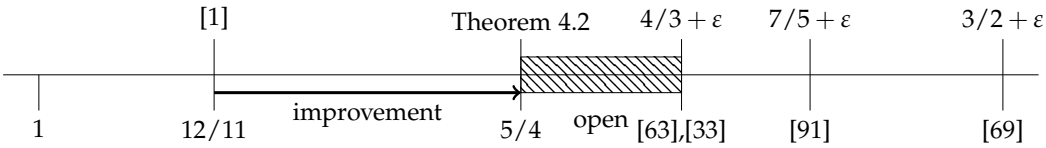


Figure 4.1. The upper and lower bounds for the best possible approximation for pseudo-polynomial Strip Packing achieved so far

job to a start point in the schedule, while ρ maps each job to the set of machines it is processed on. We say a machine i contains a job $j \in \mathcal{J}$ if $i \in \rho(j)$. A schedule is feasible if each machine processes at most one job at a time and each job is processed on the required number of machines (i.e. $|\rho(j)| = q(j)$). The objective is to find a feasible schedule S minimizing the makespan $C_{\max} := \max_{j \in \mathcal{J}}(\sigma(j) + p(j))$.

In 1989, Du and Leung [27] proved the Parallel Task Scheduling problem $P|size_j|C_{\max}$ to be strongly NP-complete for all $m \geqslant 5$, while it is solvable by a pseudo-polynomial time algorithm for all $m \leqslant 3$. In this paper, we address the case of $m = 4$, which has been open since and prove:

Theorem 4.1. *Parallel Task Scheduling on* 4 *machines, i.e.,* $P4|size_j|C_{\max}$*, is strongly* NP*-complete.*

Building on this result, we can prove a lower bound for the absolute approximation ratio of pseudo-polynomial time algorithms for Strip Packing. Remember, in the Strip Packing problem a set of rectangular items $\mathcal{I}$ has to be placed into a strip of width $W \in \mathbb{N}$ and infinite height. Each item $i \in \mathcal{I}$ has a width $w(i) \in \mathbb{N}_{\leqslant W}$ and a height $h(i) \in \mathbb{N}$. A *packing* of the items $\mathcal{I}$ into the strip is a function $\rho : \mathcal{I} \to \mathbb{N}_0 \times \mathbb{N}_0$, which places the items axis-parallel into the strip by assigning the left bottom corner of an item to a position in the strip such that for each item $i \in \mathcal{I}$ with $\rho(i) = (x_i, y_i)$ we have $x_i + w(i) \leqslant W$. A packing is *feasible* if no two items overlap. The height of a packing is defined as $H := \max_{i \in \mathcal{I}} y_i + h(i)$. The objective is to find a feasible packing of the items $\mathcal{I}$ into the strip that minimizes the packing height.

Lately, pseudo-polynomial time algorithms for Strip Packing where the width of the strip is allowed to appear polynomial in the running time of the algorithm, while it appears only logarithmic in the input size of

the instance, gained high interest. In a series of papers [69, 91, 33, 63], the best approximation ratio was improved to $\frac{4}{3} + \varepsilon$. On the other hand, it is not possible to find an algorithm with approximation ratio better than $\frac{12}{11}$, except P = NP [1]. In this paper, we improve this lower bound to $\frac{5}{4}$, which almost closes the gap between lower bound and best algorithm.

Theorem 4.2. *It is* NP*-Hard to find a pseudo-polynomial time approximation algorithm for Strip Packing with an absolute approximation ratio strictly better than* $\frac{5}{4}$.

This inapproximability result for Strip Packing transfers to problems closely related to strip packing as Strip Packing With Rotations and Contiguous Moldable Task Scheduling.

For a more detailed overview on the work related to these two problems we refer to Section 2.1 and Section 2.2.

Overview of this Chapter

In Section 4.2, we will prove Theorem 4.1 by a reduction from the strongly NP-complete problem 3-Partition. First, we describe the jobs to construct for this reduction. Afterward, we prove: if the 3-Partition instance is a Yes-instance, then there is a schedule with a specific makespan, and if there is a schedule with this specific makespan, then the 3-Partition instance has to be a Yes-instance. While the first can be seen directly, the proof of the second is more involved. Proving the second claim, we first show that it can be w.l.o.g. supposed that each machine contains a certain set of jobs. In the next step, we prove some implications on the order in which the jobs appear on the machines which finally leads to the conclusion that the 3-Partition instance has to be a Yes-instance. In Section 4.3 we discuss the implications for the inapproximability of pseudo-polynomial Strip Packing.

4.2 Hardness of Parallel Task Scheduling

In this Section, we prove Theorem 4.1 by a reduction from the 3-Partition problem. In this problem, we are given a list $\mathcal{I} = (\iota_1, \ldots, \iota_{3z})$ of $3z$ positive

integers with $\sum_{i=1}^{3z} \iota_i = zD$ and $D/4 < \iota_i < D/2$ for each $1 \leqslant i \leqslant 3z$. The problem is to decide whether there exists a partition of the set $I = \{1, \ldots, 3z\}$ into sets $I_1, \ldots, I_z$ such that $\sum_{i \in I_j} \iota_i = D$ for each $1 \leqslant j \leqslant z$. This problem is strongly NP-complete see [36] problem [SP15]. Hence, it cannot be solved in pseudo-polynomial time, unless P = NP.

Before we start constructing the reduction, we introduce some notations. Let $j \in \mathcal{J}$ and $\mathcal{J}' \subseteq \mathcal{J}$. We define the work of j as $w(j) := p(j) \cdot p(j)$ and the total work of $\mathcal{J}'$ as $w(\mathcal{J}') := \sum_{j \in \mathcal{J}'} w(j)$. For a given schedule $S = (\sigma, \rho)$, we denote by $n_j(\mathcal{J}')$ the number of jobs from the set $\mathcal{J}'$ that are finished before the start of the job j, i.e., $n_j(\mathcal{J}') = |\{i \in \mathcal{J}' : \sigma(i) + p(i) \leqslant \sigma(j)\}|$. Furthermore, we will use a notation defined in [27] for swapping a part of the content of two machines; let $j \in \mathcal{J}$ be a job that is processed by at least two machines, M and M', with start point $\sigma(j)$. We can swap the content of the machines M and M' after time $\sigma(j)$ without violating any scheduling constraint. We define this swapping operation as $\text{SWAP}(\sigma(j), M, M')$.

The main idea of our reduction is to construct a set of *structure jobs*. These structure jobs have the property that each possible way to schedule them with the optimal makespan leaves z gaps, each with processing time D, i.e., it happens exactly at z distinct times that a machine is idle, and the duration of each idle time is exactly D, see Figure 4.2 at the hatched areas. As a consequence, *partition jobs*, which have processing times equal to the 3-Partition numbers, can only be scheduled with the desired makespan if the 3-Partition instance is a Yes-instance.

Construction.

In this section, we will construct a scheduling instance for $P4|size_j|C_{\max}$ from a given 3-Partition instance. In the following two paragraphs, we will give an intuition which jobs we introduce why with which processing time. An overview of the introduced jobs and their processing times can be found in Table 4.1 and the fourth paragraph of this section.

Given a 3-Partition instance, we construct ten disjoint sets of jobs A, B, a, b, c, α, β, γ, δ, and λ, which will be forced to be scheduled as in Figure 4.2 by choosing suitable processing times. In the first step, we add a unique token to the processing time of each set of jobs to be processed simultaneously to ensure that these jobs have to be processed at the same

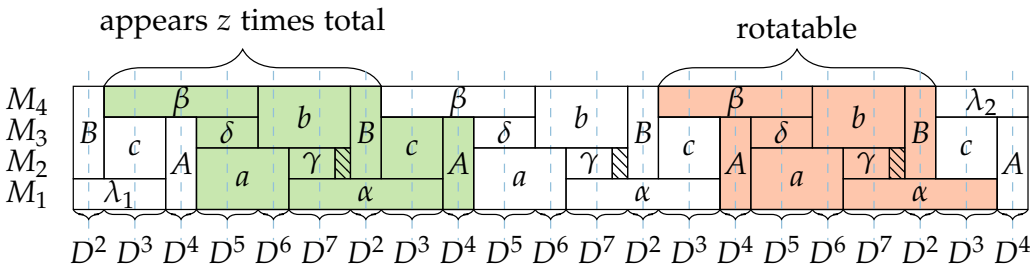


Figure 4.2. Packing of structure jobs with gaps (hatched area) for 3-Partition items. The items in the green area (left) are repeated z times. With the displayed choice of processing times, the items in the red area (right) can be rotated by 180 degrees such that α is scheduled on M_4 after the job in B and β is scheduled on M_1 before the job in A.

time in every schedule. As this token, we choose D^x, where $x \in \{2, \ldots, 7\}$ and D is the required sum of the items in each partition set. For example for jobs in B we define a processing time of D^2, while we define the processing time of each job in α such that it contains $D^7 + D^2 + D^3$, see Figure 4.2.

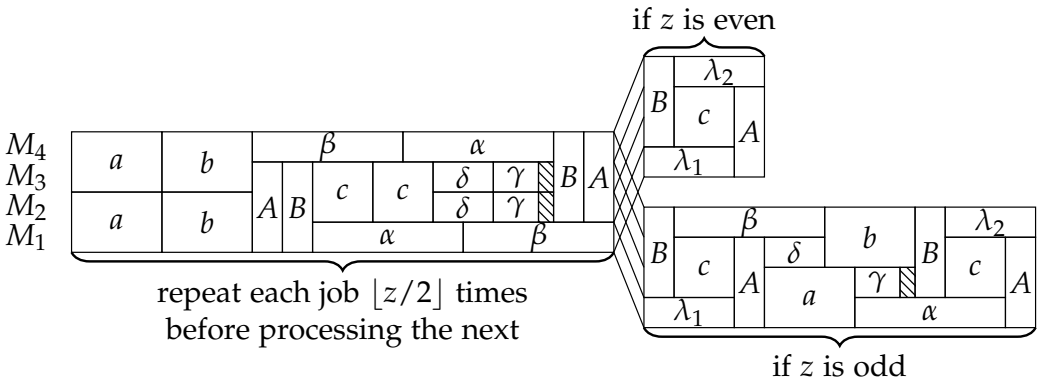


Figure 4.3. This figure presents a reordering possibility that has to be prohibited, because it fuses the areas for 3-Partition items into two areas, one area on M_2 and one area on M_3 if z is even, and into three areas if z is odd.

Unfortunately, the tokens D^2 to D^7 are not enough to ensure that the schedule in Figure 4.2 is the only possible one. Consider the jobs contained in the red area (right) in Figure 4.2. With the choice of processing times as

shown in the figure, it is possible to rotate the red area by 180 degrees such that α is scheduled on M_4 and β is scheduled on M_1. After rotating every second of these set of jobs, it is possible to reorder the jobs, and fusing the areas for the 3-Partition items into two or three areas, see Figure 4.3. To prohibit this possibility to rotate, we introduce one further token D^8. This token is added to the processing time of some jobs such that the combined processing time of the jobs in the red area on M_1 differs from the one on M_4. To ensure this, we have to give up the property that in each of the sets $A, B, a, b, c, \alpha, \beta, \gamma, \delta$ all jobs have the same processing time. More precisely, each job in the sets c, δ, and γ receives a unique processing time.

In the following, we describe the jobs constructed for the reduction. We introduce two sets A and B of 3-processor jobs, three sets a, b and c of 2-processor jobs, and five sets α, β, γ, δ, and λ of 1-processor jobs. The description of the jobs inside these sets and their processing times can be found in Table 4.1. We call these jobs *structure jobs*. Additionally, we generate for each $i \in \{1, \ldots, 3z\}$ one 1-processor job, called *partition job*, with processing time ι_i and define P as the set containing all partition jobs. Last, we define $W := (z+1)(D^2 + D^3 + D^4) + z(D^5 + D^6 + D^7) + z(7z+1)D^8$. Note that the total work of the introduced jobs adds up to $4W$, i.e., a schedule without idle times has makespan W while each schedule containing idle times has a makespan, which is strictly larger than W.

Table 4.1. Overview of the generated jobs

$m(j) = 3,\ p(j) = D^4 =: p_A$	if $j \in A$	$:= \{A_0, \ldots, A_z\}$
$m(j) = 3,\ p(j) = D^2 =: p_B$	if $j \in B$	$:= \{B_0, \ldots, B_z\}$
$m(j) = 2,\ p(j) = D^5 + D^6 + 3zD^8 =: p_a$	if $j \in a$	$:= \{a_1, \ldots, a_z\}$
$m(j) = 2,\ p(j) = D^6 + D^7 + 3zD^8 =: p_b$	if $j \in b$	$:= \{b_1, \ldots, b_z\}$
$m(j) = 2,\ p(j) = D^3 + (z+i)D^8$	if $j = c_i \in c$	$:= \{c_0, \ldots, c_z\}$
$m(j) = 1,\ p(j) = D^2 + D^3 + D^7 + 4zD^8 =: p_\alpha$	if $j \in \alpha$	$:= \{\alpha_1, \ldots, \alpha_z\}$
$m(j) = 1,\ p(j) = D^3 + D^4 + D^5 + (4z-1)D^8 =: p_\beta$	if $j \in \beta$	$:= \{\beta_1, \ldots, \beta_z\}$
$m(j) = 1,\ p(j) = D^7 + (3z-i)D^8 - D$	if $j = \gamma_i \in \gamma$	$:= \{\gamma_1, \ldots, \gamma_z\}$
$m(j) = 1,\ p(j) = D^5 + (3z-i)D^8$	if $j = \delta_i \in \delta$	$:= \{\delta_1, \ldots, \delta_z\}$
$m(j) = 1,\ p(j) = D^2 + D^3 + zD^8$	if $j = \lambda_1$	
$m(j) = 1,\ p(j) = D^3 + D^4 + 2zD^8$	if $j = \lambda_2$	
$m(j) = 1,\ p(j) = \iota_j$	if $j \in P$	$:= \{p_1, \ldots, p_{3z}\}$

Given a set $J \subseteq A \cup B \cup a \cup b \cup c \cup \alpha \cup \beta \cup \delta \cup \lambda$ of the jobs constructed this way, their total processing time $p(J)$ has the form $\sum_{i=2}^{7} x_i D^i$, with

$x_i \in \mathbb{N}$ for $i = 2, \ldots, 7$. For each occurring x_i, we want the tokens D^i to be unique in the way that $x_i D^i < D^{i+1}$ for each possible occurring sum of processing times of structure jobs and each $i = 2, \ldots, 7$. Let $k_{\max}$ be the largest occurring coefficient in the sum of processing times of any given subset of the generated structure jobs, i.e., $k_{\max} \leqslant 4z(7z+1)$. If $D \leqslant k_{\max}$, we scale each number in the 3-Partition instance with $k_{\max}$ before constructing the jobs. As a result in the scaled instance it holds that $k_{\max} D^i < D^{i+1}$. Since $k_{\max}$ depends polynomially on z, the input size of the scaled instance will still depend polynomially on the input size of the original instance. In the following, let us assume that $D > k_{\max}$ in the given 3-Partition instance. Note that in a schedule without idle times a machine cannot contain a set of jobs with processing times that add up to a value where one of the coefficients is larger than the corresponding one in W.

In the following two subsections, we will prove that there is a schedule with makespan W if and only if the 3-Partition instance is a Yes-instance.

Partition to Schedule.

Let $\mathcal{I}$ be a Yes-instance of 3-Partition with partition $I_1, \ldots, I_z$. One can easily verify that the *structure jobs* can be scheduled as shown in Figure 4.4. After each job γ_j, for each $1 \leqslant j \leqslant z$, we have a gap with processing time D. We schedule the *partition jobs* with indices out of I_j directly after γ_j. Their processing times add up to D, and therefore they fit into the gap. The resulting schedule has a makespan of W.

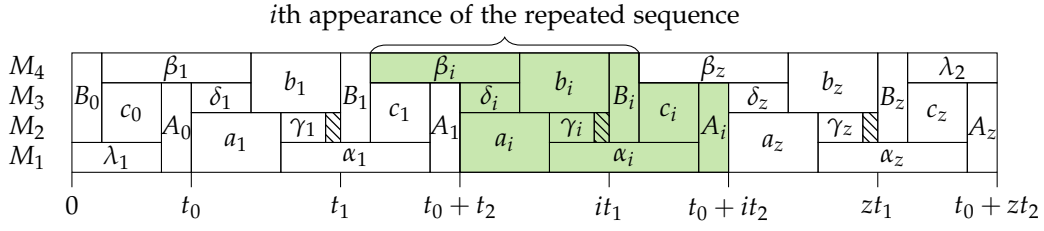


Figure 4.4. An optimal schedule, for a Yes-instance, where $t_0 := \sum_{k=2}^{4} D^k + zD^8$, $t_1 := \sum_{k=2}^{7} D^k + (7z-1)D^8$, $t_2 := \sum_{k=2}^{7} D^k + 7zD^8$, and $t_0 + zt_2 = W$.

Schedule to Partition.

Let a schedule $S = (\sigma, \rho)$ with makespan W be given. We will now step by step describe why $\mathcal{I}$ has to be a Yes-instance of 3-Partition. In the first step, we will show that we can transform the schedule such that each machine contains a certain set of jobs.

Lemma 4.3. *We can transform the schedule S into a schedule such that M_1 contains the jobs $A \cup a \cup \alpha \cup \lambda_1$, M_2 contains the jobs $A \cup B \cup c \cup \check{a} \cup \check{b} \cup \check{\gamma} \cup \check{\delta}$, M_3 contains the jobs $A \cup B \cup c \cup \hat{a} \cup \hat{b} \cup \hat{\gamma} \cup \hat{\delta}$ and M_4 contains the jobs $B \cup b \cup \beta \cup \lambda_2$, where $a = \check{a} \dot{\cup} \hat{a}$, $b = \check{b} \dot{\cup} \hat{b}$, $\gamma = \check{\gamma} \dot{\cup} \hat{\gamma}$, and $\delta = \check{\delta} \dot{\cup} \hat{\delta}$. Furthermore, if the jobs are scheduled in this way, it holds that $|\check{a}| = |\check{\gamma}|$ and $|\check{b}| = |\check{\delta}|$.*

Proof. First, we will prove that the content of the four machines can be swapped (without enlarging the makespan) such that M_2 and M_3 each contain all the jobs in $A \cup B$. We will show this claim inductively. For the induction basis, consider the job in $A \cup B$ with the smallest starting point in this set. We can swap the complete content of the machines such that M_2 and M_3 contain this job. For the induction step, let us assume that the first i jobs from the set $A \cup B$ are scheduled on the machines M_2 and M_3. Consider the $(i+1)$st job. This job is either already scheduled on the machines M_2 and M_3, and we do nothing, or there is one machine $M \in \{M_2, M_3\}$, which does not contain this job. Let us assume the latter. Let $M' \in \{M_1, M_4\}$ be the third machine containing the i-th job in $A \cup B$. We transform the schedule such that M_2 and M_3 contain the $(i+1)$-th job by performing a swapping operation $\text{SWAP}(\sigma(x_i), M, M')$. After this swap M, and hence both machines M_2 and M_3, will contain the $(i+1)$st job, which concludes the proof that the content of the machines can be swapped such that M_2 and M_3 each contain all the jobs in $A \cup B$.

In the next step, we will determine the set of jobs contained by the machines M_1 and M_4 using the token D^8. Besides the jobs in $A \cup B$, M_2 and M_3 contain jobs with total processing time of $(z+1)D^3 + zD^5 + zD^6 + zD^7 + z(7z+1)D^8$. Hence, M_2 and M_3 cannot contain jobs in $\alpha \cup \beta \cup \lambda$, since their processing times contain D^2 or D^4. Therefore, each job in $\alpha \cup \beta \cup \lambda$ has to be processed either on M_1 or on M_4. Furthermore, each job in $A \cup B$ has to be processed on one of the machines M_1 or M_4

additional to the machines M_2 and M_3 since each of these jobs needs three machines to be scheduled. In addition to the jobs jobs in $A \cup B \cup \alpha \cup \beta \cup \lambda$, M_1 and M_4 together contain further jobs with a total processing time of $zD^5 + 2zD^6 + zD^7 + 6z^2D^8$. Exclusively jobs from the set $a \cup b$ have a processing time containing D^6. Therefore, each machine processes z of them. Hence corresponding to D^8, a total processing time of $3z^2D^8$ is used by jobs in the set $a \cup b$ on each machine. This leaves a processing time of $(4z^2 + z)D^8$ for the jobs in $\alpha \cup \beta \cup \lambda$ on M_1 and M_4. All the $2(z+1)$ jobs in $\alpha \cup \beta \cup \lambda$ contain D^3 in their processing time. Therefore, each machine M_1 and M_4 processes exactly $z+1$ of them. We will swap the content of M_1 and M_4 so that λ_1 is scheduled on M_1. As a consequence, M_1 processes z jobs from the set $\alpha \cup \beta \cup \{\lambda_2\}$, with processing times that sum up to $4z^2D^8$ in the D^8 component. The jobs in α have with $4zD^8$ the largest amount of D^8 in their processing time. Therefore, M_1 has to process all of them since $z \cdot 4zD^8 = 4z^2D^8$, while M_4 contains the jobs in $\beta \cup \{\lambda_2\}$. Since we have $p(\alpha \cup \{\lambda_1\}) = (z+1)D^2 + (z+1)D^3 + zD^7 + z(4z+1)D^8$, jobs from the set $A \cup B \cup a \cup b$ that have a total processing time of $(z+1)D^4 + zD^5 + zD^6 + 3z^2D^8$ have to be scheduled on M_1. In this set, the jobs in A are the only jobs with processing times containing D^4, while the jobs in a are the only jobs with a processing time containing D^5. As a consequence, M_1 processes the jobs $A \cup a \cup \alpha \cup \{\lambda_1\}$. Analogously we can deduce that M_4 processes the jobs $B \cup b \cup \beta \cup \{\lambda_2\}$.

In the last step, we will determine which jobs are scheduled on M_2 and M_3. As shown before, each of them contains the jobs $A \cup B$. Furthermore, since no job in c is scheduled on M_1 or M_4, and they require two machines to be processed, machines M_2 and M_3 both contain the set c. Additionally, each job in $\gamma \cup \delta$ has to be scheduled on M_2 or M_3 since they are not scheduled on M_1 or M_4. Each job in $a \cup b$ occupies one of the machines M_1 and M_4. The second machine they occupy is either M_2 or M_3. Let $\breve{a} \subseteq a$ be the set of jobs that is scheduled on M_2 and $\hat{a} \subseteq a$ be the set that is scheduled on M_3. Clearly $a = \breve{a} \dot{\cup} \hat{a}$. We define the sets $\hat{b}, \breve{b}, \hat{\delta}, \breve{\delta}, \hat{\gamma}$, and $\breve{\gamma}$ analogously. By this definition, M_2 contains the jobs $A \cup B \cup \breve{a} \cup \breve{b} \cup \breve{\delta} \cup \breve{\gamma} \cup c$ and M_3 contains the jobs $A \cup B \cup \hat{a} \cup \hat{b} \cup \hat{\delta} \cup \hat{\gamma} \cup c$.

We still have to prove that $|\breve{a}| = |\breve{\gamma}|$ and $|\breve{b}| = |\breve{\delta}|$. First, we notice that $|\breve{a}| + |\breve{b}| = z$ since these jobs are the only jobs with a processing time containing D^6. So besides the jobs in $A \cup B \cup c \cup \breve{a} \cup \breve{b}$, M_2 contains jobs

with total processing time of $(z - |\check{a}|)D^5 + (z - |\check{b}|)D^7 + \sum_{i=1}^{z}(3z - i)D^8 = |\check{b}|D^5 + |\check{a}|D^7 + \sum_{i=1}^{z}(3z - i)D^8$. Since the jobs in δ are the only jobs in $\delta \cup \gamma$ having a processing time containing D^5, we have $|\check{\delta}| = |\check{b}|$ and analogously $|\check{\gamma}| = |\check{a}|$. □

In the next steps, we will prove that it is possible to transform the order in which the jobs appear on the machines to the one in Figure 4.4. Notice that, since there is no idle time in the schedule, each start point of a job i is given by the sum of processing times of the jobs on the same machine scheduled before i. So the start position $\sigma(i)$ of a job i has the form

$$\sigma(i) = x_0 + x_2D^2 + x_3D^3 + x_4D^4 + x_5D^5 + x_6D^6 + x_7D^7 + x_8D^8$$

for $-zD \leqslant x_0 \leqslant zD < D^2$ and $0 \leqslant x_j \leqslant 4z(7z + 1) \leqslant D$ for each $2 \leqslant j \leqslant 8$. Note that $-zD \leqslant x_0$ since the processing time of the jobs in γ is given by $D^7 + (3z - i)D^8 - D$ and there are at most z of them while $x_0 \leqslant zD$ since the total sum of processing times of partition jobs is at most zD. This equation for $\sigma(i)$ allows us to analyze how many jobs of which type are scheduled before a job i on the machine that processes i. For example, let us look at the coefficient x_4. This value is just influenced by jobs with processing times containing D^4. The only jobs with these processing times are the jobs in the set $A \cup \beta \cup \{\lambda_2\}$. The jobs in $\beta \cup \{\lambda_2\}$ are just processed on M_4, while the jobs in A each are processed on the three machines M_1, M_2, and M_3. Therefore, we know that at the starting point $\sigma(i)$ of a job i scheduled on machines M_1, M_2 or M_3 we have that $x_4 = n_i(A)$. Furthermore, if i is scheduled on M_4 we know that $x_4 = n_i(\beta) + n_i(\{\lambda_2\})$. In Table 4.2, we present which sets influences which coefficients in which way when job i is started on the corresponding machine.

Let us consider the start point $\sigma(i)$ of a job i that uses more than one machine. We know that $\sigma(i)$ is the same on all the used machines and therefore the coefficients are the same as well. In the following, we will study for each of the sets A, B, a, b, c what we can conclude for the starting times of these jobs. For each of the sets, we will present an equation, which holds at the start of each item in this set. These equations give us a strong set of tools for our further arguing.

Lemma 4.4. *Let be $A' \in A$, $B' \in B$, $a' \in a$, $b' \in b$, and $c' \in c$. It holds that*

Table 4.2. Overview of the values of the coefficients at the start point of a job i, if i is scheduled on machine M_j.

	M_1	M_2	M_3	M_4
x_2	$n_i(\alpha) + n_i(\{\lambda_1\})$	$n_i(B)$	$n_i(B)$	$n_i(B)$
x_3	$n_i(\alpha) + n_i(\{\lambda_1\})$	$n_i(c)$	$n_i(c)$	$n_i(\beta) + n_i(\{\lambda_2\})$
x_4	$n_i(A)$	$n_i(A)$	$n_i(A)$	$n_i(\beta) + n_i(\{\lambda_2\})$
x_5	$n_i(a)$	$n_i(\breve{a}) + n_i(\breve{\delta})$	$n_i(\hat{a}) + n_i(\hat{\delta})$	$n_i(\beta)$
x_6	$n_i(a)$	$n_i(\breve{a}) + n_i(\breve{b})$	$n_i(\hat{a}) + n_i(\hat{b})$	$n_i(b)$
x_7	$n_i(\alpha)$	$n_i(\breve{b}) + n_i(\breve{\gamma})$	$n_i(\hat{b}) + n_i(\hat{\gamma})$	$n_i(b)$

$$\begin{aligned} n_{A'}(c) - n_{A'}(\{\lambda_1\}) &= n_{A'}(B) - n_{A'}(\{\lambda_1\}) = n_{A'}(\alpha) \\ &= n_{A'}(b) = n_{A'}(a), \end{aligned} \tag{4.1}$$

$$\begin{aligned} n_{B'}(c) - n_{B'}(\{\lambda_2\}) &= n_{B'}(A) - n_{B'}(\{\lambda_2\}) = n_{B'}(\beta) \\ &= n_{B'}(a) = n_{B'}(b), \end{aligned} \tag{4.2}$$

$$n_{a'}(B) = n_{a'}(\alpha) + n_{a'}(\{\lambda_1\})) = n_{a'}(c), \tag{4.3}$$

$$n_{b'}(A) = n_{b'}(\beta) + n_{b'}(\{\lambda_2\}) = n_{b'}(c), \tag{4.4}$$

and

$$n_{c'}(b) = n_{c'}(a). \tag{4.5}$$

Proof. We will prove these equations using the conditions for the coefficients from Table 4.2. We write $=_{x_i}$ when the coefficient x_i is the reason why the equality is true.

To prove equation (4.1), we will consider the start points of the jobs in A. Each job $A' \in A$ is scheduled on machines M_1, M_2 and M_3. As a consequence, the coefficients on all these tree machines have to be the same, when the job A' starts. Therefore, we know that at $\sigma(A')$ we have

$$n_{A'}(B) =_{x_2} n_{A'}(\alpha) + n_{A'}(\{\lambda_1\}) =_{x_3} n_{A'}(c). \tag{i}$$

Furthermore, we know that

$$n_{A'}(a) =_{x_6} n_{A'}(\breve{a}) + n_{A'}(\breve{b}) =_{x_6} n_{A'}(\hat{a}) + n_{A'}(\hat{b}).$$

Since $n_{A'}(a) = n_{A'}(\breve{a}) + n_{A'}(\hat{a})$ and $n_{A'}(b) = n_{A'}(\breve{b}) + n_{A'}(\hat{b})$, because $a = \breve{a} \dot{\cup} \hat{a}$ and $b = \breve{b} \dot{\cup} \hat{b}$, we can deduce that $n_{A'}(\hat{a}) = n_{A'}(\breve{b})$ and $n_{A'}(\breve{a}) =$

$n_{A'}(\hat{b})$ and therefore

$$n_{A'}(a) = n_{A'}(b). \tag{ii}$$

Additionally, we know that

$$n_{A'}(\alpha) =_{x_7} n_{A'}(\check{b}) + n_{A'}(\check{\gamma}) =_{x_7} n_{A'}(\hat{b}) + n_{A'}(\hat{\gamma}).$$

Thanks to this equality, we can show that $n_{A'}(\alpha) = n_{A'}(b)$: First, we show $n_{A'}(\alpha) \geqslant n_{A'}(b)$. Let $b' \in b$ be the last job in b scheduled before A' if there is any. Let us w.l.o.g. assume that $b' \in \hat{b}$. It holds that

$$\begin{aligned} n_{A'}(b) &\overset{\sigma(b')<\sigma(A')}{=} n_{b'}(b) + 1 =_{x_7} n_{b'}(\hat{b}) + n_{b'}(\hat{\gamma}) + 1 \\ &\overset{\sigma(b')<\sigma(A')}{\leqslant} n_{A'}(\hat{b}) + n_{A'}(\hat{\gamma}) =_{x_7} n_{A'}(\alpha). \end{aligned}$$

If there is no such b', we have $n_{A'}(b) = 0 \leqslant n_{A'}(\alpha)$. Next, we show $n_{A'}(\alpha) \leqslant n_{A'}(b)$. Let $b'' \in b$ be the first job in b scheduled after A if there is any. Let us w.l.o.g. assume that $b'' \in \check{b}$. It holds that

$$\begin{aligned} n_{A'}(b) &\overset{\sigma(A')<\sigma(b'')}{=} n_{b''}(b) =_{x_7} n_{b''}(\check{b}) + n_{b''}(\check{\gamma}) \\ &\overset{\sigma(A')<\sigma(b'')}{\geqslant} n_{A'}(\check{b}) + n_{A'}(\check{\gamma}) =_{x_7} n_{A'}(\alpha). \end{aligned}$$

If there is no such b'', we have $n_{A'}(b) = z \geqslant n_{A'}(\alpha)$. As a consequence, we have

$$n_{A'}(\alpha) = n_{A'}(b). \tag{iii}$$

In summary, we can deduce that

$$\begin{aligned} n_{A'}(c) - n_{A'}(\{\lambda_1\}) &=_{(\mathrm{i})} n_{A'}(B) - n_{A'}(\{\lambda_1\}) =_{(\mathrm{i})} n_{A'}(\alpha) \\ &=_{(\mathrm{iii})} n_{A'}(b) =_{(\mathrm{ii})} n_{A'}(a), \end{aligned}$$

which concludes the proof of equation (4.1). Since the Table 4.2 is symmetrically, we can deduce correctness of equation (4.2) analogously.

Next we prove the equations (4.3) and (4.4). Each item $a' \in a$ is scheduled on machine M_1 and on one of the machines M_2 or M_3. For both possibilities $a \in \hat{a}$ or $a \in \check{a}$, we can deduce equation (4.3) directly from the Table 4.2:

$$n_{a'}(B) =_{x_2} n_{a'}(\alpha) + n_{a'}(\{\lambda_1\}) =_{x_3} n_{a'}(c).$$

Analogously, we deduce equation (4.4):

$$n_{b'}(A) =_{x_4} n_{b'}(\beta) + n_{b'}(\{\lambda_2\}) =_{x_3} n_{b'}(c).$$

Last, we prove equation (4.5). Each item $c' \in c$ is scheduled on M_2 and M_3. Let $a' \in a$ be the job with the smallest $\sigma(a') \geqslant \sigma(c')$. Let us w.l.o.g. assume that $a' \in \hat{a}$. It holds that

$$\begin{aligned} n_{c'}(\check{a}) + n_{c'}(\check{b}) &=_{x_6} n_{c'}(\hat{a}) + n_{c'}(\hat{b}) \leqslant n_{a'}(\hat{a}) + n_{a'}(\hat{b}) =_{x_6} n_{a'}(a) \\ &= n_{a'}(\hat{a}) + n_{a'}(\check{a}) = n_{c'}(\hat{a}) + n_{c'}(\check{a}). \end{aligned}$$

As a consequence, we have $n_{c'}(\check{b}) \leqslant n_{c'}(\hat{a})$ and $n_{c'}(\hat{b}) \leqslant n_{c'}(\check{a})$. Analogously, let $b' \in b$ be the job with the smallest $\sigma(b') \geqslant \sigma(c')$. Let us w.l.o.g. assume that $b' \in \check{b}$. It holds that

$$\begin{aligned} n_{c'}(\hat{a}) + n_{c'}(\hat{b}) &=_{x_6} n_{c'}(\check{a}) + n_{c'}(\check{b}) \leqslant n_{b'}(\check{a}) + n_{b'}(\check{b}) =_{x_6} n_{b'}(b) \\ &= n_{b'}(\hat{b}) + n_{b'}(\check{b}) = n_{c'}(\hat{b}) + n_{c'}(\check{b}). \end{aligned}$$

Therefore, $n_{c'}(\check{a}) \leqslant n_{c'}(\hat{b})$ and $n_{c'}(\hat{a}) \leqslant n_{c'}(\check{b})$. As a consequence from both equations, we have $n_{c'}(\check{a}) = n_{c'}(\hat{b})$ and $n_{c'}(\hat{a}) = n_{c'}(\check{b})$. Together with $n_{c'}(\check{a}) + n_{c'}(\check{b}) =_{x_6} n_{c'}(\hat{a}) + n_{c'}(\hat{b})$ equation (4.5), i.e., $n_{c'}(b) = n_{c'}(a)$, is a direct consequence. □

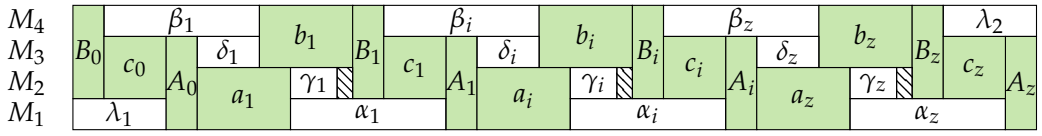


Figure 4.5. The processing order of the jobs in the sets A, B, a, b, and c.

These equations give us the tools to analyze the given schedule with makespan W. To this point we have proved that we can assume that the machines M_1 to M_4 contain the correct sets of jobs. Consider the jobs from the sets A, B, a, b, and c. Note that if one job from the set $A \cup B \cup a \cup b \cup c$ is started, no (other) job from the set $A \cup B \cup c$ can be processed at the same time, since these jobs will always share at least one machine when processed. The next step is to prove that the jobs in these sets appear in the correct order, namely $B_0, c_0, A_0, \binom{b_1}{a_1}, B_1, c_1, A_1, \ldots$ see Figure 4.5. We will prove this claim in two steps. First, we will show that in this schedule

the first and last jobs have to be elements from the set $A \cup B$ (see Lemma 4.5). Afterward, we will prove that between two successive jobs from the set A there appears exactly one jobs from each set B, a, b, and c, and prove an analogue statement for two successive jobs from the set B (see Lemma 4.6).

With the knowledge gathered in the proofs of Lemma 4.5 and Lemma 4.6, we can prove that the given schedule can be transformed such that all jobs are scheduled contiguously, i.e., on an interval of machines, and that $\mathcal{I}$ has to be a Yes-instance of 3-Partition (see Lemma 4.11). In the following, we write $=_{(l)}$, when the equation (l) is the reason why the equality is true.

Lemma 4.5. *One job $i \in A \cup B$ is the first job which is processed, i.e., $\sigma(i) = 0$. Furthermore, one job from the set $j \in A \cup B$ is the last job to be processed in the schedule, i.e., $\sigma(i) + p(j) = W$.*

Proof. Let $i := \arg\min_{i \in A \cup B} \sigma(i)$ be the job with the smallest start point in $A \cup B$, (i.e., $n_i(A) = 0 = n_i(B)$). If $i \in A$, it holds that

$$0 = n_i(B) =_{(4.1)} n_i(\alpha) + n_i(\{\lambda_1\}) =_{(4.1)} n_i(a) + n_i(\{\lambda_1\})$$

and therefore $n_i(a) = n_i(\alpha) = 0 = n_i(\{\lambda_1\})$. The jobs $a \cup \alpha \cup \{\lambda_1\} \cup A$ are the only jobs that are contained on machine M_1. Since $n_i(A) = 0$ as well, it has to be that $\sigma(i) = 0$. If $i \in B$ we can prove $\sigma(i) = 0$ analogously using equality (4.2).

Since the schedule stays valid if we mirror the schedule such that the new start points are $s'(i) = W - \sigma(i) - p(i)$ for each job i, the last job has to be in the set $A \cup B$ as well. □

Next, we will show that the items in the sets A and B have to be scheduled alternatingly. Let $(A_0, \ldots, A_z)$ be the set A and $(B_0, \ldots, B_z)$ be the set B, each ordered by increasing size of the starting points. Simply swap the jobs if they do not have this order.

Lemma 4.6. *If $\sigma(B_0) = 0$, it holds for each item $i \in \{0, \ldots, z\}$ that*

$$\begin{aligned} i =& n_{A_i}(A) = n_{A_i}(B) - 1 = n_{A_i}(c) - 1 \\ =& n_{A_i}(\alpha) = n_{A_i}(b) = n_{A_i}(a), \end{aligned} \tag{4.6}$$

and

$$\begin{aligned} i &= n_{B_i}(B) = n_{B_i}(A) = n_{B_i}(c) \\ &= n_{B_i}(\beta) = n_{B_i}(a) = n_{B_i}(b), \end{aligned} \tag{4.7}$$

and $n_{A_i}(\{\lambda_1\}) = 1$ *as well as* $n_{B_i}(\{\lambda_2\}) = 0$.

Proof. We will prove this lemma by proving the following claim

Claim 4.7. If $\sigma(B_0) = 0$, it holds for each item $i \in \{0, \dots, z\}$ that

$$n_{A_i}(A) = n_{A_i}(B) - n_{A_i}(\{\lambda_1\})$$

and $n_{A_i}(\{\lambda_1\}) = 1$.

We will prove this claim inductively and per contradiction. Assume for contradiction that

$$n_{A_0}(B) - n_{A_0}(\{\lambda_1\}) > n_{A_0}(A) = 0.$$

Therefore, we have $1 \leqslant n_{A_0}(B) - n_{A_0}(\{\lambda_1\})$. Let $a' \in a$, $b' \in b$ and $c' \in c$ be the first started jobs in their sets. Since

$$\begin{aligned} n_{A_0}(b) &=_{(4.1)} n_{A_0}(a) =_{(4.1)} n_{A_0}(c) - n_{A_0}(\{\lambda_1\}) \\ &=_{(4.1)} n_{A_0}(B) - n_{A_0}(\{\lambda_1\}) \geqslant 1, \end{aligned}$$

the jobs a', b' and c' start before A_0. It holds that $n_{b'}(c) =_{(4.4)} n_{b'}(A) = 0$. Therefore, c' has to start after b' resulting in $n_{c'}(b) \geqslant 1$. Furthermore, we have $n_{a'}(c) =_{(4.3)} n_{a'}(B) \geqslant 1$. Hence, c' has to start before a' resulting in $n_{c'}(a) = 0$. In total we have $1 \leqslant n_{c'}(b) =_{(4.5)} n_{c'}(a) = 0$, a contradiction. Therefore, we have $n_{A_0}(B) - n_{A_0}(\{\lambda_1\}) \leqslant n_{A_0}(A) = 0$. As a consequence, it holds that $1 \leqslant n_{A_0}(B) \leqslant n_{A_0}(\{\lambda_1\}) \leqslant 1$ and we can conclude $n_{A_0}(B) = 1 = n_{A_0}(\{\lambda_1\})$ as well as $n_{A_0}(B) - n_{A_0}(\{\lambda_1\}) = n_{A_0}(A)$. Therefore, $1 = n_{A_i}(\{\lambda_1\})$ holds for all $i \in \{0, \dots, z\}$. To this point we have proved the induction basis.

For the induction step it is enough to prove that $n_{A_i}(B) - 1 = n_{A_i}(A)$ for all $i \in \{0, \dots, z\}$. To prove this, we first show that if this condition holds for for all i' up to an $i \in \{0, \dots, z\}$, we can derive the equation (4.6) and an equation which is similar to equation (4.7) for all these i'. Choose $i \in \{0, \dots, z\}$ such that

$$n_{A_{i'}}(B) - 1 = n_{A_{i'}}(A) \tag{4.8}$$

for all $i' \in \{0, \dots, i\}$. A direct consequence from this equation is the equation (4.6) for all $i' \in \{0, \dots, i\}$:

$$i' = n_{A'_i}(A) =_{(4.8)} n_{A'_i}(B) - 1 =_{(4.1)} n_{A'_i}(c) - 1$$

$$=_{(4.1)} n_{A'_i}(\alpha) =_{(4.1)} n_{A'_i}(b) =_{(4.1)} n_{A'_i}(a).$$

Furthermore, we have $n_{B_i}(B) = i = n_{A_i}(A) = n_{A_i}(B) - 1$. Therefore, B_i has to be scheduled before A_i. Additionally, we have $n_{B_i}(B) - 1 = n_{B_{i-1}}(B) = i - 1 = n_{A_{i-1}}(A) = n_{A_{i-1}}(B) - 1$, so B_i has to be scheduled after A_{i-1}. Therefore, we have $n_{B_i}(B) = n_{B_i}(A)$ and the following equation is a consequence for all $i' \in \{0, \ldots, i\}$:

$$\begin{aligned} i' &= n_{B_{i'}}(B) = n_{B_{i'}}(A) =_{(4.2)} n_{B_{i'}}(c) \\ &=_{(4.2)} n_{B_{i'}}(\beta) + n_{B_{i'}}(\{\lambda_2\}) =_{(4.2)} n_{B_{i'}}(a) + n_{B_{i'}}(\{\lambda_2\}) \\ &=_{(4.2)} n_{B_{i'}}(b) + n_{B_{i'}}(\{\lambda_2\}). \end{aligned} \tag{4.9}$$

We still have to prove Claim 4.7 to prove the lemma. To this point, we have proved the base of the induction. However, we still have to prove $n_{A_{i+1}}(B) - 1 = n_{A_{i+1}}(A)$ since we already know that $n_{A_i}(\{\lambda_1\}) = 1$ for all $i \in \{0, \ldots, z\}$. We will prove this in two steps:

Claim 4.8. $n_{A_{i+1}}(B) - 1 \leqslant n_{A_{i+1}}(A)$

Proof. **Assume** for contradiction that $n_{A_{i+1}}(B) - 1 > n_{A_{i+1}}(A)$. As a consequence, we have

$$\begin{aligned} 1 &= n_{A_{i+1}}(A) - n_{A_i}(A) < n_{A_{i+1}}(B) - 1 - (n_{A_i}(B) - 1) \\ &= n_{A_{i+1}}(B) - n_{A_i}(B), \end{aligned}$$

and hence, $n_{A_{i+1}}(B) - n_{A_i}(B) \geqslant 2$. Therefore, there are jobs $B_{i+1}, B_{i+2} \in B$ that are scheduled between A_i and A_{i+1}. Since we have

$$\begin{aligned} 2 &\leqslant n_{A_i}(B) - 1 - (n_{A_i}(B) - 1) =_{(4.1)} n_{A_i}(c) - 1 - (n_{A_{i+1}}(c) - 1) \\ &=_{(4.1)} n_{A_i}(b) - n_{A_{i+1}}(b) =_{(4.1)} n_{A_i}(a) - n_{A_{i+1}}(a), \end{aligned}$$

there have to be two jobs $a', a'' \in a$, $b', b'' \in b$ and $c', c'' \in c$ that are scheduled between A_i and A_{i+1} as well. W.l.o.g. we assume that $\sigma(a') \leqslant \sigma(a'')$, $\sigma(b') \leqslant \sigma(b'')$ and $\sigma(c') \leqslant \sigma(c'')$.

Next, we will deduce in which order the jobs a', a'', b', b'', c', c'', B_{i+1}, and B_{i+2} appear in the schedule. It holds that

$$\begin{aligned} n_{b''}(c) &=_{(4.4)} n_{b''}(A) \overset{\sigma(A_i)<\sigma(b'')}{=} n_{A_i}(A) + 1 \\ &=_{(4.8)} n_{A_i}(B) =_{(4.1)} n_{A_i}(c) \end{aligned}$$

since b'' starts after A_i but before A_{i+1}. Therefore, b' and b'' have to finish before c', i.e., $\sigma(b') < \sigma(b'') < \sigma(c')$, since no job from the set c can be scheduled between A_i and b''. As a consequence, we have

$$n_{c'}(a) =_{(4.5)} n_{c'}(b) \overset{\sigma(A_i)<\sigma(b')<\sigma(b'')<\sigma(c')}{\geqslant} n_{A_i}(b) + 2 =_{(4.1)} n_{A_i}(a) + 2.$$

Hence, a'' has to start before c' as well. Additionally, it holds that

$$\begin{aligned} n_{B_{i+2}}(c) &=_{(4.2)} n_{B_{i+2}}(A) \overset{\sigma(A_i)<\sigma(B_{i+2})}{=} n_{A_i}(A) + 1 \\ &=_{(4.8)} n_{A_i}(B) =_{(4.1)} n_{A_i}(c), \end{aligned}$$

since B_{i+2} starts after A_i but before A_{i+1}. As a consequence, B_{i+2} has to start before c'. Additionally, it holds that

$$n_{a''}(B) =_{(4.3)} n_{a''}(c) \overset{\sigma(A_i)<\sigma(a'')<\sigma(c')}{=} n_{A_i}(c) =_{(4.1)} n_{A_i}(B).$$

Therefore, a'' has to start before B_{i+1}, since there cannot be a job from the set B scheduled between A_i and a''.

To this point, we have deduced that the jobs have to appear in the following order in the schedule: A_i, a', a'', B_{i+1}, B_{i+2}, c', c'', A_{i+1}. However, this schedule is not feasible, since we have

$$\begin{aligned} n_{A_i}(a) + 2 &\overset{\sigma(A_i)<\sigma(a')<\sigma(a')<\sigma(B_{i+1})}{\leqslant} n_{B_{i+1}}(a) \leqslant_{(4.2)} n_{B_{i+1}}(A) \\ &\overset{\sigma(A_i)<\sigma(B_{i+1})<\sigma(A_{i+1})}{=} n_{A_i}(A) + 1 =_{(4.1)} n_{A_i}(a) + 1, \end{aligned}$$

a contradiction. ↯

Therefore, the assumption $n_{A_{i+1}}(B) - 1 > n_{A_{i+1}}(A)$ was wrong, and it holds that $n_{A_{i+1}}(B) - 1 \leqslant n_{A_{i+1}}(A)$ proving Claim 4.8. ◁

To proof the equality, we show the other direction in the next step.

Claim 4.9. $n_{A_{i+1}}(B) - 1 \geqslant n_{A_{i+1}}(A)$

Proof. **Assume** for contradiction that $n_{A_{i+1}}(B) - 1 < n_{A_{i+1}}(A)$. As a consequence, it holds that $n_{A_{i+1}}(B) = n_{A_i}(B)$ since

$$n_{A_i}(B) - 1 \leqslant n_{A_{i+1}}(B) - 1 \overset{\text{assumption}}{\leqslant} n_{A_{i+1}}(A) - 1 = n_{A_i}(A) = n_{A_i}(B) - 1.$$

Furthermore, there has to be at least one job $B_{i+1} \in B$ that starts after

A_{i+1} since $|A| = |B|$. Therefore, we have $n_{B_{i+1}}(c) - n_{B_i}(c) = n_{B_{i+1}}(A) - n_{B_i}(A) \geqslant 2$. As a consequence, there are jobs $c', c'' \in c$ which are scheduled between B_i and B_{i+1}. Let c' be the first job in c scheduled after B_i and c'' be the next. Since we do not know the value of $n_{B_i}(\{\lambda_2\})$ or $n_{B_{i+1}}(\{\lambda_2\})$, we can just deduce from equation (4.2) that $n_{B_{i+1}}(a) - n_{B_i}(a) \geqslant 1$. Therefore, there has to be a job $a' \in a$ that is scheduled between B_i and B_{i+1}.

We will now look at the order in which the jobs A_i, A_{i+1}, c', c'' and a' have to be scheduled. First, we know that A_i and A_{i+1} have to be scheduled between c' and c'', since

$$\begin{aligned} n_{A_i}(c) &=_{(4.1)} n_{A_i}(B) \overset{\sigma(B_i)<\sigma(A_i)<\sigma(B_{i+1})}{=} n_{B_i}(B) + 1 \\ &=_{(4.9)} n_{B_i}(A) + 1 =_{(4.2)} n_{B_i}(c) + 1 \end{aligned}$$

and

$$\begin{aligned} n_{A_{i+1}}(c) &=_{(4.1)} n_{A_{i+1}}(B) \overset{\sigma(B_i)<\sigma(A_{i+1})<\sigma(B_{i+1})}{=} n_{B_i}(B) + 1 \\ &=_{(4.9)} n_{B_i}(A) + 1 =_{(4.2)} n_{B_i}(c) + 1, \end{aligned}$$

and hence there has to be exactly one job from the set c scheduled between B_i and A_i, as well as B_i and A_{i+1}. Furthermore, we know that a' has to be scheduled between c' and c'' as well, since

$$\begin{aligned} n_{a'}(c) &=_{(4.3)} n_{a'}(B) \overset{\sigma(B_i)<\sigma(a')}{=} n_{B_i}(B) + 1 \\ &=_{(4.9)} n_{B_i}(A) + 1 =_{(4.2)} n_{B_i}(c) + 1. \end{aligned}$$

As a consequence, we can deduce that there is a job $b' \in b$ which is scheduled between c' and c'', since

$$n_{c''}(b) =_{(4.5)} n_{c''}(a) \overset{\sigma(c')<\sigma(a')<\sigma(c'')}{\geqslant} n_{c'}(a) + 1 =_{(4.5)} n_{c'}(b) + 1.$$

We know about this b' that

$$n_{b'}(A) =_{(4.4)} n_{b'}(c) \overset{\sigma(B_i)<\sigma(c')<\sigma(b')<\sigma(c'')}{=} n_{B_i}(c) + 1 =_{(4.2)} n_{B_i}(A) + 1,$$

so b' has to be scheduled between A_i and A_{i+1}.

In summary, the jobs are scheduled as follows: B_i, c', A_i, b', A_{i+1}, c'',

B_{i+1}. However, this schedule is infeasible since

$$\begin{aligned} n_{A_i}(b) &=_{(4.1)} n_{A_i}(B) - 1 \overset{\text{assumption}}{=} n_{A_{i+1}}(B) - 1 \\ &=_{(4.1)} n_{A_{i+1}}(b) = n_{A_i}(b) + 1, \end{aligned}$$

a contradiction. ↯

As a consequence, it has to hold that $n_{A_{i+1}}(B) - 1 \geqslant n_{A_{i+1}}(A)$ proving Claim 4.9. ◁

Altogether as a consequence of Claim 4.8 and Claim 4.9, we have proved that $n_{A_{i+1}}(B) - n_{A_{i+1}}(\{\lambda_1\}) = n_{A_{i+1}}(A)$ for all $i \in \{0, \dots, z\}$. This concludes the proof of the Claim 4.7.

Last we have to prove the equation (4.7). To do this we have to prove that $n_{B_i}(\{\lambda_1\}) = 0$ for all $i \in \{0, \dots, z\}$. We do this by proving the following claim.

Claim 4.10. λ_2 is scheduled after the last job in B.

To prove the equation (4.7), we have to prove that $n_{B_i}(\{\lambda_2\}) = 0$ for each $i \in \{0, \dots, z\}$, i.e., we have to prove Claim 4.10. Assume there is an $i \in \{0, \dots, z\}$ with $n_{B_i}(\{\lambda_2\}) > 0$. Let i be the smallest of these indices. We know that

$$i - 1 =_{(4.9)} n_{B_i}(A) - 1 = n_{B_i}(A) - n_{B_i}(\{\lambda_2\}) =_{(4.2)} n_{B_i}(a).$$

Since $n_{A_i}(b) =_{(4.1)} n_{A_i}(a) =_{(4.6)} i = n_{B_i}(a) + 1 =_{(4.2)} n_{B_i}(b) + 1$ there has to be an unique $a' \in a$ and an unique $b' \in b$ scheduled between B_i and A_i. Furthermore, since $n_{A_i}(c) =_{(4.6)} i + 1$ and $n_{B_i}(c) =_{(4.9)} i$, there has to be a $c' \in c$ scheduled between B_i and A_i as well. At the start of b' it holds that $n_{b'}(c) =_{(4.4)} n_{b'}(A) = n_{A_{i-1}}(A) + 1 =_{(4.1)} n_{A_{i-1}}(c)$, so b' has to start before c'. Additionally, at the start of a' we have $n_{a'}(c) =_{(4.4)} n_{a'}(B) = n_{B_i}(B) + 1 =_{(4.9)} n_{B_i}(c) + 1$ and therefore a' hast to start after c'. In total, the jobs appear in the following order: B_i, b', c', a', A_i. But this cannot be the case since we have

$$\begin{aligned} n_{B_{i-1}}(a) &\overset{(\sigma(c') < \sigma(a'))}{=} n_{c'}(a) =_{(4.5)} n_{c'}(b) \\ &\overset{(\sigma(B_i) < \sigma(b') < \sigma(c'))}{=} n_{B_{i-1}}(b) + 1 =_{(4.2)} n_{B_{i-1}}(a) + 1. \end{aligned}$$

Hence, we have contradicted that assumption. Therefore, it holds that

$n_{B_i}(\{\lambda_2\}) = 0$ for all $i \in \{0, \ldots, z\}$ and equation (4.7) is a consequence:

$$n_{B_i}(b) = n_{B_i}(a) = n_{B_i}(c) = n_{B_i}(\beta) = n_{B_i}(A) = n_{B_i}(B) = i.$$

This concludes the proof of the lemma. □

A direct consequence of Lemma 4.6 is that the last job on M_2 is a job in A. Since the equations (4.1) and (4.2), as well as (4.3) and (4.4), are symmetric, we can deduce an analogue statement if the first job on M_2 is in A. More precisely, we can show that $n_{B_i}(A) - n_{B_i}(\{\lambda_2\}) = n_{B_i}(B)$ and $n_{B_i}(\{\lambda_2\}) = 1$ for each $B_i \in B$ in this case. This would imply that the last job on M_2 is a job in B. Since we can mirror the schedule such that the last job is the first job, we can suppose that the first job on M_2 is a job in B.

Lemma 4.11. *If the optimal schedule for the scheduling instance derived from $\mathcal{I}$ has makespan W then $\mathcal{I}$ is a Yes-instance for 3-Partition and we can transform the schedule such that all jobs are scheduled on contiguous machines.*

Proof. First, we will prove that M_1 processes the jobs $A \cup a \cup \alpha \cup \{\lambda_1\}$ in the order λ_1, A_0, a_1, α_1, A_1, a_2, α_2, A_2, …, a_z, α_z, A_z, where $a_i \in a$ and $\alpha_i \in \alpha$ for each $i \in \{1, \ldots z\}$, see Figure 4.6. Lemma 4.6 ensures that the first job on M_1 is the job λ_1. Furthermore, since $0 = n_{A_0}(A) =_{(4.6)} n_{A_0}(\alpha) =_{(4.6)} n_{A_0}(a)$, the second job on M_1 is A_0. For each $i \in \{1, \ldots, z\}$ it holds that $n_{A_i}(\alpha) =_{(4.6)} n_{A_{i-1}}(\alpha) + 1$ and $n_{A_i}(a) =_{(4.6)} n_{A_{i-1}}(a) + 1$. Therefore, there is exactly one job $a_i \in a$ and one job $\alpha_i \in \alpha$ scheduled between the jobs A_{i-1} and A_i. It holds that $n_{A_{i-1}}(a) + 1 =_{(4.6)} i =_{(4.7)} n_{B_i}(a)$. Therefore, a_i has to be scheduled between A_{i-1} and B_i. As a consequence, we have $n_{a_i}(\alpha) + 1 = n_{a_i}(\alpha) + n_{a_i}(\{\lambda_1\}) =_{(4.3)} n_{a_i}(B) = n_{B_i}(B) =_{(4.7)} n_{B_i}(a) = n_{a_i}(a) + 1$. Therefore, a_i has to be scheduled before α_i and the jobs appear in machine M_1 in the described order. As a result, we know about the start point of A_i that

$$\begin{aligned}\sigma(A_i) &= p(\lambda_1) + ip_a + ip_\alpha + ip_A \\ &= (i+1)(D^2 + D^3) + i(D^4 + D^5 + D^6 + D^7) + (7zi + z)D^8.\end{aligned}$$

Now, we will show that the machine M_4 processes the jobs $B \cup b \cup \beta \cup \{\lambda_2\}$ in the order B_0, β_1, b_1, B_1, β_2, b_2, B_2, …, β_z, b_z, B_z, λ_2, see Figure 4.7. The first job on M_4 is the job B_0 since Lemma 4.5 states that one of the jobs in $A \cup B$ has start point 0 and we decided w.l.o.g. that B_0 is

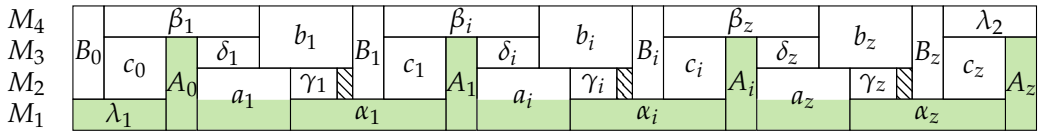


Figure 4.6. Proved positions of the jobs in the sets A, a, and α.

this job. Equation (4.7) ensures that between the jobs B_i and B_{i+1} there is scheduled exactly one job $b_{i+1} \in b$ and exactly one job $\beta_{i+1} \in \beta$. It holds that $n_{A_i}(b) + 1 =_{(4.6)} i + 1 =_{(4.7)} n_{B_{i+1}}(b)$. Therefore, b_{i+1} has to be scheduled between A_i and B_{i+1}. As a consequence, it holds that

$$\begin{aligned} n_{b_{i+1}}(\beta) &\overset{n_{b_{i+1}}(\{\lambda_2\})=0}{=} n_{b_{i+1}}(\beta) + n_{b_{i+1}}(\{\lambda_2\}) =_{(4.4)} n_{b_{i+1}}(A) \\ &\overset{\sigma(A_i)<\sigma(b_{i+1})<\sigma(B_{i+1})}{=} n_{B_{i+1}}(A) =_{(4.7)} n_{B_{i+1}}(b) \\ &\overset{\sigma(A_i)<\sigma(b_{i+1})<\sigma(B_{i+1})}{=} n_{b_{i+1}}(b) + 1. \end{aligned}$$

Hence, b_{i+1} has to be scheduled after β_{i+1} and the jobs on machine M_4 appear in the described order. As a result, we know about the start point of B_i that

$$\begin{aligned} \sigma(B_i) =& ip_b + ip_\beta + ip_B \\ =& iD^2 + iD^3 + iD^4 + iD^5 + iD^6 + iD^7 + (i(7z-1))D^8. \end{aligned}$$

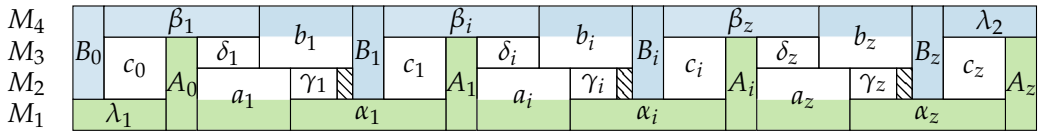


Figure 4.7. Proved positions of the jobs in the sets B, b, and β.

Next, we can deduce that the jobs in c are scheduled as shown in Figure 4.7. We have $n_{B_i}(c) =_{(4.7)} i =_{(4.6)} n_{A_i}(c) - 1$. Therefore, there exists an $c' \in c$ for each $i \in \{0, \ldots, z\}$, which is scheduled between B_i and A_i. The processing time between B_i and A_i is exactly $\sigma(A_i) - \sigma(B_i) - p(B_i) = D^3 + (z+i)D^8$. As a consequence, one can see with an inductive argument that $c_i \in c$ with $p(c_i) = D^3 + (z+i)D^8$ has to be positioned between B_i

and A_i since the job in c with the largest processing time, c_z, fits between B_z and A_z only.

In this step, we will transform the schedule such that all jobs are scheduled on contiguous machines. To this point, this property is obviously fulfilled by the jobs in $A \cup B \cup c$. However, the jobs in $a \cup b$ might be scheduled on non-contiguous machines. We know that the a_i and b_i are scheduled between A_{i-1} and B_i. One part of a_i is scheduled on M_1 and one part of b_i is scheduled on M_4, while each other part is scheduled either on M_2 or on M_3 but both parts on different machines, because $\sigma(B_i) - \sigma(A_{i-1}) - p(A_i) = D^5 + D^6 + D^7 + (6z - i)D^8 < D^5 + 2D^6 + D^7 + 6zD^8 = p(a_i) + p(b_i)$ for each $i \in \{0, \ldots, z\}$. Since A_i and B_{i+1} both are scheduled on machines M_2 and M_3, we can swap the content of the machines between these jobs such that the other part of a_i is scheduled on M_2 and the other part of b_i is scheduled on M_3. We do this swapping step for all $i \in \{0, \ldots, z-1\}$ such that all other parts of jobs in a are scheduled on M_2 and all other parts of jobs in b are scheduled on M_3 respectively. After this swapping step, all jobs are scheduled on contiguous machines.

Now, we will show that $\mathcal{I}$ is a Yes-instance. To this point we know that M_2 contains the jobs $A \cup B \cup a \cup c$. Since $\check{a} = a$ and $|\check{a}| = |\check{\gamma}|$, it has to hold by Lemma 4.3 that $\check{\gamma} = \gamma$ implying that M_2 contains all jobs in γ. Furthermore, since $\check{b} = \varnothing$ and $|\check{b}| = |\check{\delta}|$, we have $\check{\delta} = \varnothing$ and therefore M_2 does not contain any job in δ. Besides the jobs $A \cup B \cup a \cup c \cup \gamma$, M_2 processes further jobs with total processing time zD. Therefore, all the jobs in P are processed on M_2. We will now analyse where the jobs in γ are scheduled. The only possibility where these jobs can be scheduled is the time between the end of a_i and the start of B_i for each $i \in \{1, \ldots, z\}$ since at each other time the machine is occupied by other jobs. The processing time between the end of a_i and the start of B_i is exactly $\sigma(B_i) - \sigma(A_{i-1}) - p(A_{i-1}) - p(a_i) = D^7 + (3z - i)D^8$. The job in γ with the largest processing time is the job γ_1 with $p(\gamma_1) = D^7 + (3z-1)D^8 - D$. This job only fits between a_1 and B_1. Inductively we can show that $\gamma_i \in \gamma$ with $p(\gamma_i) = D^7 + (3z - i)D^8 - D$ has to be scheduled between a_i and B_i on M_2. Furthermore, since $p(\gamma_i) = D^7 + (3z - i)D^8 - D$ and the processing time between the end of a_i and the start of B_i is $D^7 + (3z - i)D^8$, there is exactly D processing time left. This processing time has to be occupied by the jobs in P since this schedule has no idle times. Therefore, we have

for each $i \in \{1, \dots, z\}$ a subset $P_i \subseteq P$ containing jobs with processing times adding up to D such that $P_1 \dot{\cup} \dots \dot{\cup} P_z = P$. As a consequence $\mathcal{I}$ is a Yes-instance. □

4.3 Hardness of Strip Packing

In this section, we will prove the Theorem 4.2. This can be done straight forward, by using the reduction from above. Note that in the transformed optimal schedule, all jobs are scheduled on contiguous machines, i.e., the machines the jobs are scheduled on are neighbors in the natural order (M_1, M_2, M_3, M_4). As a consequence, we have proved that this problem is strongly NP-complete even if we restrict the set of feasible solutions to those where all jobs are scheduled on contiguous machines. We will now describe how this insight delivers a lower bound of $\frac{5}{4}$ for the best possible approximation ratio for pseudo-polynomial Strip Packing and in this way prove the Theorem.

To show our hardness result for Strip Packing, let us consider the following instance. We define $W := (z+1)(D^2 + D^3 + D^4) + z(D^5 + D^6 + D^7) + z(7z+1)D^8$ as the width of the strip, i.e., it is the same as the considered makespan in the scheduling problem. For each job j defined in the reduction above, we introduce an item i with $w(i) = p(j)$ and height $h(i) = m(j)$. Now, we can show analogously that if the 3-Partition instance is a Yes-instance, there is a packing of height 4 (one example is the packing in Figure 4.4); and on the other hand if there is a packing with height 4, the 3-Partition instance has to be a Yes-instance. If the 3-Partition instance is a No-instance, the optimal packing has a height of at least 5 since the optimal height for this instance is integral. Therefore, we cannot approximate Strip Packing better than $\frac{5}{4}$ in pseudo-polynomial time unless P = NP.

Note that this inapproximability result transfers to Strip Packing With Rotations. The generated items all have a width larger than 4 and hence rotating an item will lead to a larger packing height than 4. Hence rotating is an option that is never used in an optimal schedule. Therefore, when allowing rotations, we still can decide for each partition instance if it is a yes or no instance if we can solve Strip Packing With Rotations with

approximation ratio better than 5/4.

Variants of Strip Packing

Lastly we look at a variant of the Strip Packing problem called Contiguous Moldable Task Scheduling. In this problem setting we are given an (arbitrary large) set of m machines, which have some kind of total order, and a set of parallel tasks $\mathcal{J}$. Each task $j \in \mathcal{J}$ has a set $D_j \subseteq \{1, \ldots, m\}$ of machine amounts it can be processed on, e.g., if $D_j = \{3, 7\}$ the job j can be processed either on three or on seven machines. For each $q \in D_j$ it has an individual processing time $p(j, q) \in \mathbb{N}_{>0} \cup \{\infty\}$. A schedule S is given by two functions $\sigma : \mathcal{J} \to \mathbb{N}$ and $\rho : \mathcal{J} \to 2^{\{1,\ldots,m\}}$. The function σ maps each job to a start point in the schedule, while ρ maps each job to an *interval* of machines it is processed on. A schedule is feasible if each machine processes at most one job at a time and each job is processed on the required number of machines, (i.e., $|\rho(j)| \in D_j$).

This problem directly contains the Strip Packing problem as a special case by setting $D_i = \{w(i)\}$ and $p(i, w(i)) = h(i)$ for each item $i \in I$, and hence, it is NP-hard to approximate it better than $\frac{5}{4}$ in pseudo-polynomial time. However, it is also NP-hard to find a better approximation than $\frac{5}{4}$ if we require $D_j = \{1, \ldots, m\}$ and $p(j, q) \in \mathbb{N}_{>0}$ for each job and number of machines. To show this, we use the reduction from above. The as number of machines we define $m := (z+1)(D^2 + D^3 + D^4) + z(D^5 + D^6 + D^7) + z(7z+1)D^8$. For each item $i \in I$ constructed for the Strip Packing instance, we introduce one job i with $p(i, q) = 5$, if $q < w(i)$ and $p(i, q) = h(i)$, if $q \geqslant w(i)$. Obviously a schedule with height 4 can be found if and only if each job i uses $w(i)$ machines and the 3-Partition instance is a Yes-instance. Otherwise the optimal schedule has a makespan of 5. Hence it is not possible to find an algorithm with approximation ratio better than 5/4, unless P = NP.

Note that the processing time function is not monotone in this construction, i.e., we do not have $p(j, q) \cdot q \leqslant p(j, q+1) \cdot (q+1)$ for each $q \in \{1, \ldots, m-1\}$. Hence, there could be a PTAS for the monotone case.

4.4 Conclusion

In this chapter, we positively answered the long standing open question whether the problem $P4|size_j|C_{max}$ is strongly NP-complete. This closes the gap between strongly NP-completeness for at least 4 machines, and the possibility to solve the problem in pseudo-polynomial time for at most 3 machines.

Furthermore, we have improved the lower bound for pseudo-polynomial Strip Packing to $\frac{5}{4}$. The best known published algorithm has an approximation ratio of $\frac{4}{3} + \varepsilon$. This leaves a gap between the lower bound and the best known algorithm. However, we where able to find a pseudo-polynomial time algorithm with approximation ratio $\frac{5}{4} + \varepsilon$ [63], which closes this gap. We will discuss this result in the following chapter.

Lastly, we have considered Contiguous Moldable Task Scheduling and proved that in the non-monotone case there is no pseudo-polynomial time algorithm with approximation ratio better than $5/4$ unless $\mathrm{P} = \mathrm{NP}$. However, in the monotone case, finding a PTAS might be possible. In our opinion, it is an interesting open problem, whether there is a PTAS for the monotone case or not, especially since there is an FPTAS for the case that $m > 8n/\varepsilon$, see [56].

Chapter 5

A Tight Pseudo-Polynomial Time Approximation for Strip Packing

In this chapter, we study a pseudo-polynomial algorithm for the Strip Packing problem. In the previous chapter, we have seen that there is no pseudo-polynomial algorithm for Strip Packing with a ratio better than $5/4$ unless P = NP. The best algorithm so far has a ratio of $4/3 + \varepsilon$. We close this gap between inapproximability result and best known algorithm by presenting an algorithm with approximation ratio $5/4 + \varepsilon$ and thus categorize the problem accurately. The algorithm uses a structural result which states that each optimal solution can be transformed such that it has one of a polynomial number of different forms. The strength of this structural result is that it applies to other problem settings as well, for example to Strip Packing with rotations (90 degrees) and Contiguous Moldable Task Scheduling. This fact enabled us to present algorithms with approximation ratio $5/4 + \varepsilon$ for these problems as well.

The results of this chapter were previously published in two papers. The first, which describes a $(4/3 + \varepsilon)$ approximation, was first published as an extended abstract in WALCOM [63] and a long version was excepted by Theoretical Computer Science [62], while an extended abstract of the second was published on ESA [60] and a preprint of the full version can be found on arXiv [61].

5.1 Results

Remember, in the Strip Packing (SP) problem, we have to pack a set $\mathcal{I}$ of rectangular items into a given strip with width $W \in \mathbb{N}$ and infinite height.

Each item $i \in \mathcal{I}$ has a width $w(i) \in \mathbb{N}_{\leqslant W}$ and a height $h(i) \in \mathbb{N}$. The area of an item $i \in \mathcal{I}$ is defined as $\text{area}(i) := h(i) \cdot w(i)$ and the area of a set of items $\mathcal{I}' \subseteq \mathcal{I}$ is defined as $\text{area}(\mathcal{I}') := \sum_{i \in \mathcal{I}'} h(i) \cdot w(i)$.

A packing of the items is given by a mapping $\rho : I \to \mathbb{N}_{\leqslant W} \times \mathbb{N}$ which assigns the lower left corner of an item $i \in I$ to a position $\rho(i) = (x_i, y_i)$ in the strip. An inner point of $i \in \mathcal{I}$ (with respect to a packing ρ) is a point from the set $\text{inn}(i) := \{(x, y) \in \mathbb{R} \times \mathbb{R} | x_i < x < x_i + w(i), y_i < y < y_i + h(i)\}$. We say two items $i, j \in \mathcal{I}$ overlap if they share an inner point (i.e., $\text{inn}(i) \cap \text{inn}(j) \neq \varnothing$). A packing is feasible if no two items overlap and if $x_i + w(i) \leqslant W$ for all $i \in I$. The objective of Strip Packing is to find a feasible packing ρ with minimal height $h(\rho) := \max\{y_i + h(i) | i \in \mathcal{I}, \rho(i) = (x_i, y_i)\}$. In the following, given an instance I of the Strip Packing problem, we will denote this minimal packing height with $\text{OPT}(I)$ and dismiss the I if the instance is clear from the context.

In this chapter, we study approximation algorithms which have a pseudo-polynomial running time with respect to the width of the strip, i.e., we consider algorithms where the width of the strip W is allowed to appear polynomially in the running time. In the last chapter, Chapter 4, we proved that we cannot find an algorithm with approximation ratio strictly better than $5/4$ unless P = NP. On the other hand, the algorithm with the best ratio so far computes a $4/3 + \varepsilon$ approximation [33, 63]. This yields a large gap and it was unknown whether the ratio of the algorithm or the lower bound was tight. We manage to close this gap and thus categorize the complexity of the problem correctly. We accomplish this by proving a strong result about the structure of optimal solutions, which enables us to close the gap to the lower bound, except for a negligibly small ε, see Figure 5.1 for an overview.

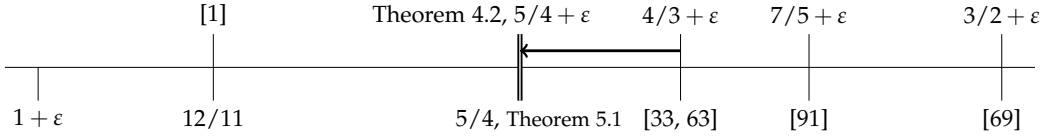


Figure 5.1. The upper and lower bounds for pseudo-polynomial approximations.

Theorem 5.1. *There is a pseudo-polynomial algorithm for Strip Packing which finds a* $(5/4 + \varepsilon)$*-approximation in* $\mathcal{O}(n \log(n)) \cdot W^{\mathcal{O}_\varepsilon(1)}$ *operations.*

The structural result applies to other problem settings as well and, therefore, the algorithmic result can be extended to them. One example is the setting of Strip Packing where we are allowed to rotate the items by 90 degrees. In this setting, the items still have to be placed axis-aligned, but we can decide if the longer or shorter side defines the height of the item.

Theorem 5.2. *There is a pseudo-polynomial algorithm for Strip Packing With Rotations which finds a $(5/4+\varepsilon)$-approximation in $(nW)^{\mathcal{O}_\varepsilon(1)}$ operations.*

A generalization of Strip Packing is the Contiguous Moldable Task Scheduling problem. In this setting, we are given a set of jobs $\mathcal{J}$ and a set of m machines. Each job $j \in \mathcal{J}$ can be scheduled on different numbers of machines given by $M_j \subseteq \{1,\dots,m\}$. Depending on the number of machines $i \in M_j$, each job $j \in \mathcal{J}$ has a specific processing time $p_j(i) \in \mathbb{N}$.

A schedule S is given by three functions: $\sigma : \mathcal{J} \to \mathbb{N}$ which maps each job $j \in \mathcal{J}$ to a starting time $\sigma(j)$; $\rho : \mathcal{J} \to \{1,\dots,m\}$ which maps each job $j \in \mathcal{J}$ to the number of processors $\rho(j) \in M_j$ it is processed on; and $\varphi : \mathcal{J} \to \{1,\dots,m\}$ which maps each job $j \in \mathcal{J}$ to the first machine it is processed on. The job $j \in \mathcal{J}$ will use the machines $\varphi(j)$ to $\varphi(j)+\rho(j)-1$ contiguously. A schedule $S = (\sigma, \rho, \varphi)$ is feasible if each machine processes at most one job at a time and its makespan is defined by $\max_{j\in\mathcal{J}} \sigma(j) + p_j(\rho(j))$. The objective is to find a feasible schedule, which minimizes the makespan.

This problem is a true generalization of Strip Packing as it contains this problem (and Strip Packing With Rotations) as a special case: We define the number of machines m as the width of the strip W and for each item $i \in \mathcal{I}$ we introduce one job i with $M_i := \{w(i)\}$ and processing time $p_i(w(i)) = h(i)$ (or introduce one job i with $M_i := \{w(i), h(i)\}$ and processing times $p_i(w(i)) = h(i)$ and $p_i(h(i)) = w(i)$ respectively). Therefore, we cannot hope for a pseudo-polynomial algorithm with a ratio better than $5/4$ unless P = NP. We managed to adapt the algorithmic result to find an algorithm with an approximation ratio, which almost matches this bound.

Theorem 5.3. *There is a pseudo-polynomial algorithm for Contiguous Moldable Task Scheduling which finds a $(5/4+\varepsilon)$-approximation in $(nm)^{\mathcal{O}_\varepsilon(1)}$ operations.*

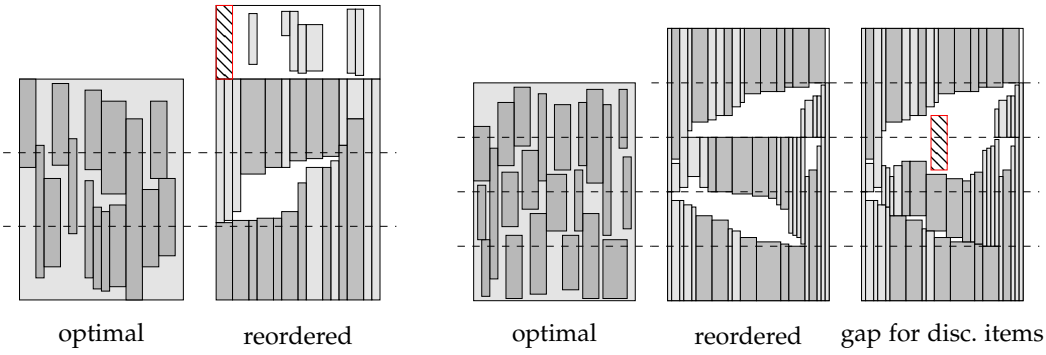
We say a job $j \in \mathcal{J}$ is monotone if the work of the job $w(p_j(i)) := p_j(i) \cdot i$

does not increase if we decrease the number of machines. There is an FPTAS by Jansen and Land [56] for the case that all jobs are monotonic and $m \geqslant 8n/\varepsilon$. For the case that $m < 8n/\varepsilon$, we apply the algorithm from Theorem 5.3, yielding a polynomial algorithm for the case of monotonic jobs.

Corollary 5.4. *There is a polynomial algorithm for Scheduling Monotonic Moldable Tasks on Contiguous Machines which finds a* $(5/4+\varepsilon)$*-approximation in* $n^{\mathcal{O}_\varepsilon(1)}$ *operations.*

For a more detailed overview of the work related to Strip Packing and Contiguous Moldable Task Scheduling, we refer to Section 2.2.

Methodology and Organization of this Chapter



(a) Previous reordering technique

(b) The new shifting and reordering technique

Figure 5.2. Comparison of old and new strategies in the simplified case

In the approaches seen before, i.e., in [91], [33] and [63], there arises a natural set of critical items, e.g., all items with height larger than $1/3$ OPT in [33] and [63]. The characteristic of this set is that the aspired approximation ratio is exceeded if we place one of these items on top of the packing.

The technique used in these previous approaches is heavily dependent on the fact that there can be at most two critical items on top of each other. This allows to place all critical items in the optimal packing area while

discarding some noncritical items, which are placed on top of the optimal packing later (see Figure 5.2a). If three critical items can be put on top of each other, this technique will not work. To find an algorithm with ratio $4/3 - \varepsilon$, we need to overcome this major obstacle.

To construct a $(5/4 + \varepsilon)$-approximation, we introduce a new technique to handle this difficulty, in the following called *shifting and reordering*. While we cannot guarantee that all critical items are packed in the optimal packing area, we can place the critical items with height larger than $1/4\mathrm{OPT}$ on three shelves using the area $W \times (5/4 + \varepsilon)\mathrm{OPT}$ (see Figure 5.2b).

A challenge which arises using this new strategy is the fact that some of the other items have to be discarded due to slicing. Since the packing area is already extended by the factor $(1/4 + \varepsilon)$, it is no longer possible to simply place these discarded items on top of the packing, as shown in Figure 5.2b. The discarded items have to be placed carefully into the gaps generated by the shifting and reordering technique. We prove that for each possible optimal packing the corresponding rearranged packing contains suitable gaps for these items.

In Section 5.2, we describe this shifting and reordering technique and the specific structure it generates for the simplified case that just these critical items have to be placed integrally while all other items are allowed be partitioned into vertical slices, which do not have to be placed contiguously.

For the structural result, all items have to be placed integral; thus, we cannot slice all noncritical items. Nevertheless, we may still slice certain narrow items. We use and prove the property that each optimal packing can be partitioned into few rectangular areas. A characteristic of this partition is that each critical item is contained in an area exclusively containing critical and sliceable items. Up to three critical items can overlap each of the vertical borders of these areas and these overlapping items may not be shifted horizontally or vertically. We managed to extend the new strategy to these areas although it becomes much more involved in this extension, as described in Section 5.3.

Combining our new techniques to place critical items on three shelves, find suitable gaps for discarded non-critical items and handle the exclusive slicing of narrow items together enables us to prove the structural result from Lemma 5.24. In the algorithm, we guess the structure of the

optimal packing and use dynamic programming to place the items into this structure.

The strength of the structure result is that it applies to all optimal solutions with the property that they consist of rectangular objects placed into a rectangle that is extendable on one side. Optimal solutions of the three considered problems all have this property. Thanks to this feature, we were able to obtain algorithms which find $5/4 + \varepsilon$ approximations for Strip Packing with and without rotations and for Contiguous Moldable Task Scheduling by carefully adapting the dynamic program.

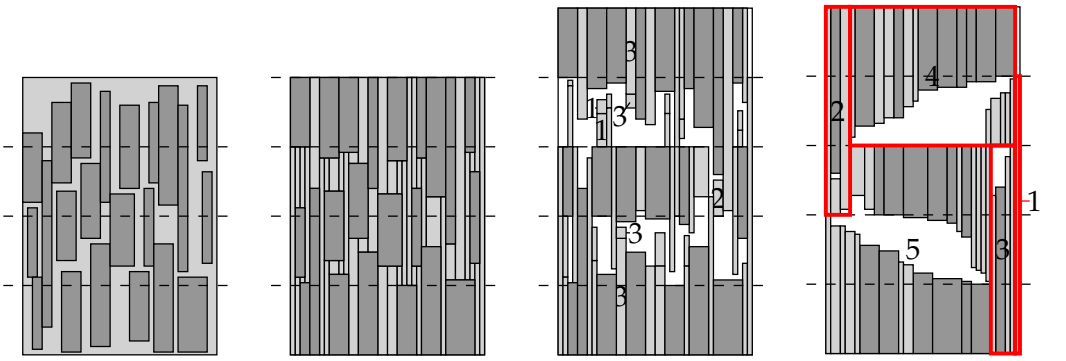
5.2 Introducing the Shifting and Reordering Technique

To demonstrate the central new idea which leads to the improved structural result – the shifting and reordering technique – we consider the following simplified case. In this scenario, we have to pack items with a tall height integrally, while we are allowed to slice all other items vertically.

Let a packing with height H be given. We define tall items as the items which have a height larger than $1/4H$ and call the others non-tall. Let us assume that there is an arithmetic grid with $N + 1$ horizontal grid lines with distance H/N such that each tall item starts and ends at the grid lines. For now, we can think of this grid as the integral grid with $H + 1$ grid lines. Later, we can reduce the grid lines by rounding the heights of the items. We are interested in a fractional packing of the non-tall items. Therefore, we replace each non-tall item i by exactly $w(i)$ items with height $h(i)$ and width 1. This step is called slicing.

Lemma 5.5. *By adding at most $1/4H$ to the packing height and slicing non-tall items, we can rearrange the items such that we generate at most $3/2N$ rectangular subareas, called boxes, which contain tall items with the same height only, and at most $9/4N + 1$ boxes for sliced items.*

Proof. In this proof, we will present a rearrangement strategy which provides the desired properties. This strategy consists of two shifting steps and one reordering step. In the shifting steps, we shift items in the vertical



(a) optimal packing **(b)** first shift **(c)** second shift **(d)** reordered packing

Figure 5.3. States of the item rearrangement. Dark rectangles represent tall items while light gray areas represent sliced items

direction, while in the reordering step we change the item positions horizontally. In the first shifting step, we ensure that tall items intersecting the horizontal lines $1/4H$ or $3/4H$ will touch the bottom or the top of the packing area, respectively. In the second shift, we ensure that tall items not intersecting these lines have a common upper border as well. Last, we reorder the items such that tall items with the same height are positioned next to each other if they have a common upper or lower border.

Step 1: First shift. Note that there is no tall item completely below $1/4H$ or completely above $3/4H$ since each tall item has a height larger than $1/4H$. We shift each tall item t intersecting the horizontal line $1/4H$ down so that its bottom border touches the bottom of the strip. The sliced items below t are shifted up exactly $h(t)$ so that they are now positioned above t. In the same way, we shift each tall item intersecting the horizontal line at $3/4H$ but not the horizontal line at $1/4H$ such that its upper border is positioned at H and shift the sliced items down accordingly, see Figure 5.3b.

Step 2: Introducing pseudo items. At this point, we introduce a set of containers for the sliced items, which we call pseudo items, see Figure 5.3b. At each left or right border of a tall item, we draw vertical lines of height H from the bottom to the top of the packing and erase these lines on any tall item. Each area between two consecutive lines which is bounded on

top and bottom by a tall item or the packing area and contains sliced items represents a new item called pseudo item. Note that no sliced item is intersecting any box border since they are positioned on integral widths only. Furthermore, when we shift a pseudo item, we shift all sliced items in this container accordingly.

When constructing the pseudo items, we consider one special case. Consider a tall item t with height larger than $3/4H$. There can be no tall item positioned above or below t, and t was shifted down. For these items, we introduce one pseudo item of height H and width $w(t)$ containing t and all sliced items above. Note that each pseudo item has a height, which is a multiple of H/N. Furthermore, note that each tall or pseudo item touching the top or the bottom border of the packing area has a height larger than $1/4H$.

Step 3: Second shift. Next, we do a second shifting step consisting of three sub-steps. First, we shift each tall or pseudo item intersected by the horizontal line at $3/4H$ but not the horizontal line at $1/4H$ exactly $1/4H$ upwards. Second, we shift each pseudo item positioned between the horizontal lines at $1/2H$ and $3/4H$ so that their lower border is positioned at the horizontal line $3/4H$. Last, we shift each tall or pseudo item intersected by the horizontal line at $1/2H$ but not the horizontal line at $1/4H$ or $3/4H$ so that its upper border is positioned at the horizontal line $3/4H$. After this shifting, no item overlaps another item since we have shifted the items intersecting the line at $3/4H$ exactly $1/4H$, while each item below is shifted at most $1/4H$.

Step 4: Fusing pseudo items. After the second shift, we will fuse and shift some pseudo items. We want to establish the property that each tall and pseudo item has one border (upper or lower), which touches one of the horizontal lines at 0, $3/4H$, or $5/4H$. At the moment there can be some pseudo items between the horizontal lines $1/4H$ and $1/2H$, which do not touch one of the three lines. In the following, we study the three cases where those pseudo items can occur. These items do only exist if there is a tall item touching the bottom of the packing and another tall item above this item with lower border at or below $1/2H$ before the second shifting step. Consider two consecutive vertical lines we had drawn to generate the pseudo items. If a tall item overlaps the vertical strip between these

lines, its right and left borders lie either on the strips borders or outside of the strip.

Case 1: In the first considered case there are three tall items, t_1, t_2, and t_3 from bottom to top, which overlap the strip. In this scenario t_1 must have its lower border at 0, t_2 its upper border at $3/4H$, and t_3 its upper border at $5/4H$. As a consequence, there are at most two pseudo items: One is positioned between t_1 and t_2, and the other one between t_2 and t_3. We will stack them so that the lower border of the stack is positioned at $3/4H$ and prove that this is possible without overlapping t_3. The total height of both pseudo items is $H - h(t_1) - h(t_2) - h(t_3)$. The total area not occupied by tall items is $H - h(t_1) - h(t_2) - h(t_3) + 1/4H$ since we have added $1/4H$ to the packing height. The distance between t_1 and t_2 is at most $1/4H$ since t_1's lower border is at 0 and t_2's upper border is at $3/4H$ and both have a height larger than $1/4H$. Therefore, the distance between t_2 and t_3 is at least $H - h(t_1) - h(t_2) - h(t_3)$, see Figure 5.3c at the items marked with 1.

Case 2: Now consider the case where there is one tall item t_1 touching the bottom, and one tall item t_2 with height at least $1/2H$ touching $5/4H$. Obviously, t_2 has a height of at most $3/4H$. Furthermore, there is at most one pseudo item, and it has to be positioned between $1/4H$ and $1/2H$. We shift this pseudo item up until its bottom touches $1/2H$, see Figure 5.3c at the item marked with 2. This is possible without constructing any overlap, because the distance between t_1 and the horizontal line $1/2H$ is less than $1/4H$ and, therefore, the distance between the line $1/2H$ and the lower border of the tall item is larger than the height of the pseudo item.

After this step, we consider each tall item t with height larger than $1/2H$ touching $5/4H$. We generate a new pseudo item with width $w(t)$ and height $3/4H$, with upper border at $5/4H$ and lower border at $1/2H$, containing all pseudo items below t touching $1/2H$ with their lower border.

Case 3: In the last case we consider, there are two tall items t_1 and t_2 and two pseudo items; one of the items t_1 and t_2 touches the top of the packing or the bottom, while the other ends at $3/4H$. Hence, the distance between the tall items has to be smaller than $1/4H$. Furthermore, one of the pseudo items has to touch the top or the bottom of the packing while the other is positioned between t_1 and t_2. Since the distance between t_1 and t_2 is less than $1/4H$ one of the distances between the packing border and the

lower border of t_1 or the upper border of t_2 is at least $H - h(t_1) - h(t_2)$. Therefore, we can fuse both pseudo items by shifting the one between t_1 and t_2 so that it is positioned above or below the other one, see Figure 5.3c at the items marked with 3.

Observation 5.6. After the shifting and fusing, each tall or pseudo item touches one of the horizontal lines at 0, $3/4H$ or $5/4H$.

Step 5: Reordering the items. In the last part of the rearrangement, we reorder the items horizontally to place pseudo and tall items with the same height next to each other. In this reordering step, we create five areas each reserved for certain items. To do so, we take vertical slices of the packing and move them to the left or the right in the strip. A vertical slice is an area of the packing with width one and height of the considered packing area, i.e. $5/4H$ in this case. While rearranging these slices, it will never happen that two items overlap. However, it can happen that some of the tall items are placed fractionally afterwards. This will be fixed in later steps.

Area 1: First, we will extract all vertical slices containing (pseudo) items with height H. Then, shifting all the remaining vertical slices to the left as much as possible, we create one box for pseudo items of height H at the right, see Figure 5.3d at Area 1. In this area, we sort the pseudo items so that the pseudo items containing tall items with the same height are placed next to each other. In this step, we did not place any tall item fractionally.

Area 2: Afterward, we take each vertical slice containing a (pseudo) item with height at least $1/2H$ touching the horizontal line at $5/4H$. Remember, there might be pseudo items containing a tall item t with height between $1/2H$ and $3/4H$. We shift these slices to the left of the packing and sort them in descending order of the tall items height $h(t)$, see Figure 5.3d at Area 2. Afterward, we sort the pseudo items below these tall items, which are touching $1/2H$ with their bottom in ascending order of their heights, which is possible without generating any overlapping. In this step, it can happen that we slice tall items which touch the bottom of the strip. We will fix this slicing in one of the following steps, when we consider Area 5.

Area 3: Next, we look at vertical slices containing (pseudo) items t with height at least $1/2H$ touching the bottom of the strip. We shift them to the

right until they touch the Area 1 and sort these slices in ascending order of the heights $h(t)$, see Figure 5.3d at Area 3. Note that there are no pseudo or tall items with upper border at $3/4H$ in these slices. In this step, it can happen that we slice tall items touching the top of the packing. This will be fixed in the next step.

Area 4: Look at the area above $3/4H$ and left of the Area 2 but right of Area 1, see Figure 5.3d at Area 4. In this area no item overlaps the horizontal line $3/4H$. Therefore, we have a rectangular area where each item either touches its bottom or its top and no item is intersected by the area's borders. In [91] it was shown that, in this case, we can sort the items touching the line $3/4H$ in ascending order of their height and the items touching $5/4H$ in descending order of heights and no item will overlap another item. Now all items with the same height are placed next to each other, thus we have fixed the slicing of tall items on the top of the strip.

Area 5: In the last step, we will reorder the remaining items, namely the items touching the bottom of the strip left of Area 3 and the items touching the horizontal line at $3/4H$ with their top between Area 2 and Area 3. The items touching the bottom are sorted in descending order of their height and the items touching the horizontal line at $3/4H$ are sorted in ascending order regarding their heights.

Claim 5.7. After the reordering of Area 5 no item overlaps another.

Proof. First, note that the items touching $5/4H$ have a height of at most $3/4H$. Therefore, no item touching the bottom having height at most $1/2H$ can overlap with these items. Furthermore, note that before the reordering no item was overlapping another. Let us assume there are two items b and t, which overlap at a point (x, y) after this reordering. Then all items left of x touching $3/4H$ have their lower border below y, while all items touching the bottom left of x have their upper border above y. Therefore, at every point left and right of (x, y) in the Area 5 there is an item overlapping it. Hence, the total width of items overlapping the horizontal line y is larger than the width of the Area 5. Therefore in the original ordering, there would have been items overlapping each other already since we did not add any items – a contradiction. As a consequence in this new ordering, no two items overlap, which concludes the proof of the claim. ◁

Analyzing the number of constructed boxes. In the last part of this proof, we analyze how many boxes we have created. Each tall item with height at least $3/4H$ touches the bottom and we create at most one box in Area 1 for each height. Therefore, we create at most $N/4$ boxes for these items. Each tall item of height between $1/2H$ and $3/4H$ touches the bottom or the horizontal line $5/4H$. On each of these lines, we create at most one box for items with the same height. Therefore, we create at most $2N/4$ boxes for these items. Last, each tall item with height larger than $1/4H$ but smaller than $1/2H$ either touches the bottom of the packing, the horizontal line $3/4H$ or the horizontal line $5/4H$. At each of these lines, we create at most one box for each height. Therefore, we create at most $3N/4$ of these boxes. In total, we create at most $\frac{3}{2}N$ boxes for tall items.

Let us consider the number of boxes for sliced items. Each pseudo item's height is a multiple of H/N. Therefore, we have at most N different sizes for pseudo items. There are at most 4 boxes for each height less than $1/4H$. One is touching H with its top border in Area 1, one is touching $3/4H$ with its bottom border in Area 4, one is touching $3/4H$ with its top border in Area 5, and one is touching $1/2H$ with its bottom border in Area 2. Furthermore, there are at most 3 boxes for each size between $1/4H$ and $1/2H$. One is touching $5/4H$ with its top border in Area 4, one is touching $3/4H$ with its top border in Area 5, and one is touching 0 with its bottom border in Area 5. Additionally, there are at most 2 boxes for each pseudo item size larger than $1/2H$. One is touching $5/4H$ with its top border in Area 2, the other one is touching 0 with its bottom border in Area 3. Last there is only one pseudo item with height larger than $3/4H$ in Area 1. It has height H. Since the grid is arithmetically, we have at most $N/4$ sizes with height at most $1/4H$, $N/4$ sizes between $1/4H$ and $1/2H$ and at most $1/4N$ sizes between $1/2H$ and $3/4H$. Therefore, we create at most $4 \cdot 1/4N + 3 \cdot 1/4N + 2 \cdot 1/4N + 1 = \frac{9}{4}N + 1$ boxes for sliced items.

□

In this section, we have proven that in this simplified case it is possible to reorder the items so that they have a nice structure, where there are at most few boxes for each tall item height containing only items with the same height. However, we are interested in a simple structure for a

packing, where no item is sliced. The main key to find such a structure is presented in the next section.

5.3 Reordering in the General Case

In the structural result, all items have to be placed integrally; thus, we cannot slice the non-tall items as we do in the previous chapter. Nevertheless, we may still slice certain narrow items, because for them we have techniques to place them integrally afterward. We call these sliceable items vertical items. In Section 5.4 we will see that it is possible to partition the packing area into a constant number of rectangular subareas, called boxes, so that boxes containing tall items will contain tall and vertical items only. In this section, we will consider such boxes and show that it is possible to reorder the items inside these boxes similarly as in Section 5.2. A challenge that arises when considering these subareas is that up to three tall items can overlap the left and right box border. Since we do not want to slice these items, we fix their position and call them unmovable. These unmovable items complicate the reordering in the box compared to Section 5.2. We overcome this difficulty by a more careful reordering of the items.

In the following, we again assume that all tall items are placed on an arithmetic grid with $N+1$ horizontal grid lines with distance H/N and that the non-tall items are sliced. Furthermore, we assume that the box also starts and ends at these grid lines. Let $S(B)$ be the y-coordinate of the lower box border. For simplicity of notation, we will assume that $S(B) = 0$. Often we will speak about horizontal lines through the box at a specific height. If $S(B) \neq 0$, one has to shift up the considered lines by $S(B)$.

The reordering of the items inside the boxes differs by the height of the boxes. In boxes with height at most $1/2H$ there can be at most one tall item cut by any vertical line through the box. The reordering inside these boxes is the simplest and is described in Section 5.3.1. If the box has a height between $1/2H$ and $3/4H$, there can be up to two tall items per vertical line through the strip and the reordering gets more involved. For the reordering of these boxes, we need an extra box of height $1/4H$ for the first time, and the algorithm is described in Section 5.3.2. The last kind of boxes are the boxes with height up to H. The reordering inside these

boxes is the most involved but uses similar techniques as in the reordering of the previous section and can be found in Section 5.3.3.

5.3.1 Reordering inside small Boxes

In this section, we consider the reordering of the items inside boxes B that have a small height of at most $h(B) \leqslant H/2$. These boxes are the most convenient boxes regarding the reordering since there can be at most one tall item per vertical line trough the strip.

Lemma 5.8. *Consider a small box B with height $h(B) \leqslant H/2$. By slicing non-tall items, we can rearrange the tall and non-tall items in this box such that there are at most $N/4+2$ sub-boxes for tall items which contain tall items with the same height only and at most $N/4+4$ sub-boxes for non-tall items.*

Proof. Since the box B has a height of at most $H/2$ and tall items have a height larger than $H/4$, each vertical line through the box will cut at most one tall item. Therefore, there can be at most one tall item of each side of the box overlapping the box border. These items are unmovable items. Since there is at most one tall item per vertical line through the box, we can shift the (movable) tall items down to the bottom of the box, similar as seen above in the proof of Lemma 5.5, since the non-tall items are sliced. Again, we draw vertical lines at the left and right borders of the tall items, to partition the area containing non-tall items into pseudo items.

After this step, the box B is partitioned into vertical strips containing no tall item, strips containing tall items, and two strips containing the unmovable items. We do not move the strips containing the unmovable items, but sort the other strips by size of the contained tall item. After this sorting step, all tall items with the same height (apart from the unmovable items) are positioned next to each other.

For each appearing height of tall items, we create at most one sub-box containing just tall items of the same height. Additionally, we create two sub-boxes containing the unmovable items. Hence, the total number of sub-boxes created for tall items is bounded by $N/4+2$ since the tall items have sizes between $H/4$ and $H/2$.

On the other hand, the number of sub-boxes created for the non-tall items is bounded by $N/4+4$: For each appearing tall item height, we

create one sub-box for non-tall items, except for the tall item height, which has height of the box. However, this loss is evened out by the box that we create for non-tall items for the case that there exist strips containing no tall item at all. Furthermore, we create 4 sub-boxes that are positioned above and below the two unmovable items, which concludes the proof of this lemma. □

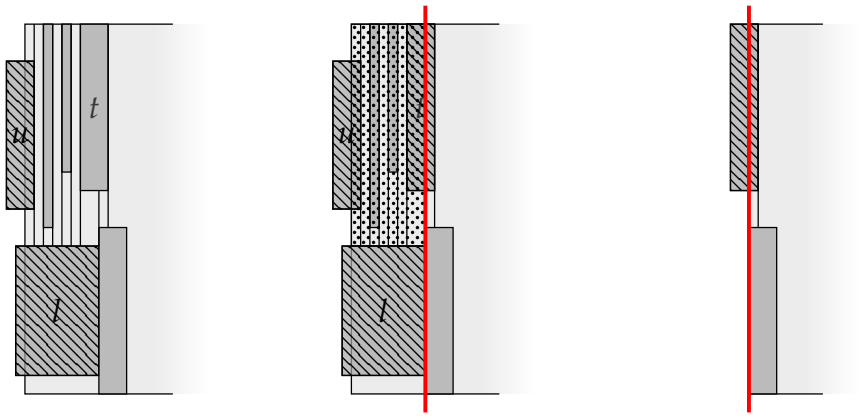
5.3.2 Reordering inside medium Boxes

In this section, we consider boxes which have a height between $1/2H$ and $3/4H$. These boxes have the property that each vertical line trough the box intersects at most two tall items. We use this feature to prove that these boxes can be divided into a constant number of sub-boxes so that each sub-box contains only items of the same size. However, to create this division, we need an additional box with a height $1/4H$ and a width which is bounded by the boxes width. More precisely, we will prove the following lemma.

Lemma 5.9. *Consider a box B with $h(B) \in (1/2H, 3/4H]$. Using an additional box B_P of height $1/4H$ and width of at most $(1-(1/N))w(B)$ it is possible to rearrange the items inside this box such that we generate at most $\mathcal{O}(1/N^2)$ sub-boxes for tall items containing only items with the same height, and at most $\mathcal{O}(1/N^2)$ sub-boxes for vertical items.*

We will prove this lemma in several steps. First, we transform the box to a box where all tall items are touching the top or the bottom of the box. To perform this transformation, we introduce a first set of sub boxes and introduce pseudo items, see Lemma 5.10. In the second step, we remove all the pseudo items that are not touching the top or the bottom of the box and shift them into an extra box of height $1/4H$ and width $w(B)$. Since the extra box that we are allowed to use has a width of at most $(1-(1/N))w(B)$, we prove that for every reordering of the tall and pseudo items it is possible to place some of the removed pseudo items back inside the box, see Lemma 5.11. In the last step, we prove that in a box where all items are touching the bottom or top of the box, we can find a reordering such that we only use a suitable number of sub-boxes for the tall and pseudo items, see Lemma 5.13.

Lemma 5.10. *Consider a box B with $h(B) \in (1/2H, 3/4H]$. By introducing at most $N/2 + 6$ sub-boxes for tall and at most $N/2 + 10$ sub-boxes for vertical items, we can generate a sub-box B' of B with height $h(B)$ and width at most $w(B)$, where all the tall items (including unmovable items) are touching the top or the bottom of the box and each side of the box has at most one overlapping unmovable item.*



(a) a left box border **(b)** define new box **(c)** define new border

Figure 5.4. A left box border, where two unmoveable items (hatched) overlap. The red line defines the new box border and the dotted area defines a new box (5.4b). The hatched area defines the new unmoveable item (5.4c).

Proof. The box B has a height of at most $3/4H$. Therefore, each vertical line trough the box intersects at most two tall items and hence there are at most two unmovable items overlapping each vertical box border. Furthermore, each tall item is intersected by the horizontal line at $H/4$ or the horizontal line at $H/2$. We shift each movable tall item intersecting $H/4$ down such that it touches the bottom of the box and each movable tall item intersecting $H/2$ up such that it touches the top of the box while we shift the sliced items up or down accordingly.

Let us consider one side of the box where two tall items l, u overlap the box border. Let l be below u. W.l.o.g. let $x_l + w(l) \geqslant x_u + w(u)$, i.e., let the right side of l be more to the right than the right side of u, see Figure 5.4a.

We draw a vertical line L through the box at the right side of l, see Figure 5.4b at the red line. We define L as the left box border of the box B'. There is at most one unmovable item overlapping this border. More precisely there can be a tall item t, positioned at the top of the box, which intersects L. This item is defined as a new unmovable item of the box B'.

We still have to deal with the items left of this line. For the item l, we introduce an new box containing just this item. For the area below this box, we introduce a new box spanning from the left lower corner of the original box to the right lower corner of the item l. This box contains just sliced items and hence is a sub-box for sliced items on its own. Last consider the area spanning from the intersection point of the top of l with the original box border and the intersection of L with the upper box border. We define this area as a new box of height at most $1/2H$. Therefore, we can use Lemma 5.8 to reorder the items in this segment and generate at most $N/4 + 2$ sub-boxes for tall and at most $N/4 + 4$ additional sub-boxes for non tall items.

We repeat this step on the right side of the box. □

In the following step, we consider a box generated by the previous lemma. More precisely this box has a height of $h(B) \in (1/2H, 3/4H]$ and each tall item either touches the top or the bottom of the box. When we introduce pseudo items for the non-tall items as seen above, each of these pseudo items either touches the top or the bottom of the box, or is contained between two tall items. In the next lemma, we are interested in these last pseudo items, which do not touch the top or bottom border.

These pseudo items interfere when we try to reorder the items touching the top or the bottom of the box. For this reason, we would like to remove these pseudo items and place them into an extra box. Since these pseudo items are always positioned between two tall items and the box B has a height of at most $h(B) \leqslant 3/4H$, these pseudo items can have a height of at most $1/4H$. Thus, they fit into the extra box B_P regarding the height. However, the total width of these items can be up to $w(B)$, but B_P has a width of less than $w(B)$. Therefore, we need to place some of these pseudo items back into the original box. The next lemma states the total width of pseudo items that can be placed back into the original box. To simplify the handling of these pseudo items, we assume that they are sliced vertically

such that each slice has a width of exactly 1.

Lemma 5.11. *Let B be a box where each tall item (movable and unmovable) either touches the bottom or the top of the box and all non-tall items are contained inside pseudo items.*

Let β be the minimal difference between the heights of two tall or pseudo items. Let t be the shortest tall or pseudo item touching the top and b be the shortest tall or pseudo item touching the bottom, with $h(t) > 0$ and $h(b) > 0$. Define $h := (h(B) - h(t) - h(b))$. Let $\mathcal{C}_B$ be the set of (sliced) pseudo items neither touching the top or the bottom of the box.

For each feasible reordering of the tall and pseudo items touching the top or bottom box border and each $\alpha \leqslant \beta/(\beta + h)$, we can find a subset $S \subseteq \mathcal{C}_B$ that can be placed in the reordered packing such that $(1-\alpha)w(B) \geqslant w(\mathcal{C}_B) - |S|$, i. e., the residual pseudo items in $\mathcal{C}_B$ can be placed inside a box width width bounded by $(1-\alpha)w(B)$.

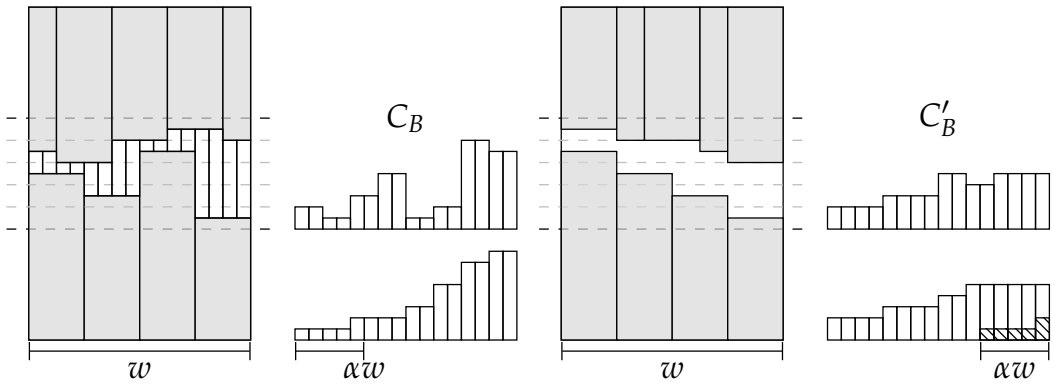


Figure 5.5. Two orderings of the items in $T \cup P$.

Proof. Let be given any reordering of the tall and pseudo items touching the top or the bottom of the box. Furthermore, let C'_B be the set of sliced pseudo items that would be generated between the tall and pseudo items in this given reordering. We sort the pseudo items in both sets C_B and C'_B in ascending order and index them from 1 to $w := w(B)$. If there are less than w pseudo items, we introduce pseudo times with height zero until there are exactly w of them. We will show that the $\lceil \alpha w \rceil$ smallest pseudo

items in C_B fit inside the $\lceil \alpha w \rceil$ largest pseudo items in C'_B. Let l be the container with index $\lceil \alpha w \rceil$ in the set C_B and let r be the container with index $w - \lceil \alpha w \rceil + 1$ in the set C'_B. If $h_l \leqslant h_r$ the $\lceil \alpha w \rceil$ shortest container in $\mathcal{C}_B$ can be placed into the $\lceil \alpha w \rceil$ longest container $\mathcal{C}'_B$, see Figure 5.5.

Assume for contradiction that $h_l > h_r$. We know about the area of the sets of containers that $A(C_B) = A(C'_B)$. Since each pseudo item in C_B with index at least l has height of at least h_l, we know that $A(C_B) \geqslant h_l(w - \lceil \alpha w \rceil + 1)$. Furthermore, we know that $A(C'_B) \leqslant h_r(w - \lceil \alpha w \rceil + 1) + h(\lceil \alpha w \rceil - 1)$ since each pseudo item $i \in C'_B$ with index at most r has height of at most h_r and each pseudo item $i \in C'_B$ with index larger than r has height at most h. In total, we have

$$h_l(w - \lceil \alpha w \rceil + 1) \leqslant A(C_B) = A(C'_B) \leqslant h_r(w - \lceil \alpha w \rceil + 1) + h(\lceil \alpha w \rceil - 1).$$

Since $h_l > h_r$ and the height difference between two items tall and pseudo is at least β, we have $h_l \geqslant h_r + \beta$. This leads to

$$(h_r + \beta)(w - \lceil \alpha w \rceil + 1) \leqslant h_r(w - \lceil \alpha w \rceil + 1) + h(\lceil \alpha w \rceil - 1).$$

It follows that $w\beta \leqslant (\beta + h)(\lceil \alpha w \rceil - 1)$. Since $\lceil \alpha w \rceil - 1 < \alpha w$, this leads to $\beta < (\beta + h)\alpha$, which is a contradiction for each $\alpha \leqslant \beta/(h + \beta)$.

□

We use this lemma for the reordering of boxes with height between $1/2H$ and $3/4H$. Since all the items touching the top or bottom have a height of at least $1/4H$, we know that $h \leqslant 1/4H$. Furthermore, we know that all the tall and pseudo items have a height, which is a multiple of H/N since the box starts and ends at grid lines and so do the tall items. Therefore, we know that $\beta \geqslant H/N$. As a result, we can choose $\alpha \geqslant (H/N)/((H/N) + 1/4H) = (1/N)/(1/N + 1/4) \geqslant 1/N$.

Corollary 5.12. *The box we use to place the pseudo items that do not touch the top or the bottom of the box needs a width of at most* $(1 - 1/N)w(B)$.

In the last step, we analyze how we can rearrange the items touching the top or the bottom of the considered box while generating a constant number of sub-boxes for tall and pseudo items. Note that the properties needed in the following lemma are provided by the previous lemmas. The number of different item heights appearing in the box is bounded by $\mathcal{O}(1/N)$.

Lemma 5.13. *Consider a box B, where all tall and pseudo items (including the unmovable items) either touch the top or the bottom of the box and there is at most one unmovable item overlapping each vertical border of the box. Let k be the number of different appearing heights of the items, which are all multiples of H/N. We can reorder the items in this box such that we need to introduce at most $\mathcal{O}(k^2)$ sub-boxes for tall items and at most $\mathcal{O}(k^2)$ sub-boxes for non-tall items*

Proof. Let us assume that on each part of the upper or lower border there is a pseudo or tall item touching this border. If not, we introduce a new item with height zero.

Let $h_{b,0}$ be the height of a tallest item touching the bottom of the box and $i_{b,l}$ be the leftmost and $i_{b,r}$ the rightmost item of height $h_{b,0}$. Similarly choose $h_{t,0}$, $i_{t,l}$, and $i_{t,r}$ with respect to the top of the box. Further, let i_l be the item in $\{i_{b,l}, i_{t,l}\}$ which is most to the left and i_r the item which is most to the right in $\{i_{b,r}, i_{t,r}\}$. If i_l and i_r are touching the same border we change i_r to the second right most item in $\{i_{b,r}, i_{t,r}\}$. Let w.l.o.g. $i_{l,0} = i_{b,l}$ and $i_{r,0} = i_{t,r}$.

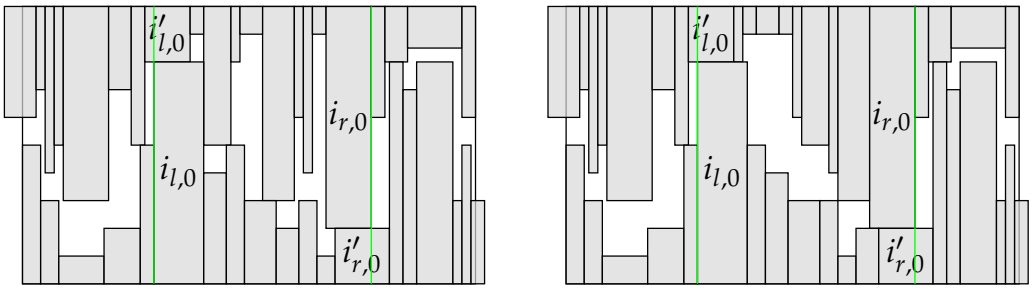


Figure 5.6. A packing before and after the first step of the reordering.

We draw a vertical line at the left border of $i_{l,0}$. The item we cut with this line becomes a new unmovable item $i'_{l,0}$. We do the same on the right side of $i_{r,0}$ and name the cut item $i'_{r,0}$ (see Figure 5.6). Next, we sort the movable items between the drawn vertical lines. The movable items touching the top are sorted in ascending order with respect to their height, while the movable items touching the bottom are sorted in descending order.

Claim 5.14. After this reordering no two items overlap.

Proof. There is no tall item touching the bottom that overlaps $i'_{l,0}$ since each item touching the bottom has height at most $h_{b,0}$. Since $i'_{l,0}$ was placed above $i_{l,0}$, $i'_{l,0}$ fits above each item in the box B. Analogously one can show that no item overlaps $i'_{r,0}$.

Assume now there is an item i_b touching the bottom that overlaps an item i_t touching the top. Let $p = (x_p, y_p)$ be a point, which is overlapped by the item i_b and i_t. Let (x_l, y_b) denote the left bottom corner of b_l and (x_r, y_t) the right top corner of t_l. By our reordering there must be a set of items I_b touching the bottom with total width greater than $x_p - x_l$, which is placed between x_l and x_r and has height at least $y_p - y_b$. Furthermore, there must be a set of items I_t with total width greater than $x_r - x_p$ touching the bottom and having height at least $y_t - y_p$. Since the area the items can be placed in has a width of $x_r - x_l$ and the sets I_b and I_t have a total width of $w(I_t \cup I_b) > x_p - x_l + x_r - x_p = x_r - x_l$ by the pidgin hole principle there must be an item in I_b that overlaps an item in I_t in the original packing – a contradiction, which concludes the proof of the claim. ◁

We now look at the items touching the top and having the same height as $i'_{l,0}$. We remove this set of items, shift the items smaller than $h(i'_{l,0})$ to the right and place the items with height $h(i'_{l,0})$ next to $i'_{l,0}$. After this shifting no two items overlap since all the items we shifted are smaller than $i'_{l,0}$ and hence they cannot overlap an item from the bottom of the box. By this shifting, we avoid to introduce an extra box for the item $i'_{l,0}$. We do the same on the bottom with the items with height $h(i'_{r,0})$.

So far, we have achieved the following: We have at most $2k$ boxes for tall items between i_l and i_r and at most $2k$ boxes for pseudo items. On the left of $i_{l,0}$, we have reduced the number of different item heights touching the bottom by at least one as they are all smaller than $h(i_{l,0})$. Likewise, on the right side of $i_{r,0}$, the number of distinct item heights touching the top is reduced by one.

We now describe how to continue to reorder the packing: We repeat the following step on the right side of $i_{r,0}$ and on the left side of $i_{l,0}$ until a break condition occurs. Since the steps are mirrored for $i_{l,0}$ and $i_{r,0}$, we only describe them for $i_{l,0}$.

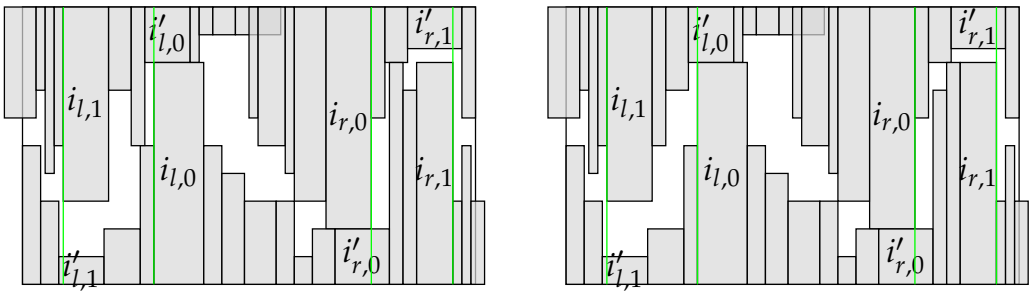


Figure 5.7. First iteration of the iterative rearrangement of the tall and pseudo items.

Let us assume that we are in the jth iteration. We look on the left side of $i_{l,j}$. W.l.o.g. let $i_{l,j}$ touching the bottom of the box. Let $i'_{l,j}$ be the item, which was intersected by the vertical line at the left border of $i_{l,j}$. Let $h_{t,j+1}$ be the height of the largest item left of $i_{l,j}$ touching the other side than $i_{l,j}$, i. e., the top in this case since $i_{l,j}$ touches the bottom. We define $i_{l,j+1}$ as the left most item touching the top, which has height $h_{t,j+1}$. Again, we draw a vertical line on the left side of $i_{l,j+1}$. Let $i'_{l,j+1}$ be the item intersected by this line. We define $i'_{l,j}$ and $i'_{l,j+1}$ as unmovable items. We sort the movable items touching the bottom between $i'_{l,j+1}$ and $i_{l,j}$ in ascending order and the movable items touching the top in descending order. With the same arguments in the proof of Claim 5.14, one can see that by this reordering no item from the bottom overlaps an item from the top. By choosing $i'_{l,j+1}$ as the leftmost tallest item touching the top, we have reduced the total number of different heights touching the top by at least one.

We repeat the described step until one of the following conditions occur:

1. The tallest item touching the top and the tallest item touching the bottom have a summed height of at most $h(B)$.

2. The item $i_{l,j}$ touches or overlaps the left border of the box.

If condition 1 occurs in any reordering of the items, it cannot happen that a tall or pseudo item touching the bottom overlaps any tall or pseudo item touching the top since their height is not large enough. So at this

point, we simply sort the items touching the top in ascending order and the items touching the bottom just as well.

If condition 2 occurs, we repeat the normal reordering step once again. When we draw the vertical line, it will be placed exactly on the box border, and we are finished.

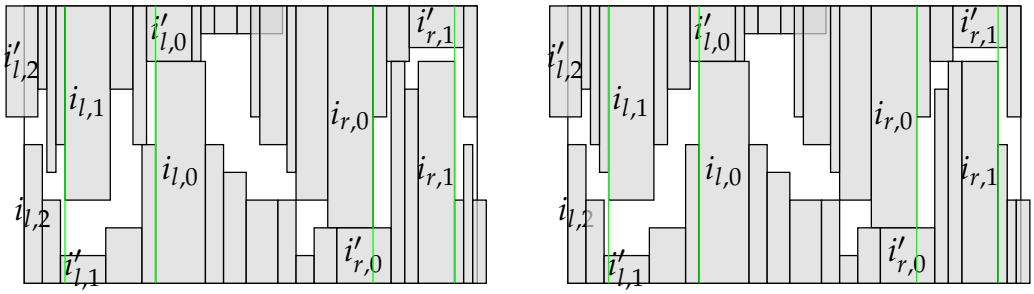


Figure 5.8. On the left side break condition 2 occurs, while on the right side break condition 1 occurs.

Let us count how many different sub-boxes for tall and pseudo items we create. In each of the partitioning steps, we create at most $2k$ sub-boxes for tall items and at most $2k$ sub-boxes for pseudo items, i. e., we create at most one sub-box at the top and one sub-box at the bottom of the box for each size and item type. Furthermore, in each of these steps, we reduce the total number of different heights touching the bottom or touching the top by one. If the tallest item touching the top and the tallest item touching the bottom are both smaller than $h(B)/2$, then condition 1 is fulfilled. Hence, we need at most $\mathcal{O}(k)$ iterations until the tallest item touching the bottom and the tallest item touching the top both have a height of at most $h(B)/2$ since all sizes are multiples of H/N. Hence in total, we create at most $\mathcal{O}(k^2)$ sub-boxes for tall items and at most $\mathcal{O}(k^2)$ sub-boxes for pseudo items. □

When performing the steps of the Lemmas 5.10, 5.11, and 5.13 one after another, we obtain a reordering, which fulfills the properties of Lemma 5.9.

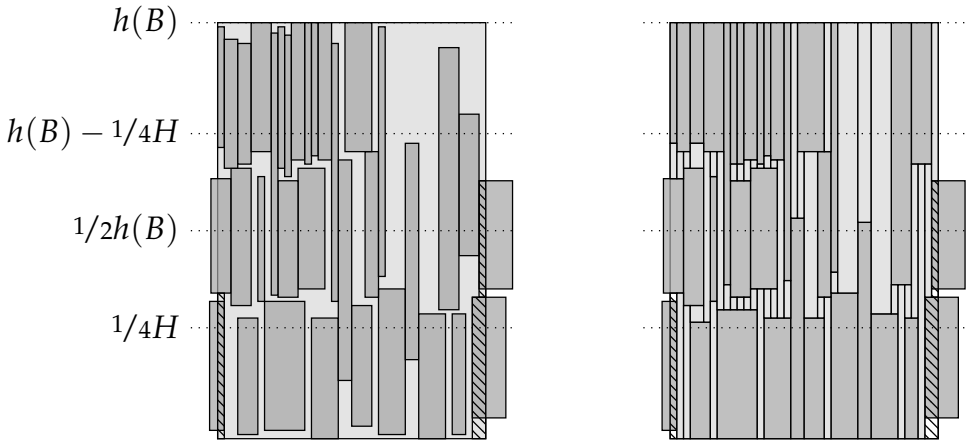
5.3.3 Reordering inside tall Boxes

In the final step, we consider tall boxes B with height $h(B) > 3/4H$. The reordering inside these boxes is similar to the reordering described in Section 5.2. However, we have to be much more careful since there could be up to three tall items overlapping the box borders. To simplify the analysis of the reordering, we assume that the lower border of B is at 0. If not, we shift all horizontal lines accordingly. Furthermore, we will assume that no tall item overlaps the left or right box border at (or above) $S(B) + h(B) - 1/4H$. In the proof of the Structure Lemma, we will establish this feature before applying the following lemma.

Lemma 5.15. *Let B be a box with height $h(B) > 3/4H$ such that no tall item overlaps the left or right box border at (or above) $h(B) - 1/4H$. By adding at most $1/4H$ to B's height, we can rearrange the items in B such that we generate at most $\mathcal{O}(N^2)$ boxes for tall and at most $\mathcal{O}(N^2)$ boxes for vertical items without moving the unmovable items. The vertical items are sliced while each tall item is placed as a whole.*

Proof. In this proof, we present a reordering strategy for the items in these boxes. Let $h(B)$ be the height of B. Notice that there are at most two tall items overlapping the left or right box border since we assumed that there is no tall item overlapping the border at $h(B) - 1/4H$. In the first step, we shift all movable items according to the first and second shifting step seen in the proof of Lemma 5.5. However, the reordering works differently than before. If there are items taller than $1/2H$ touching the top of the box, we find the leftmost item i_l and the rightmost item r of them. We introduce three areas: one left of i_l, one between i_l and i_r and one right of i_r. While we reorder the leftmost and the rightmost area with known techniques, we need a new trick to reorder the middle part.

Step 1: Shifting the items. Let us first consider the unmovable items on the left box side. There can be two of these, one overlapping the box at $1/4H$, the other at $1/2h(B)$. In the case that there is just one item, we extend it to the bottom of the strip, generating one unmovable pseudo item. If there are two items, t_1 at $1/4H$ and t_2 at $1/2h(B)$, we extend t_1 to the bottom. Then, depending on which of the items t_1 or t_2 has its right border farther on the left, we extend t_1 to the bottom of t_2 or t_2 to the top of t_1, i.e., the



(a) The introduction of unmovable pseudo items (hatched area).

(b) The first shift and the introduction of pseudo items.

Figure 5.9. Shifting the items

one whose appearance inside the box is more narrow is extended, see Figure 5.9a at the hatched areas. We do the same on the right side of the box.

Next, we perform the first shifting step, which works analogue to the one in the simple case with exception for the unmovable items which will not be shifted, see Figure 5.9b. First, we shift each movable item crossed by the line $1/4H$ down to the bottom of the box. Afterward, we shift each movable item crossed by the line $h(B) - 1/4H$ to the top of the box. We introduce pseudo items as described in the proof of Lemma 5.5, with the difference that each tall item t with height larger than $3h(B)/4$ generates a pseudo item with height $h(B)$ and width $w(t)$.

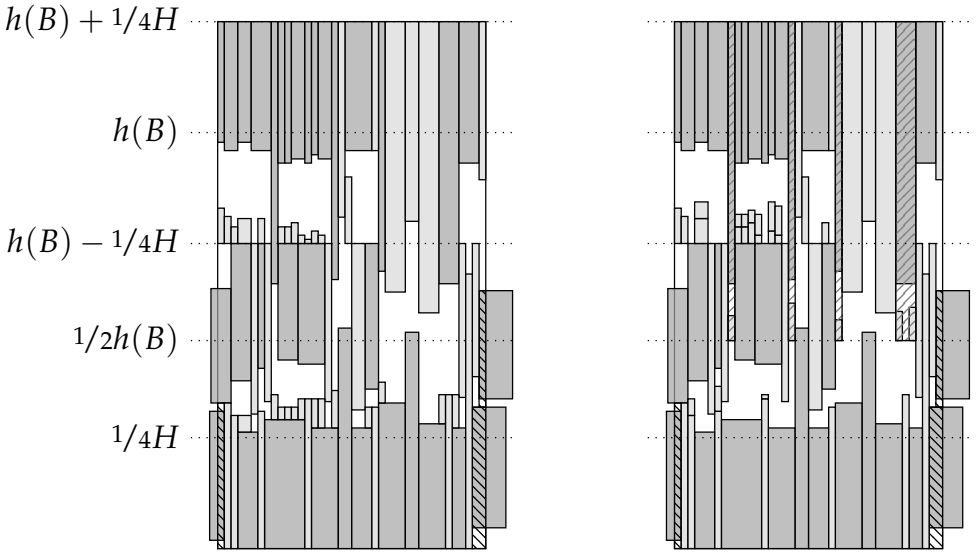
Next, we do the second shifting step, see Figure 5.10a. Each tall and pseudo item cut by the line $h(B) - 1/4H$ is shifted up exactly $1/4H$. Remember, there is no unmovable item intersecting this line. We shift each pseudo item between the lines $h(B) - 1/4H$ and $1/2h(B)$ such that its bottom touches $h(B) - 1/4H$. Afterward, we shift each not shifted movable tall and pseudo item crossed by the line $1/2h(B)$ such that its top touches $h(B) - 1/4H$. Again, no item overlaps another after this shift.

Last, we will fuse the pseudo items as described in Lemma 5.5, see

Figure 5.10b. The fusion is possible since the considered distance in each of the Cases 1 to 3 is at most $1/4H$, too. After this fusion, we can assume that each item t with height larger than $1/2h(B)$ touching $h(B) + 1/4H$ has a height of exactly $1/2h(B) + 1/4H$, see Case 2 in the proof of Lemma 5.5.

Furthermore, we can assume that each item t touching the bottom with height taller than $1/2h(B)$ has height $h(B) - 1/4H$: There can be at most two items above t, one tall item and one pseudo item. The pseudo item has its lower border at $h(B) - 1/4H$. Therefore, we can extend the item t to the horizontal line $h(B) - 1/4H$.

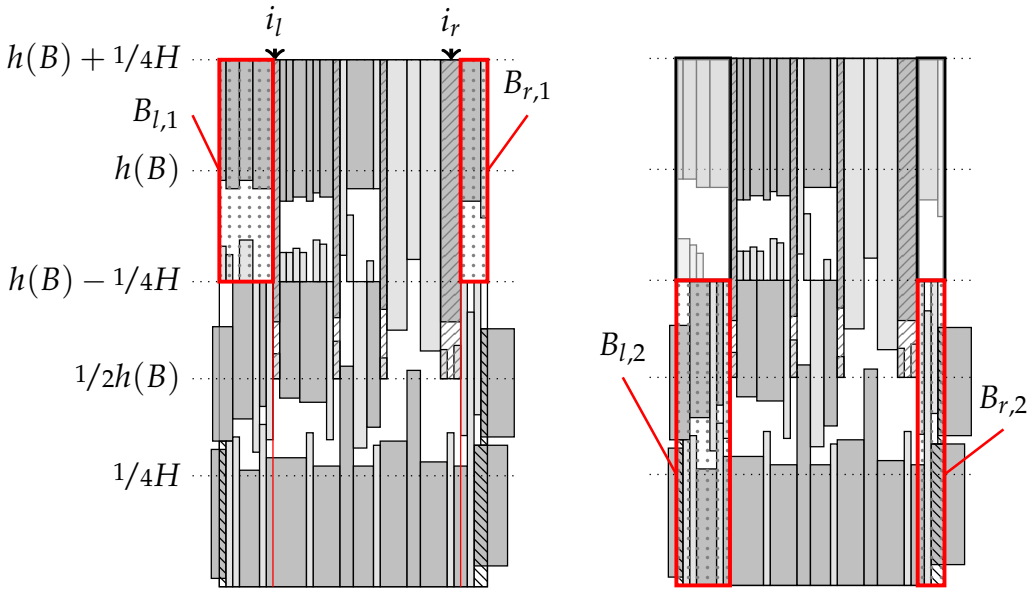
After this shift, each movable item has one border at one of the following horizontal lines 0, $h(B) - 1/4H$, or $h(B) + 1/4H$. Furthermore, only (pseudo) items with height $1/2h(B) + 1/4H$ or larger are crossing the line $h(B) - 1/4H$.



(a) The second shift. **(b)** Fusing the pseudo items.

Figure 5.10. Shifting the items

Step 2: Reordering. Let us assume for simplicity that there is no (pseudo) item with height $h(B)$ in B. Later we will see what happens if there are any of these ones. In the following, we will reorder the items step by step by considering a constant number of smaller subareas of the box. We



(a) The area $B_{l,1}$ and $B_{r,1}$.

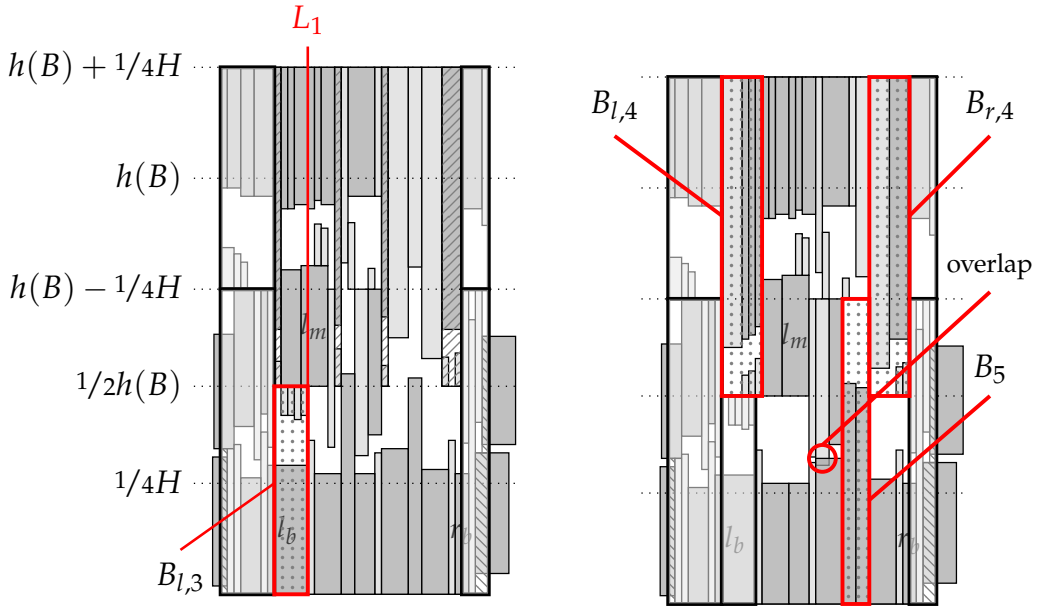
(b) The areas $B_{l,2}$ and $B_{r,2}$.

Figure 5.11. Reordering the items

number these subareas from one to nine. These subareas are generated symmetrically on the left and the right side of the box and we call them $B_{l,i}$ and $B_{r,i}$ accordingly for the ith subarea. In the following, we will describe the steps only for the boxes $B_{l,i}$.

Area $B_{l,1}$: Consider the leftmost (pseudo) item i_l with height $1/2h(B) + 1/4H$ touching $h(B) + 1/4H$ and let i_r be the right most of these items. Left of i_l inside the box B, there is no item intersecting the horizontal line $h(B) - 1/4H$ since only (pseudo) items with height $1/2h(B) + 1/4H$ touching $h(B) + 1/4H$ overlap this horizontal line, see Figure 5.11a. Therefore, each item left of i_l above $h(B) - 1/4H$ either touches $h(B) - 1/4H$ with its lower border or $h(B) + 1/4H$ with its upper border. Since there is no item intersecting the left box border, we can sort the items left of i_l touching $h(B) + 1/4H$ in descending order and the items touching $h(B) - 1/4H$ in ascending order of their heights, without constructing any overlap. The same holds for the right side of i_r. We call these areas $B_{l,1}$ and $B_{r,1}$.

Area $B_{l,2}$: We draw a vertical line at the left border of i_l to the bottom



(a) The shift at the left of L_1 and the introduction of area $B_{l,3}$.

(b) The areas $B_{l,4}$, $B_{r,4}$, and B_5.

Figure 5.12. Reordering the items

of the box, see Figure 5.11b. If this line cuts a tall item l_b at the bottom, it defines a new unmovable item. Let us consider the area between the line and the left box border below $h(B) - 1/4H$. We call this area $B_{l,2}$. In $B_{l,2}$ each item either touches the horizontal line at 0 or $h(B) - 1/4H$ and on each side there are at most two tall unmovable items. We extend the unmovable item intersecting $h(B)/2$ on the top such that it touches the horizontal line at $h(B) - 1/4H$ and reorder this box with the techniques from Lemma 5.13. We do the the same on the right of i_r.

Cases for i_l and i_r: If i_l and i_r do not exist, there are no items overlapping the horizontal line $h(B) - H/4$ and we can partition the box in two areas B_1 and B_2. We reorder B_1 as described for $B_{l,1}$ and B_2 as described for $B_{l,2}$. In the case that i_l equals i_r, we introduce $B_{l,1}$, $B_{l,2}$, $B_{r,1}$ and $B_{r,2}$ as described, and order the tall items completely below i_l such that items with the same height are positioned next to each other.

Area $B_{l,3}$ Now, we look at the area between the left border of i_l and the

right border of i_r. We denote by $r(i)$ the right border of an item i. If $r(l_b)$ is to the right of $r(i_l)$, we draw a vertical line at $r(l_b)$, called L_1. If L_1 intersects a tall item with upper border at $h(B) - 1/4H$, we call this item l_m. Left of L_1 and right of i_l, we shift up each item touching $h(B) - 1/4H$ with its top (the item l_m inclusively) such that its lower border touches $1/2h(B)$, and shift down each pseudo item touching $h(B) - 1/4H$ with its lower border such that it touches $1/2h(B)$ with its upper border. All pseudo items right of L_1 above l_m are shifted such that they touch the top of l_m with their bottom, see Figure 5.12a. Note that no pseudo item is intersected by the line L_1.

Claim 5.16. After this shift no item overlaps another.

Proof. Consider an item i that was shifted up such that it starts at $1/2h(B)$. Note that the distance between the upper border of l_b and $1/2h(B)$ is less than $1/4H$ because the upper border of l_b is above $1/4H$. Hence there has to be some free space left between the upper border of s and the lower border of each item above since we added $1/4H$ to the packing height.

Now consider an item i' that was down shifted such that it ends at $1/2h(B)$. Above this item there has to be a tall item i'' starting at $1/2h(B)$, which has a height larger than $1/4H$. The item i''' above i'', i. e., an item ending at $h(B) + 1/4H$ has a height larger than $1/4H$ as well since all items ending at $h(B) + 1/4H$ have at least this height. Therefore, the vertical distance between i'' and i''' is smaller than $1/4H$. Since we have added $1/4H$ to the packing height, the vertical distance between the bottom of i' and the top of l_b has to be larger than zero. This concludes the proof of this claim that no item overlaps another after the described shift. ◁

Let $I_{l,1/2h(B)}$ be the set of shifted items now touching $1/2h(B)$ with their bottom. All the items in $I_{l,1/2h(B)}$ have a height of at most $1/2h(B)$. The area left of L_1 and right of the left border of i_l below $1/2h(B)$ is called $B_{l,3}$. This area contains pseudo items touching $1/2h(B)$ and a part of l_b at the bottom. We sort the pseudo items above l_b touching $1/2h(B)$ in descending order of their heights.

On the other hand, if $r(l_b)$ is left of $r(i_l)$, we introduce the line L_1, but do not shift any item. On the right of i_r, we introduce the same line and area named R_1 and $B_{r,3}$ respectively.

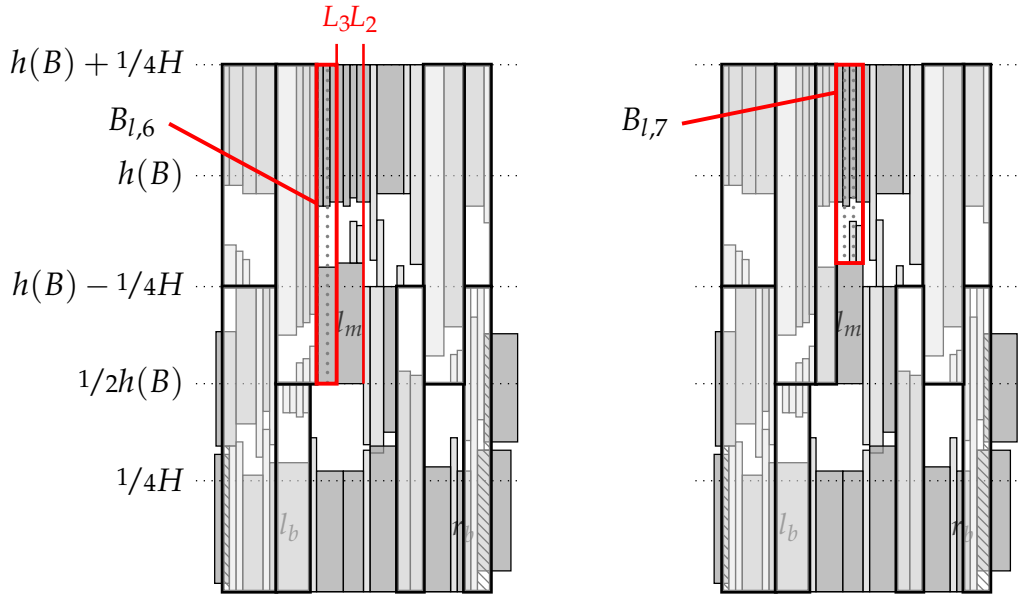
Simple cases. It is possible that l_b equals r_b, or one of the lines L_1 or R_1 intersects with i_r or i_l respectively, or that L_1 equals R_1. In each of these cases, there is no item with height larger than $h(B)/2$ touching the bottom of the box between the lines L_1 and R_1. If there is no such item, we shift all the items touching $h(B) - 1/4H$ with their top between L_1 and R_1 such that they touch $1/2h(B)$ with their bottoms and the pseudo items touching $h(B) - 1/4H$ with their bottom such that they touch $1/2h(B)$ with their top (similar as we did with the items above l_b). Now there is no item intersecting the horizontal line $1/2h(B)$. Hence, we can sort the items above $1/2h(B)$ between i_l and i_r by their heights as well as the items below $1/2h(B)$. After this step, we do not need any further reordering.

Area $B_{l,4}$ and Area B_5: We now consider the case that there is an item with height taller than $1/2h(B)$ at the bottom between i_l and i_r and, hence, we need further reordering. The objective is to reorder the items of height $1/2h(B) + 1/4H$ touching $h(B) + 1/4H$ such that they build two blocks, one next to i_l and one next to i_r. These blocks will be areas $B_{l,4}$ and $B_{r,4}$. To make this reordering possible, we have to define a border between i_l and i_r such that all these items left of this border are shifted to the item i_l while all these items right of this border are shifted to the item i_r. Let i be an item of height larger than $1/2h(B)$ touching the bottom between L_1 and R_1. This item defines the border between i_l and i_r.

Consider items with height $1/2h(B) + 1/4H$ touching $h(B) + 1/4H$, right of i_l and left of i. Note that none of these items is positioned above i. We shift those items left of i to the left until they touch i_l and those items right of i to the right until they touch i_r. All other items with parts above $1/2h(B)$ are shifted to the right or left accordingly, see Figure 5.12b. We sort the items with height $1/2h(B) + 1/4H$ such that the pseudo items containing tall items with an equal height are positioned next to each other. The area containing these items left of i (i_l inclusively) is called $B_{l,4}$.

While we shift the items with height $1/2h(B) + 1/4H$ touching $h(B) + 1/4H$ such that they are close to i_l and i_r, we shift all the items between i_l and i_r with height $h(B) - H/4$ touching the horizontal line at 0 such that they are next to i and shift the other items to the left or right accordingly. These items form a new area around i called B_5.

In this step of creating the areas $B_{l,4}$, $B_{r,4}$, and B_5, it can happen that



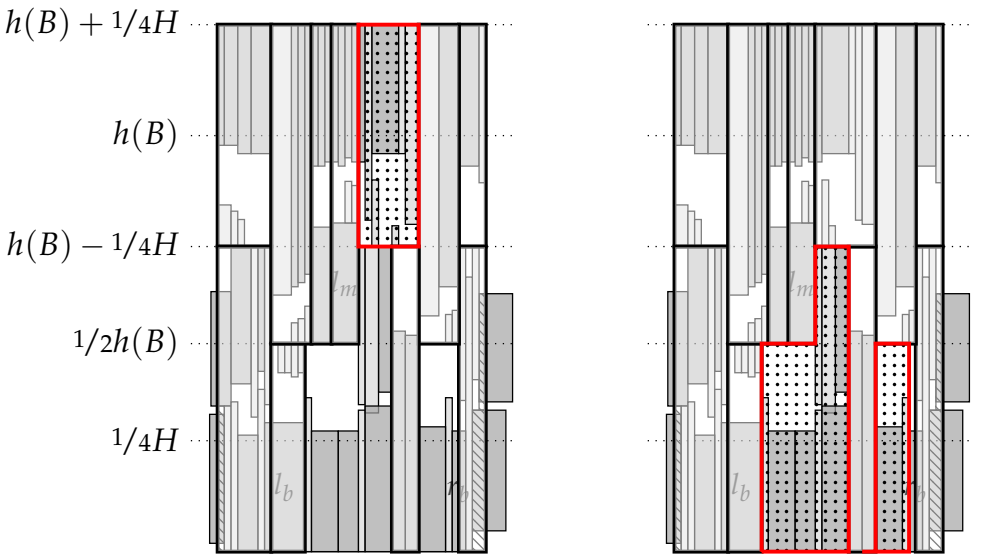
(a) The area $B_{l,6}$. **(b)** The area $B_{l,7}$.

Figure 5.13. Reordering the items

items touching $h(B) - 1/4H$ with their top are intersecting items touching 0 with their bottom, see Figure 5.12b. We will fix this in a later step, when we consider area $B_{l,9}$ and $B_{r,9}$.

Area $B_{l,6}$: Note that the items in the set $I_{l,1/2h(B)}$ are now placed next to each other (before it was possible that items with height $1/2h(B) + 1/4H$ where positioned between them). In addition, there is no item touching $h(B) + 1/4H$ above an item touching $h(B) - 1/4H$ with their bottom, which was not above this item before. Furthermore, the total width of items with bottom border above $1/4H$ and below $1/2h(B)$ between L_1 and the right of i has not changed.

If l_m exists, we draw a vertical line L_2 at the right of l_m and a vertical line L_3 at the left of l_m, see Figure 5.13a. Let $l_{t,r}$ and $l_{t,l}$ be the tall items touching $h(B) + 1/4H$ intersected by this line if there are any. We look at the area left of L_3 and right of $B_{l,4}$, which is bounded at the top by $h(B) + 1/4H$ and at the bottom by $1/2h(B)$. We call this area $B_{l,6}$. In this area, each item touches the bottom or the top, and there is at most one



(a) The area B_8. **(b)** The areas $B_{l,9}$ and $B_{r,9}$.

Figure 5.14. Reordering the items

item $l_{t,l}$ intersecting the border, see Figure 5.13a. We use the reordering in Lemma 5.13 to reorder the items in $B_{l,6}$.

Area $B_{l,7}$: The area above l_m is called $B_{l,7}$, see Figure 5.13b. In this area, all items are touching $h(B) + 1/4H$ or the top of l_m. All the items touching l_m with their bottom (and not $h(B) + 1/4H$ with their top) are pseudo items. We order the items touching $h(B) + 1/4H$ in ascending order of their heights and move the pseudo items below with them. Now, we look at the overlapping items $l_{t,r}$ and $l_{t,l}$. We move items with the height $h(l_{t,r})$ and $h(l_{t,l})$ next to these overlapping items. This generates three areas for pseudo items. The first is positioned below the first overlapping item together with the items with the same height, the second below the other overlapping item together with the items with the same height, and the last between these areas. In each of these areas, we sort the pseudo items in descending order of their height.

The areas $B_{l,6}$ and $B_{l,7}$ exist only if l_m exists. If l_m does not exist, we introduce the vertical line L_2 at the left border of the area $B_{l,4}$. We introduce R_2 and the areas $B_{r,6}$ and $B_{r,7}$ analogously on the left of i_r.

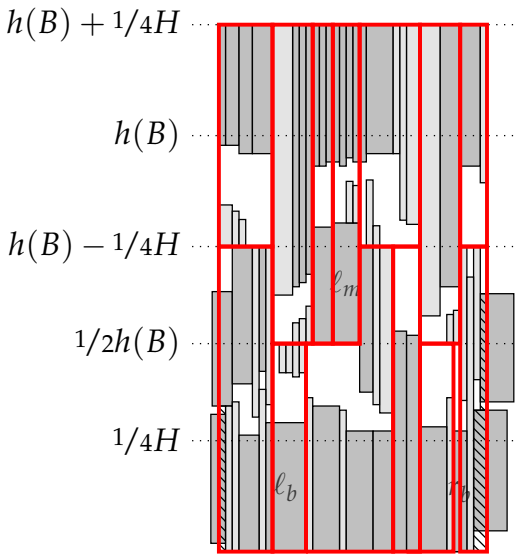


Figure 5.15. Overview of the reordered packing

Area B_8: Look at the area above $h(B) - 1/4H$ right of L_2 and left of R_2, see Figure 5.14. We call this area B_8. There are at most two unmovable items overlapping this area. One item $l_{t,r}$ on the left touching $h(B) + 1/4H$ and one item $r_{t,l}$ on the right touching $h(B) + 1/4H$. Since B_8 does not contain any item of height $1/2h(B) + 1/4H$ or items from the sets $I_{l,1/2h(B)}$ or $I_{r,1/2h(B)}$, each item touches either the top or the bottom of this area. Furthermore, all items touching the bottom are pseudo items. Therefore, we can sort the items in this area as they are sorted in area $B_{l,7}$.

Area $B_{l,9}$: Last, we have to look at the items on the bottom between L_1 and R_1 as well as at the items touching $h(B) - 1/4H$ with their top between L_2 and R_2, see Figure 5.14. We consider the items touching the bottom between L_1 and the left border of B_5 and the items touching $h(B) - 1/4H$ with their top between L_2 and the left border of B_5. The area containing these items is called $B_{l,9}$. In $B_{l,9}$, we sort all items touching $h(B) - 1/4H$ in ascending order of their heights and the items at the bottom in descending order of their heights such that the tallest on the bottom touches l_b and the smallest touches the area B_5. We do the same but mirrored on the right side of i in the area $B_{r,9}$.

Claim 5.17. After this step there is no item which overlaps another in the

area $B_{l,9}$.

Proof. First, no item from the bottom will overlap the items with height $1/2h(B) + 1/4H$ from the top since their lower border is at $1/2h(B)$ and the items below have a height less than $1/2h(B)$ (otherwise they would be contained in area B_5).

Let us assume there is an item b from the bottom intersecting an item t from the top at an inner point (x, y) in the area $B_{l,9}$. As a consequence, for each x' larger than x up to the left border of B_5, the point (x', y) is overlapped by an item touching $h(B) - 1/4H$. On the other hand, for each $x' \leqslant x$ but right of L_1 (i.e., $x' \geqslant L_i$) the point (x', y) is overlapped by an item from the bottom of the box. Note that the total width of items with lower border below y and above $1/4H$ between L_1 and the left border of i has not changed after the shifting of items with height $1/2h(B) + 1/4H$ on the top of the box (the items left of l_m have their lower border at $h(B)/2$). Additionally, the total width of items touching the bottom of the box with upper border above y in this area has not changed either. Therefore, the total width of items overlapping the horizontal line at y in this area is larger than the width of this area. As a result the items must have had an overlapping before the first horizontal shift – a contradiction. Hence, there is no item overlapping another item in this area, which concludes the proof of the claim. ◁

Items with height $h(B)$: Last, let us consider the case that (pseudo) items with height $h(B)$ exist in B. In this case in the very first step, we choose one of the items with height $h(B)$ and shift all the other items with this height to the left or to the right such that they are positioned next to this item. This shifting can be done without slicing any tall item. Afterward these items form an area B_{10} which just contains items of height $h(B)$. We sort those pseudo items of height H which contain tall items so that tall items with the same height are placed next to each other. Then we will search for i_l left of this area and for i_r right of this area. This area divides the box and represents the splitting item i. In this case the box B_8 is split into two parts $B_{l,8}$ and $B_{r,8}$, as well as the area B_5 is divided into $B_{l,5}$ and $B_r, 5$. All the following steps are done as described above.

Analyzing the number of constructed boxes. In the worst case possible there

are (pseudo) items with height $h(B)$ in B and both i_l and i_r exist. Furthermore, the left border of l_b should be right of the left border of i_l as well as the left border of r_b should be left of i_r. Until now we did not need the assumption that the tall items are placed on an arithmetic grid. However, to count the generated boxes, it is convenient to make this assumption. First, we will analyze the number of boxes for tall items we generate.

Claim 5.18. The number of boxes for tall items is bounded by $2N^2 + 15N/4 + 8$, where H/N is the distance between the grid lines.

Proof. We proof this claim by considering the generated areas one after another and count the number of boxes generated in each of these areas.

Area $B_{l,1}$: $N/4$ boxes. In the areas $B_{l,1}$ and $B_{r,1}$ there are tall items with heights between $1/4H$ and $1/2H$ on the top of the box. For each of these sizes we generate at most one box in each area. Therefore, both contain at most $N/4$ boxes for tall items.

Area $B_{l,2}$: $N^2/2 + N/4 + 3$ boxes. In the boxes $B_{l,2}$ and $B_{r,2}$, we create at most one box for each item height larger than $1/2h(B)$ and lower than $h(B)/3$. There are at most $N/4$ sizes. For the other occurring sizes we create by Lemma 5.13 at most $4S_TS_{T\cup P} + 3$ boxes in total since there are at most three unmovable items overlapping this area. We have $S_T \leqslant N/4$ since the tall items have heights between $1/4H$ and $1/2h(B)$, $S_P \leqslant N/2$ since they have heights smaller than $1/2h(B)$, and $S_{T\cup P} \leqslant N/2$ as a consequence. Therefore, we create at most $4\frac{N}{4}\frac{N}{2} + \frac{N}{4} + 3 = N^2/2 + N/4 + 3$ boxes for tall items in each of the areas $B_{l,2}$ and $B_{r,2}$.

Area $B_{l,3}$: 0 boxes. The area $B_{l,3}$ just contains the item l_b as a tall item. Since this item overlaps the area $B_{l,2}$, we have already counted this item.

Area $B_{l,4}$: $N/4$ boxes. The area $B_{l,4}$ contains just tall items with height between $1/2h(B)$ and $h(B) - 1/4H$. For each size we create one box. Therefore, we create at most $N/4$ boxes for tall items in this area.

Area $B_{l,5}$: $N/4$ boxes. The area $B_{l,5}$ contains the tall items with height between $1/2h(B)$ and $h(B) - 1/4H$. For each of these sizes we create at most one box, resulting in at most $N/4$ boxes in this area.

Area $B_{l,5}$: $N^2/2 + 1$ boxes. In the areas $B_{l,6}$ and $B_{r,6}$ each tall and pseudo item has a size of less than $1/2h(B)$. Analogously to the boxes $B_{l,2}$ and $B_{r,2}$,

we create by Lemma 5.13 at most $N^2/2 + 1$ boxes for tall items per area $B_{l,3}$ and $B_{r,3}$ since there is at most one overlapping item.

Area $B_{l,6}$: $N/4$ boxes. The area $B_{l,7}$ or $B_{r,7}$ is the area containing l_m or r_m respectively. Above l_m and r_m we create at most $N/4$ boxes for tall items each since the tall items have a height of at most $1/2h(B)$ and at least $1/4H$. The box for the item overlapping L_3 is already counted.

Area B_m: $N/4$ boxes. B_8 is divided into two boxes, if we have (pseudo) items with height $h(B)$. In each of these parts each tall item has height at most $1/2h(B)$ and we create one box per item size. Therefore, we create at most $N/4$ boxes in this area in each part. The boxes for the items overlapping L_2 or R_2 are already counted for area $B_{l,7}$.

Area $B_{l,7}$: $2N/4$ boxes. We consider now the areas $B_{l,9}$ and $B_{r,9}$. In these areas, all items have height of at most $1/2h(B)$ and for each item height we create one box at the bottom and one box at $h(B) - 1/4H$. Therefore, we create at most $2N/4$ boxes in each area.

Area $B_{l,9}$: $N/4$ boxes. Last, we create at most one box for each item with height larger than $h(B) - 1/4H$ resulting in at most $N/4$ boxes for these items.

In total the number of generated boxes is bounded by $2N^2 + 15N/4 + 8$, which concludes the proof of the claim. ◁

Let us consider the number of boxes for vertical items.

Claim 5.19. The number of boxes for vertical items is bounded by $4N^2 + 31N/4 + 5$.

Proof. We proof this claim by considering the generated areas one after another and count the number of boxes generated in each of these areas.

Area $B_{l,1}$: $N/2$ boxes. In the areas $B_{l,1}$ and $B_{r,1}$, there are at most $N/4$ boxes for items touching the bottom since they have height of at most $1/4H$ and at most $N/4$ boxes for items touching the top since they have height of at least $1/4H$ and at most $1/2h(B)$. Therefore, in each of the areas $B_{l,1}$ and $B_{r,1}$ we generate at most $N/2$ boxes.

Area $B_{l,2}$: $N^2 + 2$ boxes. In the areas $B_{l,1}$ and $B_{r,1}$ the pseudo items touching the bottom have sizes between $1/4H$ and $1/2h(B)$ and the items touch-

ing the top have sizes up to $1/2h(B)$. By Lemma 5.13 we generate at most $4S_PS_{P\cup T} \leqslant 4\frac{N}{2}\frac{N}{2} = N^2$ boxes plus the two boxes for extending the unmovable items in each area. Therefore, in the areas $B_{l,2}$ and $B_{r,2}$, we create at most $N^2 + 2$ boxes for pseudo items each.

Area $B_{l,3}$: $N/4$ boxes. In the area $B_{l,3}$ and $B_{r,3}$ above the items l_b and r_b respectively, there are pseudo items with heights up to $1/4H$. For each size we generate at most one box. Therefore, we generate at most $N/4$ boxes in each of these areas.

Area $B_{l,4}$: $N/2$ boxes. In the area $B_{l,4}$ below the items with height larger than $1/2h(B)$, we have areas for pseudo items with height at most $1/4H$. We have two blocks of these items, one at i_l, the other at i_r. In each of these areas, we create at most $N/4$ boxes for these items. Furthermore, there can be pseudo items with heights between $1/2h(B)$ and $h(B) - 1/4H$; for each of these heights we create at most one box resulting in $N/4$ boxes for these items in each area $B_{l,4}$ and $B_{r,4}$.

Area $B_{l,5}$: $N/4$ boxes. In the area $B_{l,5}$ above the tall items with height between $1/2h(B)$ and $h(B) - 1/4H$ there are no pseudo items. They where shifted up, to have their lower border at $h(B) - H/4$. This area contains just pseudo items with height between $1/2h(B)$ and $h(B) - 1/4H$ and for each size we create at most one box, hence at most $N/4$ boxes.

Area $B_{l,6}$: N^2 boxes. In the areas $B_{l,6}$ and $B_{r,6}$ the contained pseudo items have heights between $1/4H$ and $1/2h(B)$ on the top and heights up to $1/2h(B)$ on the bottom. Therefore, by Lemma 5.13 we generate at most N^2 boxes analogously to the boxes $B_{l,2}$. Here, we do not create another pseudo item since the item $l_{t,l}$ already touches the top of the area. Therefore, we generate at most N^2 boxes in each of the areas $B_{l,5}$ and $B_{r,5}$.

Area $B_{l,7}$: N boxes. In the area $B_{l,7}$ above l_m, we have at most three areas for pseudo items touching l_m with their lower border. These items have a height of at most $1/4H$. Therefore, we create at most $3N/4$ for these pseudo items. Furthermore, the pseudo items touching $h(B) + 1/4H$ with their top have a height between $h(B)/4$ and $1/2h(B)$. For each height we generate at most one box. Therefore, we create at most $N/4$ boxes for these items. In total we generate at most N boxes for pseudo items in the areas $B_{l,7}$ and $B_{r,7}$ each.

Area B_8: $N/2$ boxes. In the area B_8 pseudo items with height up to $1/4H$

touch the bottom and items with sizes between $1/4H$ and $1/2h(B)$ are touching the top. For each size we generate at most one box. Therefore, in B_8 we generate at most $N/2$ boxes. The area B_8 can be split in two by the items with height $h(B)$. Therefore, we have to count the boxes in B_8 twice.

Area $B_{l,9}$: $3N/4$ boxes. In the area $B_{l,9}$ the tall items on the bottom have height between $1/4H$ and $3h(B)/4$. For each of these sizes we create at most one box, summing up to at most $N/4$. On the top of this area the pseudo items have heights up to $1/2h(B)$ and we create one box per size, creating at most $N/2$ boxes. Therefore in the areas $B_{l,4}$, we have at most $3N/4$ boxes in total.

Area $B_{l,10}$: $N/4$ boxes. Last, we consider the items with height larger than $3h(B)/4$ in area $B_{l,10}$. Above these items there can be pseudo items with heights up to $1/4H$. For each size we create at most one box. Therefore, we create at most $N/4$ boxes above these items. Furthermore, there can be at most one box contain a pseudo item with height $h(B)$.

In total we create at most $2(N/2 + N^2 + 2 + N/4 + N/2 + N^2 + N + N/2 + 3N/4 + N/4) + N/4 + 1 = 4N^2 + 31N/4 + 5$ boxes for vertical items, which concludes the proof of the claim. ◁

Since for both types of sub-boxes, those containing tall items and those containing sliceable items, we have shown that their number is small enough, this concludes the proof of this lemma. □

5.4 Structure Result

In this section, we prove the key to achieve the approximation ratio $(5/4 + \varepsilon)$OPT – the structural lemma. Roughly it states that each optimal solution can be transformed such that it has a simple structure, see Lemma 5.24. The heart of the proof – to reorder the items inside the boxes of height taller than $3/4$ – was discussed in the previous section. However, one challenge remains to be resolved: the placement of a constant number of extra boxes for vertical items that is used to provide an integral packing of those items after the rearrangement step. Unlike in the approaches in [91], [33] or [63], we cannot place them on the top of the packing since

we have to extend the packing beforehand by $1/4H$ using a shifting step to establish the simple structure. Fortunately, this shift creates some free area. A careful analysis of this area shows that this it can be used to place the boxes inside.

In this section, we will assume that we are given an instance with set of items $\mathcal{I}$ and an $\varepsilon \in \mathbb{R}$ such that $1/\varepsilon$ is integral. Furthermore, we are given an optimal packing of the items $\mathcal{I}$ with height OPT.

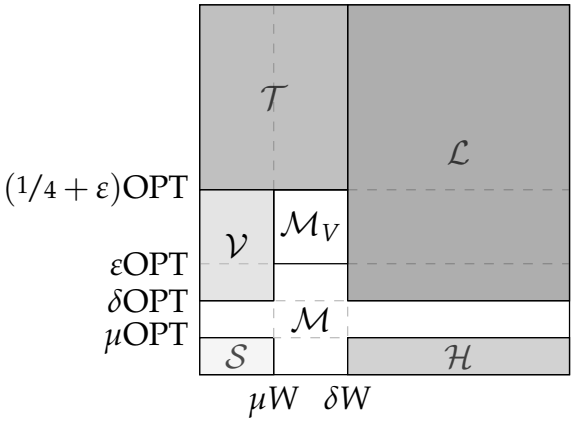


Figure 5.16. Partition of the items. Each item is represented by a dot in this plane. The x-coordinate represents its width while the y-coordinate represent its height.

In the first simplification step, we partition the set of items $\mathcal{I}$, see Figure 5.16 for an overview. Let $\delta = \delta(\varepsilon) \leqslant \varepsilon$ and $\mu = \mu(\varepsilon) < \delta$ be suitable constants depending on ε, and let OPT be the height of an optimal packing. We define

- $\mathcal{L} := \{i \in \mathcal{I} \mid h(i) > \delta\text{OPT}, w(i) \geqslant \delta W\}$ as the set of large items,
- $\mathcal{T} := \{i \in \mathcal{I} \mid h(i) \geqslant (1/4+\varepsilon)\text{OPT}, w(i) < \delta W\}$ as the set of tall items,
- $\mathcal{V} := \{i \in \mathcal{I} \mid \delta\text{OPT} \leqslant h(i) < (1/4+\varepsilon)\text{OPT}, w(i) \leqslant \mu W\}$ as the set of vertical items,
- $\mathcal{M}_V := \{i \in \mathcal{I} \mid \varepsilon\text{OPT} \leqslant h(i) < (1/4+\varepsilon)\text{OPT}, \mu W < w(i) \leqslant \delta W\}$ as the set of vertical medium items,
- $\mathcal{H} := \{i \in \mathcal{I} \mid h(i) \leqslant \mu\text{OPT}, \delta W \leqslant w(i)\}$ as the set of horizontal items,

- $\mathcal{S} := \{i \in \mathcal{I} \mid h(i) \leqslant \mu\text{OPT}, w(i) \leqslant \mu W\}$ as the set of small items and
- $\mathcal{M} := \{i \in \mathcal{I} \mid h(i) < \varepsilon\text{OPT}, \mu W < w(i) \leqslant \delta W\} \cup \{i \in \mathcal{I} \mid \mu\text{OPT} < h(i) \leqslant \delta\text{OPT}\} = \mathcal{I} \backslash (\mathcal{L} \cup \mathcal{T} \cup \mathcal{V} \cup \mathcal{M}_V \cup \mathcal{H} \cup \mathcal{S})$ as the set of medium sized items.

We want to choose δ and μ such that the total area of the items in $\mathcal{M}$ and $\mathcal{M}_V$ is small. The following Lemma states that we can find such suitable values for δ and μ.

Lemma 5.20. *Let $f : \mathbb{R} \to \mathbb{R}$ be any function such that $1/f(\varepsilon)$ is integral. Consider the sequence $\sigma_0 = f(\varepsilon)$, $\sigma_{i+1} = \sigma_i^2 f(\varepsilon)$. There is a value $i \in \{0, \ldots, (2/f(\varepsilon)) - 1\}$ such that the total area of the items in $\mathcal{M} \cup \mathcal{M}_V$ is at most $f(\varepsilon)W\text{OPT}$, if we set $\delta := \sigma_i$ and $\mu := \sigma_{i+1}$.*

Proof. This can be proven by the pidginhole principle. The sequence σ and the corresponding choice of δ and μ builds a sequence of $2/f(\varepsilon)$ sets $\mathcal{M}_{\sigma_i} \cup \mathcal{M}_{V\sigma_i}$. Each item $i \in I$ can occur in at most two of these sets, either because of its width or its height. Since the total area of all items is at most $W \cdot \text{OPT}$ one of the sets must have an area which is at most $f(\varepsilon) \cdot W \cdot \text{OPT}$. □

For this application it is sufficient to choose $f(\varepsilon) = \varepsilon^{13}/k$ for a constant $k \in \mathbb{N}$ which has to fulfill certain properties, as can be seen later. Since $1/\varepsilon \in \mathbb{N}$, we have that $1/f(\varepsilon) \in \mathbb{N}$. Let δ and μ be the values defined as in Lemma 5.20. Note that $\sigma_i = f(\varepsilon)^{(2^{i+1}-1)}$ and, therefore, $\delta \geqslant \sigma_{f(\varepsilon)-1} \geqslant (\varepsilon^{13}/k)^l \in \varepsilon^{\mathcal{O}(l)}$, where $l := (2^{2k/\varepsilon^{13}})$, i.e., $\delta \geqslant \varepsilon^{2^{\mathcal{O}(1/\varepsilon^{13})}}$. In the following steps, we need δ to be of the form ε^x for some $x \in \mathbb{N}$. Therefore, define $\delta' := \varepsilon^x$ such that $x \in \mathbb{N}$ and $\delta' \leqslant \delta \leqslant \delta'/\varepsilon$. Note that $\mu := \delta^2\varepsilon^{13}/k \leqslant (\delta'/\varepsilon)^2\varepsilon^{13}/k = \delta'^2\varepsilon^{11}/k$ and $\mu := \delta^2\varepsilon^{13}/k \geqslant \delta'^2\varepsilon^{13}/k$. In the following we will use δ' for all the steps, but omit the prime for simplicity of notation. By this choice it still holds that the set of medium sized items has a total area of at most $(\varepsilon^{13}/k)W\text{OPT}$ because by reducing δ and not changing μ we only removed jobs from this set.

Observation 5.21. Since each item in $\mathcal{M}_V$ has a height of at least εOPT and width of at least $\mu W \geqslant (\delta^2\varepsilon^{13}/k)W$, i.e., each item has an area of at least

$(\delta^2\varepsilon^{14}/k)W\text{OPT}$, it holds that

$$|\mathcal{M}_V| \leqslant (\varepsilon^{13}/k)W\text{OPT}/((\delta^2\varepsilon^{14}/k)W\text{OPT}) = 1/\delta^2\varepsilon.$$

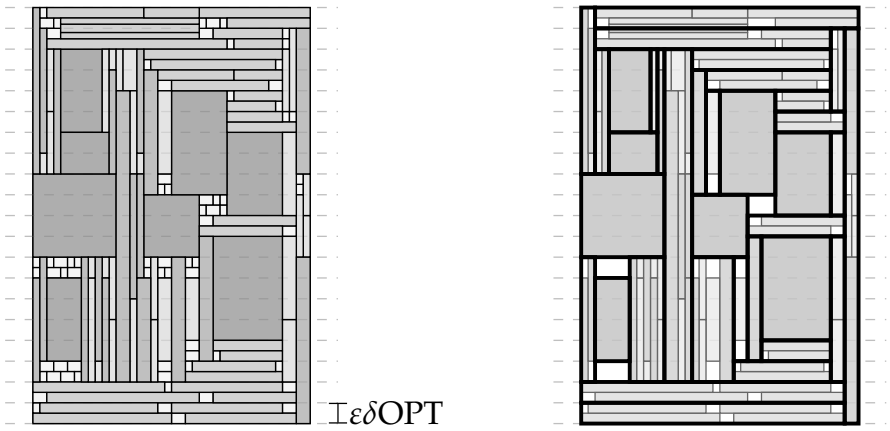
After we have found the corresponding values for δ and μ and after we have partitioned the set of items accordingly, we round the height of all items with height at least δOPT with the rounding technique from Lemma 3.2 using OPT as T. This is possible since δ is of form ε^x. Note that after this rounding the packing has a height of $(1+2\varepsilon)$OPT and all the items with processing time larger than δOPT will start at integral multiples of $\varepsilon\delta$OPT, while all times taller than εOPT will start at integral multiples of ε^2OPT. Furthermore, the number of item heights larger than δOPT is bounded by $\mathcal{O}(\log_\varepsilon(1/\delta)/\varepsilon^2)$ and the number of heights larger than εOPT is bounded by $\mathcal{O}(1/\varepsilon^2)$.

After this rounding step, we remove all items in $\mathcal{M} \cup \mathcal{S}$ from the optimal packing. Later, we will show that these items can be placed back into the packing with the NFDH algorithm (see [22]) without increasing the packing height too much, see Lemma 5.29 and Lemma 5.30.

At this point, the considered packing has a height of at most $(1+2\varepsilon)$OPT and contains the items $\mathcal{L} \cup \mathcal{T} \cup \mathcal{V} \cup \mathcal{M}_V \cup \mathcal{H}$. When rearranging the packing, we are allowed to slice the items in $\mathcal{V}$ vertically, while all the other items cannot be sliced. Therefore, in order to use the techniques from Section 5.3, we need to partition the packing area into sub-boxes that divide the vertical and tall items from the residual ones. The following lemma states that this division is possible by introducing a constant number of sub-boxes.

Lemma 5.22. *We can partition a rounded optimal packing, where the small and medium items are removed, into at most $\mathcal{O}(1/\delta^2\varepsilon)$ boxes such that the following conditions hold:*

- *There are $|\mathcal{L}| + |\mathcal{M}_V| \leqslant \mathcal{O}(1/\delta^2\varepsilon)$ boxes $\mathcal{B}_\mathcal{L}$ each containing exactly one item from the set $\mathcal{L} \cup \mathcal{M}_V$ and all items from this set are contained in these boxes.*
- *There are at most $\mathcal{O}(1/\delta^2\varepsilon)$ boxes $\mathcal{B}_\mathcal{H}$ containing all horizontal items $\mathcal{H}$ such that $\mathcal{B}_\mathcal{L} \cap \mathcal{B}_\mathcal{H} = \varnothing$. The horizontal items can overlap horizontal box borders, but never vertical box borders.*



(a) A rounded optimal packing **(b)** The partition into boxes

Figure 5.17. In this figure one can see an optimal packing (in 5.17a) and its partition into the rectangular subareas (in 5.17b). Note that some of the horizontal, vertical and tall items overlap the box borders.

▻ *There are at most* $\mathcal{O}(1/\delta^2\varepsilon)$ *boxes* $\mathcal{B}_{\mathcal{T}\cup\mathcal{V}}$ *containing all items in* $\mathcal{T}\cup\mathcal{V}$ *such that* $\mathcal{B}_{\mathcal{T}\cup\mathcal{V}} \cap (\mathcal{B}_{\mathcal{H}} \cup \mathcal{B}_{\mathcal{L}}) = \varnothing$*. The items contained in these boxes can overlap vertical box borders, but never horizontal box borders.*

▻ *The lower and upper border of each box is positioned at a multiple of* $\varepsilon\delta$OPT*.*

Proof. Let us consider our stretched optimal packing in the strip with height $(1+2\varepsilon)$OPT. In the first step, we give each item in $\mathcal{L}\cup\mathcal{M}_V$ its personal box, which has exactly the dimensions of that item. Since the item does not overlap any other item, the box does not either. Since $|\mathcal{L}\cup\mathcal{M}_V| \in \mathcal{O}(1/\delta^2\varepsilon)$, we create at most this number of boxes.

In the next step, we define boxes for the horizontal items: We partition the strip at multiples of $\varepsilon\delta$OPT into horizontal layers of height $\varepsilon\delta$OPT. Each item in $\mathcal{L}\cup\mathcal{M}_V\cup\mathcal{T}\cup\mathcal{V}$ starts and ends at a multiple of $\varepsilon\delta$OPT, so none of these items does start within one of these layers. To find the boxes for

horizontal items, we iterate the layers and partition them in the following way: We scan the considered layer from left to the right until we meet the first item in the set $\mathcal{H}$ or in the set $\mathcal{L} \cup \mathcal{M}_V \cup \mathcal{T} \cup \mathcal{V}$. We remember which item we met first, and draw a vertical line when we meet the first item out of the other set. We remember from which set the item came we had just met. We iterate further to the right until we again met an item from another set or the border of the strip. If we have not yet met the border of the strip, we draw a vertical line and continue as before. When we met the border of the strip, we look at our vertical lines. If we look at the set of items between two vertical lines we see that they either contain items from $\mathcal{L} \cup \mathcal{M}_V \cup \mathcal{T} \cup \mathcal{V}$ or items from $\mathcal{H}$, but there is no set which contains items from $\mathcal{H}$ and $\mathcal{L} \cup \mathcal{M}_V \cup \mathcal{T} \cup \mathcal{V}$ as well. The area between two vertical lines, which contains items from $\mathcal{H}$, defines a box for items in $\mathcal{H}$. Since each item in $\mathcal{H}$ has a width of at least δW we get at most $1/\delta - 1$ of these boxes per horizontal strip of height $\varepsilon\delta\mathrm{OPT}$. Since we have $(1+2\varepsilon)/(\varepsilon\delta)$ of these strips we get at most $(1+2\varepsilon)/(\varepsilon\delta) \cdot (1/\delta - 1) \leqslant (1+2\varepsilon)/(\varepsilon\delta^2)$ of these boxes. We call the set of these boxes $\mathcal{B}_\mathcal{H}$. Note that each strip that contains a large item can contain one less box for horizontal items. Since each large items overlaps at least $1/\varepsilon$ strips, we can generate at most $(1+2\varepsilon)/(\varepsilon\delta^2) - |\mathcal{L}|/\varepsilon$ boxes for horizontal items. Hence, the total number of boxes generated for horizontal, large and medium-vertical items is bounded by $|\mathcal{L}| + |\mathcal{M}_V| + |\mathcal{B}_\mathcal{H}| \leqslant (2+2\varepsilon)/(\varepsilon\delta^2) \in \mathcal{O}(1/(\delta^2\varepsilon))$.

Now we describe how to generate the boxes for the items in $\mathcal{T} \cup \mathcal{V}$. For each of the boxes $\mathcal{B}_\mathcal{H}$ and the items in $\mathcal{L} \cup \mathcal{M}_V$, we draw vertical lines on the left and on the right side, until they meet the first item out of $\mathcal{L} \cup \mathcal{M}_V$ or the first box in $\mathcal{B}_\mathcal{H}$. The area, which is bounded within two of these lines, defines a box for vertical and tall items. We call this set of boxes $\mathcal{B}_{\mathcal{T}\cup\mathcal{V}}$.

Let us consider how many boxes $\mathcal{B}_{\mathcal{T}\cup\mathcal{V}}$ we create using the above technique. We draw two lines for each of the boxes for horizontal, tall, and medium-vertical items. Additionally, we have the strip border which gives two additional lines. Each line touches at most 3 boxes in $\mathcal{B}_{\mathcal{T}\cup\mathcal{V}}$. Each of these boxes needs two lines as a border. So in total we have at most $3(2+2\varepsilon)/(\varepsilon\delta^2)$ boxes in $\mathcal{B}_{\mathcal{T}\cup\mathcal{V}}$. Furthermore, the total number of vertical lines intersecting vertical box borders is bounded by $(2+2\varepsilon)/(\varepsilon\delta^2)$. □

After applying the shifting and reordering technique from Section 5.3, the vertical items will be sliced. In the next lemma, we show that it is possible to place these items integral again. However, this integral placement comes at a cost. Namely, we have to introduce a constant number of narrow extra boxes for these items.

Lemma 5.23. *Let Y be the number of different heights of vertical items and μW the maximal width of a vertical item. Furthermore, let $\mathcal{B}_\mathcal{P}$ be the set of boxes, containing all sliced vertical items and only them.*

There exists a non fractional placement of the vertical items into the boxes $\mathcal{B}_\mathcal{P}$ and at most $7(Y + |\mathcal{B}_\mathcal{P}|)$ additional boxes $\mathcal{B}'_\mathcal{P}$ each of height at most $1/4H$ and width μW such that the boxes $\mathcal{B}_\mathcal{P} \cup \mathcal{B}'_\mathcal{P}$ are partitioned into at most $\mathcal{O}((Y + |\mathcal{B}_\mathcal{P}|)/\delta)$ sub-boxes $\mathcal{B}_\mathcal{V}$, containing only vertical items of the same height and at most $\mathcal{O}(Y + |\mathcal{B}_\mathcal{P}|)$ empty boxes $\mathcal{B}^\mathcal{V}_\mathcal{S}$ with total area $\text{area}(\mathcal{B}^\mathcal{V}_\mathcal{S}) \geqslant \text{area}(\mathcal{B}_\mathcal{P}) - \text{area}(\mathcal{V})$.

Proof. To prove this lemma, we first define a configuration LP, see Section 3.3 for a more detailed description of configuration LPs. In this configuration LP a configuration contains a set of jobs that can be placed on top of each other without exceeding the boundaries of the box the set of items is to be placed into. More precisely $C = \{a_h : h | h \in Y\}$, $h(C) := \sum_{h \in Y} h \cdot a_h$ and $\mathcal{C}_B$ is the set of configurations with heights at most $h(B)$. Furthermore, we define for each $h \in Y$ the value w_h as the total width of all vertical items with height h.

Consider the following configuration LP:

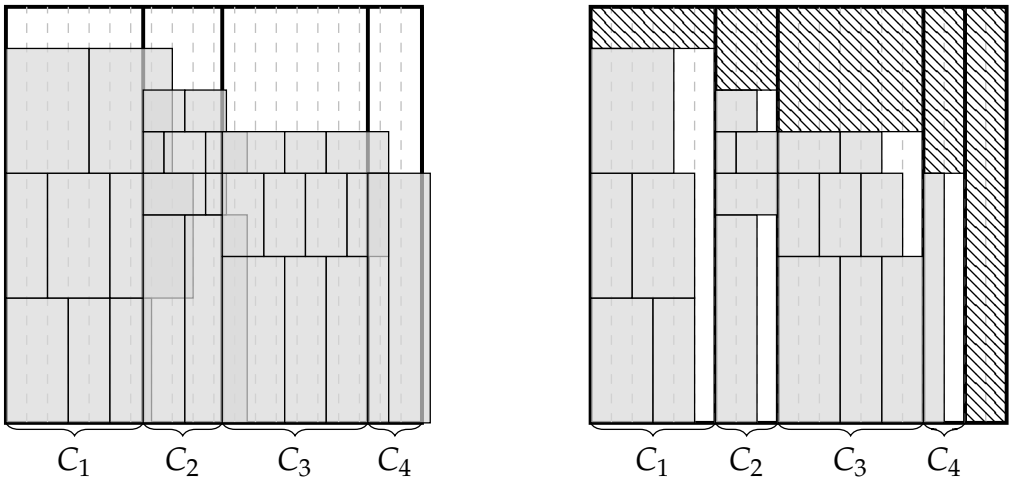
$$\sum_{C \in \mathcal{C}_B} X_C = w(B) \qquad \forall B \in \mathcal{B}_\mathcal{P}$$

$$\sum_{B \in \mathcal{B}_\mathcal{V}} \sum_{C \in \mathcal{C}_B} X_{C,B} a_{h,C} = w_h \qquad \forall h = 1, \ldots, Y$$

$$X_{C,B} \geqslant 0 \qquad \forall B \in \mathcal{B}_\mathcal{V}, C \in \mathcal{C}_B$$

It has, as each linear program, a basic solution with at most $Y + |\mathcal{B}_\mathcal{P}|$ non-zero components since it has at most this number of conditions.

Given such a basic solution, we place the corresponding configurations into the boxes. Afterward, we place the items into the configurations such that the last item overlaps the configuration border. Each configuration has a height of at most H since the boxes $\mathcal{B}_\mathcal{P}$ have at most this height.



(a) Configurations inside a box B with vertical items placed inside them.

(b) The hatched areas are empty boxes that can be used to place small items.

Figure 5.18. Configurations before and after removing the overlapping items and reducing the width to integrals.

We partition the set of overlapping items in each configuration into 7 boxes with height $1/4H$ and width μW in the following way: First, we stack the items in four boxes one by one on top of each other such that the last item overlaps the box on top. Since the total height of the items is at most H, there are at most three overlapping items. Each of them is placed into their own box. We call the set of these boxes $\mathcal{B}'_{\mathcal{P}}$. In total, we generate at most $7(|H_{\mathcal{V}}| + |\mathcal{B}_{\mathcal{P}}|)$ boxes of width μW. The items can be placed non-fractionally inside these boxes since they have a width of at most μW.

Note that the configuration width defined by the considered basic solution of the linear program might not be integral. However, we can reduce the configuration width to the next smaller integer since we have removed all the overlapping items and hence only need an integral width. As a result we might get an empty configuration inside the strip, which has at least the width of the sum of all non integral fractions we removed from the configurations in the box. This empty configuration has an integral width since the box has an integral width and all the other configurations

have an integral width as well.

Since the configurations have a height of at most H and each item has a height of at least δOPT, each configuration contains at most $H/(\delta\text{OPT}) \in \mathcal{O}(1/\delta)$ items. Therefore, the set of boxes $\mathcal{B}_\mathcal{P} \cup \mathcal{B}'_\mathcal{P}$ is divided into at most $2(Y + |\mathcal{B}_\mathcal{P}|)H/(\delta\text{OPT}) \in \mathcal{O}((|H_\mathcal{V}| + |\mathcal{B}_\mathcal{P}|)/\delta)$ sub-boxes containing only vertical items of the same height.

Consider a configuration $C \in \mathcal{C}_B$ which has a non-zero entry $X_{C,B}$ in the considered solution. Above this configuration there is a free area of height $h(B) - h(C)$ and width $X_{C,B}$ inside the box B, see Figure 5.18. Furthermore, in each box there might be a new empty configuration, which generates an empty box as well. Let $\mathcal{B}^\mathcal{V}_\mathcal{S}$ be the set of these boxes. There are at most $\mathcal{O}(|H_\mathcal{V}| + |\mathcal{B}_\mathcal{P}|)$; at most one above each configuration and one extra for each box. Since the configurations use exactly the area of the vertical items, the total area of these empty boxes has to be $\text{area}(\mathcal{B}^\mathcal{V}_\mathcal{S}) \geqslant \text{area}(\mathcal{B}_\mathcal{P}) - \text{area}(\mathcal{V})$. □

Consider the rounded optimal packing that is partitioned into the sub-boxes by the first partitioning step in Lemma 5.22. This packing has a height of at most $(1 + 2\varepsilon)\text{OPT}$. We will rearrange the items inside this packing and partition the packing some further such that the tall and vertical items are contained in boxes that only contain items with the same height.

Lemma 5.24. *(Structure Lemma) By extending the packing area to* $(5/4 + 5\varepsilon)\text{OPTW}$ *each rounded optimal packing can be rearranged and partitioned into* $\mathcal{O}(1/\delta^3\varepsilon^5)$ *boxes with the following properties:*

- *There are* $|\mathcal{L}| + |\mathcal{M}_V| = \mathcal{O}(1/\delta^2\varepsilon)$ *boxes* $\mathcal{B}_\mathcal{L}$ *each containing exactly one item from the set* $\mathcal{L} \cup \mathcal{M}_V$ *and all items from this set are contained in these boxes.*

- *There are at most* $\mathcal{O}(1/\delta^2\varepsilon)$ *boxes* $\mathcal{B}_\mathcal{H}$ *containing all horizontal items* $\mathcal{H}$ *with* $\mathcal{B}_\mathcal{H} \cap \mathcal{B}_\mathcal{L} = \varnothing$*. The horizontal items can overlap horizontal box borders, but never vertical box borders.*

- *There are at most* $\mathcal{O}(1/\delta^2\varepsilon^5)$ *boxes* $\mathcal{B}_\mathcal{T}$ *containing tall items such that each tall item* t *is contained in a box with rounded height* $h(t)$*.*

- ▷ *There are at most $\mathcal{O}(1/\delta^3\varepsilon^5)$ boxes $\mathcal{B}_\mathcal{V}$ containing vertical items such that each vertical item v is contained in a box with rounded height $h(v)$.*
- ▷ *There are at most $\mathcal{O}(1/\delta^2\varepsilon^5)$ boxes $\mathcal{B}_\mathcal{S}$ for small items such that the total area of these boxes combined with the total free area inside the horizontal boxes is at least as large as the total area of the small items.*
- ▷ *The lower and top border of each box is positioned at a multiple of $\varepsilon\delta$OPT.*

Proof. In the following, we give a short overview of this proof. We start with the partition from Lemma 5.22 and define $H := (1+2\varepsilon)$OPT. Note that by this definition we have $H/4 \leqslant (1/4+\varepsilon)$OPT and thus each tall item has a height larger than $H/4$ as needed. Since we already have seen how it is possible to reorder the items inside the boxes (see Section 5.3), the main task in this proof is to find a place for the extra boxes for vertical items, which we need to place them integrally, see Lemma 5.23. We consider three options to place these boxes. First, we consider the widest tall items intersecting the horizontal line at $1/2H$ and fix their position. We aim to place the extra boxes on top of them if the total width of these items is large enough. Otherwise, we know that all the tall items intersecting this line are very thin and we can find a way to place the extra boxes inside the boxes with height at least $3/4H$, if the total width of these boxes is large enough. The last option is to place them on top of the boxes with height between $1/2H$ and $3/4H$.

Another task in this proof is to provide the condition assumed in Section 5.3. Namely we have to ensure that the following conditions are provided:

First, no box B with height at least $3/4H$ is allowed to be intersected at its border at the horizontal line $S(B)+h(B)-H/4$ by a tall item. This can be done by introducing at most two further boxes of height at most $3/4H$ per box B.

Second, we need space above the tall boxes to be able to extend them by $1/4H$. Hence the next step is to shift up the boxes which have their lower border above $3/4H$ by $1/4H+\varepsilon$OPT. We need the extra shift by εOPT for technical reasons.

Last, the tall and medium boxes have to start and end at the grid lines. Since the tall items start and end at multiples of ε^2OPT, we choose these

lines as the grid lines and change the start and endpoints of the tall and medium boxes accordingly at a small loss in the approximation ratio.

When all these properties are fulfilled, we can apply Lemmas 5.15, 5.9 and 5.8 to reorder the items inside the boxes $\mathcal{B}_{\mathcal{T}\cup\mathcal{V}}$. Afterward, we analyze the number of containers constructed for vertical items and find a place for the resulting set of additional containers, which we need by Lemma 5.23 to place the vertical items non-fractional. In the final step, we consider the boxes for horizontal and small items.

Step 1: The widest tall items intersecting $H/2$. In the first step, we look at the $1/(\delta^2\varepsilon)$ widest tall items crossing the horizontal line at $1/2H$. We call the set of these items $T_{1/2H}$. Each of these items defines a new unmovable item. It splits the box containing it into three parts: The part left of this item, the part right of it and the part containing it. The parts left and right will be reordered as any other box, while the part containing this item is reordered differently. The item itself is not moved, while the part above and below have a height of less than $1/2H$. These parts define new boxes, which are small and hence can be reordered by Lemma 5.8 such that they create at most $\mathcal{O}(N)$ sub-boxes for tall and vertical items total. These are less than the number of sub-boxes created for one box of height larger than $3/4H$. Therefore, we can count this part as one box without making any error and assume that we add at most $2/(\delta^2\varepsilon)$ boxes total. After this step, the total number of boxes containing both, tall and vertical items, is bounded by $\mathcal{O}(1/(\delta^2\varepsilon))$. Furthermore, the number of vertical lines at box borders through the strip is bounded by $\mathcal{O}(1/\delta^2\varepsilon)$ as well.

Step 2: Providing the conditions assumed for the reordering. In this step, we have to provide three conditions: First, no item is allowed to overlap the tall box borders at the horizontal line at $S(B) + h(B) - H/4$; second, we need a gap of height $H/4$ between the upper border of each box of height at least $3/4H$ as well as some extra free area above the medium sized boxes, to place the discarded pseudo items; third, the medium and tall boxes have to start and end at the grid lines.

First condition: No overlapping at $S(B) + h(B) - H/4$. To provide the first condition, we look at each box B with height at least $3/4H$, see Figure 5.19. Remember that in Lemma 5.15, we had assumed that no tall item overlaps B's left or right box border at $S(B) + h(B) - 1/4H$. We will establish this

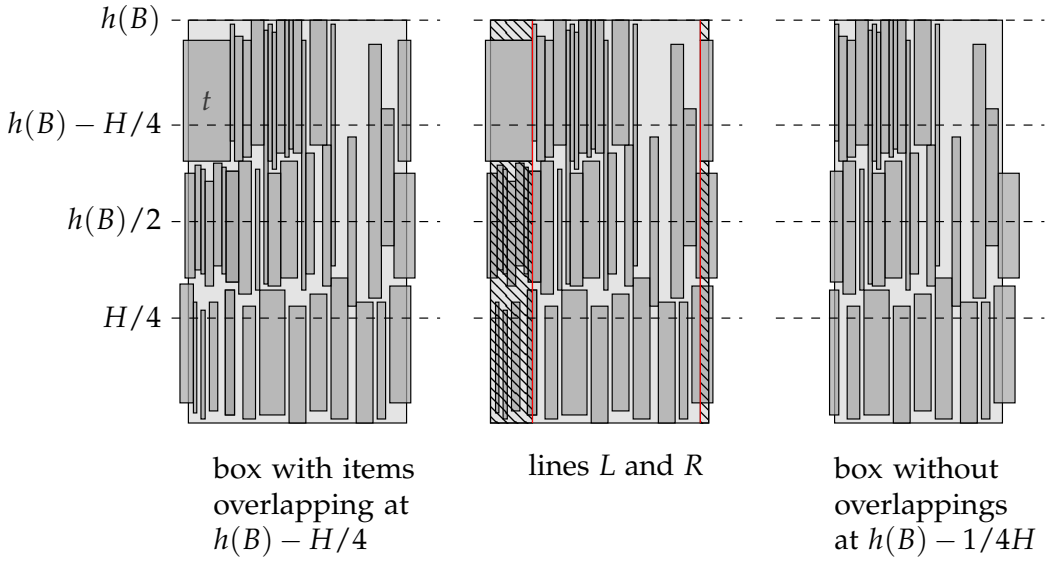


Figure 5.19. Eliminating overlaps at $h(B) - 1/4H$

property by introducing two boxes for tall and vertical items of height less than $3/4H$.

Assume there is a tall item t overlapping the left box border at $S(B) + h(B) - 1/4H$, see Figure 5.19. We draw a vertical line L at the right border of t inside our box. Tall items crossed by L represent new unmovable items. Obviously, L is not intersected by a tall item at height $S(B) + h(B) - 1/4H$. Consider the rectangular area between the left border of the box B and L bounded on top by t. This area builds a new box for vertical and tall items with height less than $3/4H$ and will later be reordered accordingly using Lemma 5.9. The rectangular area above t between these vertical lines builds a pseudo item containing vertical items. We repeat this step on the right side of the box.

In this step, we created for each of the tall boxes at most four new ones. Hence the number of boxes for tall and vertical items is still bounded by $\mathcal{O}(1/(\delta^2\varepsilon))$. This number, denoted as N_B, will not increase in the following steps. Furthermore, the number of distinct vertical lines at each box border

through the strip, denoted as N_L, is bounded by $\mathcal{O}(1/(\delta^2\varepsilon))$.

Second condition: Free area above tall and medium boxes. To ensure the second property, we draw a horizontal line at $3/4H$ trough the strip and shift each box with lower border above or at this line exactly $1/4H + \varepsilon\text{OPT}$ upwards. We split each item that is overlapping the box border at the border during this shift. Note that no tall item is shifted since they start before $3/4H$ and, thus, their boxes do, too. Hence the only items that will be split are vertical items (which are already sliced) and horizontal items, which we might slice horizontally. We fix this splitting in a later step. After this shift, on top of each box with height at least $3/4H$, there is a gap of height $1/4H + \varepsilon\text{OPT}$ since these boxes end after $3/4H$ and thus all the boxes above are shifted upwards.

Notice that we add an extra εOPT to the height. In the later reordering of boxes with height at least $3/4H$, we have shifted the items crossing the line $h(B) - 1/4H$ exactly $1/4H + \varepsilon\text{OPT}$ upwards as just $1/4H$ like in the proof of Lemma 5.15 is not sufficient. This extra height inside the tall boxes in $\mathcal{B}_{\mathcal{T}\cup\mathcal{V}}$ is necessary to prove the existence of the gaps where we place the extra boxes for vertical items.

Consider a box B of height larger than $1/2H$ and at most $3/4H$. By Lemma 5.9, we need an extra box with height $1/4H$ and width $(1 - (1/N))w(B) \leqslant (1-\varepsilon^2)w(B)$ to rearrange the items in B. Due to the shifting, somewhere above this box, there is free area of height $1/4H + \varepsilon\text{OPT}$ and width $w(B)$, which is possibly divided into several vertical slices. Let us look at the free area above all the boxes with height between $3/4H$ and $1/2H$. This free area is scattered into at most $N_L + 1$ vertical pieces since there are at most N_L vertical lines at box borders. We allocate this free area above the boxes as contiguously as possible. For each piece of the free area we use, we introduce one box for vertical items (at most $N_L + 1$). Let $W_{1/2}$ be the total width of boxes with height larger than $1/2H$ and at most $2/4H$. The total width of the free area above these boxes, which we have to use to place the pseudo items from inside the medium sized boxes, is bounded by $(1-\varepsilon^2)W_{1/2}$ and we have a total width of at least $\varepsilon^2 W_{1/2}$ to position the extra boxes needed to pack the vertical items non fractional.

Third condition: Alignment of tall and medium boxes. In Lemmas 5.15 and 5.9 we assume that each box with height larger than $1/2H$ starts and ends

at grid points. In this step, we generate this property. Grid lines are defined as the multiples of ε^2OPT. Let B be a box with height larger than $1/2H$. Look at the horizontal line l at the smallest multiple of ε^2OPT in this box. The distance between l and the bottom border is smaller than ε^2OPT. In the box B, we will remove all the vertical items below and each item cut by l and position them in an extra box at the end of the packing. Since each item with height larger than εOPT starts and ends at multiples of ε^2OPT, the items cut by l have a height of at most εOPT. We do the same on top of this box and for each other box. We create above $5/4H + \varepsilon$OPT a box with height $2(\varepsilon + \varepsilon^2)$OPT and width W. For each vertical line trough the strip, there is at most one box with height larger than $H/2$. Hence, when shifting up these items such that they are positioned inside the new box while they maintain their relative positions, we do not provoke any overlapping.

Step 3: Reorder tall and vertical items. After all necessary conditions are fulfilled, we apply the Lemmas 5.15, 5.9, and 5.8 to reorder the items inside the boxes for tall and vertical items. Since we create the most sub-boxes for tall boxes, we pessimistically assume that all the given boxes for tall and vertical items are tall, i.e., have a height larger than $3/4H$. We create at most $\mathcal{O}(1/\varepsilon^4)$ sub-boxes for tall and at most $\mathcal{O}(1/\varepsilon^4)$ sub-boxes for vertical items per box for tall and vertical items; remember that $N = \lceil (1+3\varepsilon)/\varepsilon^2 \rceil$. To this point, we generate at most $\mathcal{O}(1/(\delta^2\varepsilon^5))$ boxes for tall items in total.

It is necessary to further divide the sub-boxes inside the tall boxes in $\mathcal{B}_{\mathcal{T}\cup\mathcal{V}}$ to enable the placement of the extra boxes for vertical items. Consider the boxes for tall and vertical items in $\mathcal{B}_{\mathcal{T}\cup\mathcal{V}}$ that have a height larger than $3/4H$. In each of these boxes B, we draw a vertical line at the left border of each contained sub-box. If a sub-box for vertical items inside B is intersected by such a vertical line, we split the sub-box at this line. Each of these lines intersects at most three boxes for vertical items since at each point there can be at most four boxes (for tall or vertical items) on top of each other inside B. Hence, by splitting the vertical boxes this way, we introduce at most three new boxes for vertical items, per vertical line. Since there are at most $\mathcal{O}(1/(\delta^2\varepsilon^5))$ sub-boxes for tall and vertical items, the number of vertical lines is bounded by $\mathcal{O}(1/(\delta^2\varepsilon^5))$ as well. And hence after the splitting the number of boxes for vertical items is still bounded

by $\mathcal{O}(1/(\delta^2\varepsilon^5))$.

The area between two consecutive lines defines a strip, where the height of all the intersected boxes does not change. We have at most $\mathcal{O}(1/(\delta^2\varepsilon^5))$ of these strips total. We define N_S as the number of these strips.

Step 4: Placing the extra boxes for vertical items. By Lemma 5.23, we need at most $\mathcal{O}(|H_\mathcal{V}| + |\mathcal{B}_\mathcal{P}|)$ additional boxes with height $1/4H$ and width μW to place the vertical items non-fractionally into the boxes, where $\mathcal{B}_\mathcal{P}$ are the boxes for vertical items created so far. We call the set of these additional boxes $\mathcal{B}_{\mu W}$. We can bound the variables in the following way. There are at most $|\mathcal{B}_\mathcal{P}| \in \mathcal{O}(1/(\delta^2\varepsilon^5))$ boxes for vertical items and at most $|H_\mathcal{V}| \leqslant 1/\delta\varepsilon$ different heights of the items (which is a rather rough estimation). Therefore, we need at most $N_F \in \mathcal{O}(1/(\delta^2\varepsilon^5))$ extra boxes $\mathcal{B}_{\mu W}$.

We have to place the additional boxes inside the packing area $W \cdot 5/4H$. In the following steps, we will prove that it is possible to place them by considering three possibilities. Consider again the vertical lines at the box borders (not the sub-box borders). These N_L lines generate at most $N_L + 1$ strips. Let W_T be the total width of the strips containing items from $T_{1/2H}$, W_H be the total width of the strips containing boxes with height at least $3/4H$ and W_R be the total width of all other strips. In total we have $W_T + W_H + W_R = W$. We can assume $N_B \leqslant c_B/(\delta^2\varepsilon)$, $N_S \leqslant c_S/(\delta^2\varepsilon^5)$, $N_F \leqslant c_F/(\delta^2\varepsilon^5)$ and $N_L \leqslant c_L/(\delta^2\varepsilon)$ for some constants $c_B, c_S, c_F, c_L \in \mathbb{N}$. At this point it is necessary to define the function f to find the values δ and μ more precisely and we specify $f(\varepsilon)$ by choosing $k \leqslant (4(c_B + c_F + c_L)c_S)$. Hence it holds that $\mu \leqslant \delta^2\varepsilon^{11}/(4(c_B + c_F + c_L)c_S)$.

Consider the strips without boxes of height $3/4H$ or the items in $T_{1/2H}$. These strips can contain boxes with height larger than $1/2H$. Therefore, we have free area with total width of at least $\varepsilon^2 W_R$ in these strips.

Claim 5.25. If $W_R \geqslant \varepsilon^4 W$, we can place the N_F boxes $\mathcal{B}_{\mu W}$ into these areas.

Proof. The considered strips might contain boxes with height larger than $1/2H$ and less than $3/4H$. Therefore, the free area in these strips will be partially used by the extra boxes for pseudo items for these boxes. Nevertheless, these strips contain free area with width at least $\varepsilon^2 W_R$ that we can use to place the extra boxes $\mathcal{B}_{\mu W}$, see Lemma 5.9. In each of these

at most $(N_l + 1)$ strips the free area is contiguous. However, we have to calculate a small error that might occur: Each of the boxes in $\mathcal{B}_{\mu W}$ has a width of μW and, therefore, in each strip there is a residual width of up to μW where we cannot place a box from the set $\mathcal{B}_{\mu W}$, see Figure 5.20.

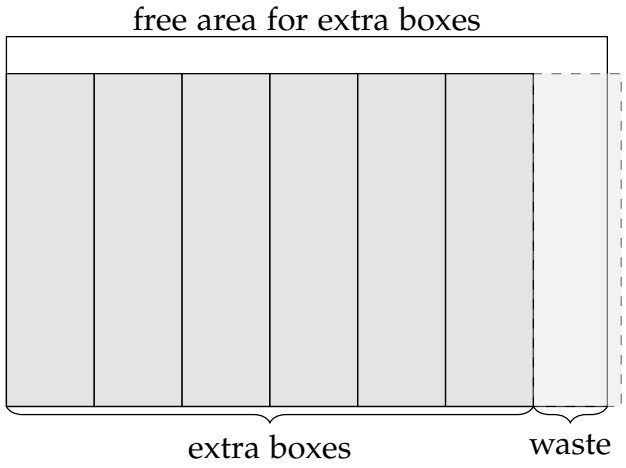


Figure 5.20. The waste of the free area, which can have a width of up to μW.

On the positive side, we can use an area with total width of at least $\varepsilon^2 W_R - (N_L + 1)\mu W$ to place the boxes in $\mathcal{B}_{\mu W}$ since there are at most $N_L + 1$ strips. Therefore if $\varepsilon^2 W_R - (N_L + 1)\mu W \geqslant N_F \mu W$, we can place all the boxes. Using $\mu := \delta^2 \varepsilon^{11} / (4(c_B + c_F + c_L)c_S)$, it holds that

$$N_F \mu W + (N_L + 1)\mu W = \mu W(c_F/\delta^2 \varepsilon^5 + c_L/\delta^2 \varepsilon + 1) \leqslant \varepsilon^{11} W/\varepsilon^5 \leqslant \varepsilon^6 W.$$

Therefore, if $W_R \geqslant \varepsilon^4 W$, it holds that $\varepsilon^2 W_R - (N_L + 1)\mu W \geqslant N_F \mu W$ and we can place all the boxes $\mathcal{B}_{\mu W}$, which concludes the claim. ◁

Claim 5.26. If $W_T \geqslant \varepsilon^6 W/(4c_S)$, we can place the N_F boxes $\mathcal{B}_{\mu W}$ in the strips containing the items in $T_{1/2H}$.

Proof. There are at most $N_L + 1$ strips containing parts of the items in $T_{1/2H}$. In these strips the free area is contiguous and can be fully used since these strips do not contain boxes with height larger than $1/2H$. Each box in $\mathcal{B}_{\mu W}$ has a width of exactly μW. Hence, in each strip there is an area with width of at most μW which we cannot use to place the boxes. Therefore,

if $W_T - \mu W N_L \geqslant N_F \mu W$, we can place all the N_F boxes into these strips. Using $\mu := \delta^2 \varepsilon^{11} / (4(c_B + c_F + c_L)c_S)$, it holds that

$$N_F \mu W + \mu W N_L = \mu W (c_F / \delta^2 \varepsilon^5 + c_L / \delta^2 \varepsilon) \leqslant \varepsilon^6 W / (4 c_S).$$

Therefore, if $W_T \geqslant \varepsilon^6 W / (4c_S)$, it holds that $W_T - \mu W N_L \geqslant N_F \mu W$ and we can place all the boxes $\mathcal{B}_{\mu W}$, which concludes the claim. ◁

Claim 5.27. If $W_T < \varepsilon^6 W / (4c_S)$ and $W_R < \varepsilon^4 W$, we can place all the boxes for vertical items inside the boxes of height at least $3/4H$.

Proof. In this case it holds that $W_H = W - (W_T + W_R) > (1 - 2\varepsilon^4)W \geqslant \varepsilon W$. Furthermore, each tall item not in $T_{1/2H}$ crossing $1/2H$ has a width of at most $W_T \cdot \delta^2 \varepsilon < \varepsilon^7 \delta^2 W / 4c_S := w_{\max}$. After the reordering in the boxes, there are at most N_S strips in the boxes total. We are interested in the total height of the free area inside a strip. This area might be non-contiguous since there could occur an item in the middle of this strips and some free area above and below this item. In the shifting step, we have added a total area of $W_H(1/4H + \varepsilon \mathrm{OPT})$ to all of these strips. Let $\check{W}_H$ be the total width of the strips containing free area with total height less than $1/4H$ and let $\hat{W}_H$ be the total width of strips containing free area with height larger than $1/4H$. We want to use the strips containing free area of total height at least $1/4H$ to place the extra boxes. Therefore, we have to prove that these strips have a sufficient minimum total width; more precisely we prove the following remark:

Remark. It holds that $\hat{W}_H \geqslant \varepsilon W_H$.

In each strip the total free area can have a height of at most $3/4H + \varepsilon \mathrm{OPT}$ since at the top and at the bottom there are always boxes with height at least $1/4H$ or there has to be a box with height at least $3/4H$ on the bottom. It holds that $\check{W}_H + \hat{W}_H = W_H$. Furthermore, it holds that

$$1/4H \cdot \check{W}_H + (3/4H + \varepsilon \mathrm{OPT}) \cdot \hat{W}_H \geqslant W_H(1/4H + \varepsilon \mathrm{OPT})$$

since the free area in $\check{W}_H$ has a total height of at most $1/4H$ and the free area in $\hat{W}_H$ has a height of at most $(3/4H + \varepsilon \mathrm{OPT})$ and the total free area is bounded by $W_H(1/4H + \varepsilon \mathrm{OPT})$. As a consequence, we can prove that

$\hat{W}_H$ has a sufficient minimum size. It holds that

$$\begin{aligned} W_H(1/4H + \varepsilon\mathrm{OPT}) &\leqslant 1/4H \cdot \check{W}_H + (3/4H + \varepsilon\mathrm{OPT}) \cdot \hat{W}_H \\ &= 1/4H \cdot W_H + (1/2H + \varepsilon\mathrm{OPT}) \cdot \hat{W}_H \\ &= 1/4H \cdot W_H + ((1+2\varepsilon)\mathrm{OPT}/2 + \varepsilon\mathrm{OPT}) \cdot \hat{W}_H \\ &= 1/4H \cdot W_H + ((1+4\varepsilon)\mathrm{OPT}/2) \cdot \hat{W}_H, \end{aligned}$$

and, therefore, we can deduce

$$\varepsilon W_H \leqslant ((1+4\varepsilon)/2) \cdot \hat{W}_H.$$

Thus, it holds that

$$\hat{W}_H \geqslant 2\varepsilon W_H/(1+4\varepsilon) \geqslant \varepsilon W_H, \quad \text{for } \varepsilon \leqslant 1/4,$$

which concludes the proof of the remark.

Consequently, strips with total width of at least εW_H contain free area with total height at least $1/4H$. The free area in this strips can be scattered into at most two pieces. We will fuse this free area by shifting the boxes for vertical or tall items. Notice that we can shift the boxes for vertical items in each strip freely up and down since their box borders are at the strip borders by construction. This is different for the sub-boxes for tall items, which can be positioned between $1/2h(B)$ and $h(B) - 1/4H$. These sub-boxes possibly contain tall items overlapping the strip's borders. Remember that each tall item in this strip has a width of at most $w_{\max} = \varepsilon^7\delta^2 W/(4c_S)$. Hence, in each strip with width larger than $2w_{\max} = \varepsilon^7\delta^2 W/(2c_S)$, we can shift the middle part of these sub-boxes such up or down such that the free area is connected. We do not shift the sub-boxes touching the bottom or the top of the box. In each strip, there is an area with width at most μW which we cannot use to place the boxes. Therefore, we can place all boxes for previously fractional vertical items, if $\varepsilon W_H - 2w_{max}N_S - \mu W N_S \geqslant \mu W N_F$. It holds that

$$\begin{aligned} &2w_{max}N_S + \mu W N_S + \mu W N_F \\ &\leqslant (\varepsilon^7\delta^2 W/(2c_S))(c_S/\delta^2\varepsilon^5) + \mu W(c_S/\delta^2\varepsilon^5 + c_F/\delta^2\varepsilon^5) \\ &\leqslant \varepsilon^2 W/2 + \varepsilon^6 W \leqslant \varepsilon^2 W. \end{aligned}$$

Thus, if $W_H \geqslant \varepsilon W$, it holds that $\varepsilon W_H - 2w_{max}N_S - \mu W N_S \geqslant \mu W N_F$ and we

can place all boxes in this case which concludes the proof of this claim. $\triangleleft$

In this step, we create at most $2N_S \in \mathcal{O}(1/(\varepsilon^5\delta^2))$ new boxes for tall items and no new box for vertical items. The boxes for tall items already do contain just tall items with the same height. Hence, we introduce at most $\mathcal{O}(1/(\varepsilon^5\delta^2))$ boxes for tall items in total. Furthermore, by Lemma 5.23, we create at most $\mathcal{O}(1/(\varepsilon^5\delta^3))$ boxes for vertical items $\mathcal{B}_\mathcal{V}$ such that each box $B \in \mathcal{B}_\mathcal{V}$ contains just items with height $h(B)$.

The boxes for small items. The free area inside the boxes from the partition in Lemma 5.22 for horizontal, tall, and vertical items is at least as large as the total area for the small items since the small items where contained in the optimal packing area.

Bounding the packing height. Let us recapitulate what we added to the packing height during this process. We started with a packing of height OPT. After the rounding of the items with height larger than δ and rounding the horizontal items, we received a packing with height $(1+2\varepsilon)\text{OPT}$. With the shifting at the horizontal line $3/4H$ we added $1/4H + \varepsilon\text{OPT} \leqslant 1/4(1+2\varepsilon)\text{OPT} + \varepsilon\text{OPT}$ to the packing height. Then, we shifted some vertical items to ensure that the boxes with height taller than $1/2H$ start and end at multiples of $\varepsilon^2\text{OPT}$. This added further $2(\varepsilon+\varepsilon^2)\text{OPT}$ to the packing height. In total we have added at most $1/4(1+2\varepsilon)\text{OPT} + 2(\varepsilon+\varepsilon^2)\text{OPT} \leqslant (1/4+3\varepsilon)\text{OPT}$ to the packing height (if $\varepsilon \leqslant 1/2$) such that the structured packing has a height of at most $(5/4+5\varepsilon)\text{OPT}\cdot W$.

$\square$

In the next step, we proof that there is an algorithm that can place the horizontal items inside their boxes. This algorithm creates a constant number of sub boxes for small items.

Lemma 5.28. *There is an algorithm with running time* $(\log(1/\delta)/\varepsilon)^{\mathcal{O}(1/\varepsilon\delta^3)}$ *that places the horizontal items into the boxes* $\mathcal{B}_\mathcal{H}$ *and an extra box* B_H *of height at most* $\varepsilon^9\text{OPT}$ *and width* W.

Furthermore, the algorithm creates at most $\mathcal{O}(1/\varepsilon\delta^2)$ *empty boxes* $\mathcal{B}_\mathcal{S}^\mathcal{H}$ *with total area* $\text{area}(\mathcal{B}_\mathcal{S}^\mathcal{H}) = \text{area}(\mathcal{B}_\mathcal{H}) - \text{area}(\tilde{\mathcal{H}})$.

Proof. The first step is to round the horizontal items. We stack horizontal items on top of each other ordered by their width such that the widest item is positioned at the bottom. This stack has a height of at most OPT/δ since each item has a width of at least δW and their total area is bounded by $\mathrm{OPT} \cdot W$. We group the items in the stack to at most $1/\varepsilon\delta^2$ groups, each of height $\varepsilon\delta^2\mathrm{OPT}/\delta = \varepsilon\delta\mathrm{OPT}$ and round the items in the groups to the widest width occurring inside this group. This step reduces the number of different sizes to at most $1/\delta\varepsilon^2$. The rounded horizontal items can be placed fractionally into the non-rounded items of the group containing the next larger items. The group containing the widest rounded items has to be placed on top of the packing. Therefore, the total height of items we put on the top of the packing has a height of at most $\delta\varepsilon\mathrm{OPT}$. We define an extra box of width W and height $\delta\varepsilon\mathrm{OPT}$ for these items. For simplicity of notation we assume in the following that $\mathcal{B}_\mathcal{H}$ contains this extra box as well.

We place the rounded horizontal items into the boxes using a configuration LP. In this scenario, a configuration is a set of items that fit next to each other inside the boxes, i.e., a configuration C is a multiset of the form $\{a_w : w \mid w \in \mathcal{W}_\mathcal{H}\}$ and the width of a configuration is defined as $w(C) := \sum_{w \in \mathcal{W}_\mathcal{H}} a_w w$. Furthermore, $\mathcal{C}_w$ denotes the set of configurations with width at most w, where $\mathcal{W}_\mathcal{H}$ is the set of different width appearing in the set of rounded horizontal items $\mathcal{H}$. Finally, we define $h(w)$ as the total height of all the items with width w.

The set of configurations $\mathcal{C}_W$ is bounded by $\mathcal{O}((\log(1/\delta)/\varepsilon)^{1/\delta})$ because the items have a width of at least δW and hence there can be at most $1/\delta$ items in each configuration. The following configuration LP is solvable since the rounded horizontal items fit fractionally into the boxes $\mathcal{B}_\mathcal{H}$.

$$\sum_{C \in \mathcal{C}_{w(B)}} X_{C,B} = h(B) \qquad \forall B \in \mathcal{B}_\mathcal{H}$$

$$\sum_{B \in \mathcal{B}_\mathcal{H}} \sum_{C \in \mathcal{C}_{w(B)}} X_{C,B} a_{w,C} = h(w) \qquad \forall w \in \mathcal{W}_\mathcal{H}$$

$$X_{C,B} \geqslant 0 \qquad \forall B \in \mathcal{B}_\mathcal{H}, C \in \mathcal{C}_{w(B)}$$

We can solve this linear program by guessing the at most $|\mathcal{W}_\mathcal{H}| + |\mathcal{B}_\mathcal{H}| = \mathcal{O}(1/(\varepsilon\delta^2))$ non-zero entries of the basic solution and solve the

resulting equality system using the Gauß-Jordan-Elimination. We use the first solution we find where all the variables are non-negative. Such a solution can be found in at most $\mathcal{O}(|\mathcal{C}_W|^{|\mathcal{W}_\mathcal{H}|+|\mathcal{B}_\mathcal{H}|} \cdot (|\mathcal{W}_\mathcal{H}| + |\mathcal{B}_\mathcal{H}|)^3) \leqslant (\log(1/\delta)/\varepsilon)^{\mathcal{O}(1/\varepsilon\delta^3)}$ operations since the configuration LP has to be solvable for the correct partition.

We place the corresponding configurations into the corresponding boxes and place the original horizontal items greedily into the configurations such that the last item overlaps the configuration border. We place the original items one by one inside an area reserved by the configurations for their rounded counterparts until an item overlaps this area on the top. Then we proceed to the next area. Since the total height of these parts is exactly as large as the total height of the items with this rounded width, there are enough parts to place all of them.

In the next step, we remove the overlapping items and place them on top of the box. Each of these removed items has a height of at most $\mu\mathrm{OPT}$. We add at most $\varepsilon^{10}\mathrm{OPT}$ to the packing height by shifting the overlapping items to the top of the packing, because first, a basic solution has at most $\mathcal{O}(1/(\delta^2\varepsilon))$ configurations; second, all the items in one configuration can be placed next to each other; and third, $\mu \leqslant \delta^2\varepsilon^{11}/k$ for a suitable large constant k. Together with the extra box that we need due to the rounding, the total added height is bounded by $\varepsilon^{10}\mathrm{OPT} + \delta\varepsilon\mathrm{OPT} \leqslant \varepsilon^9\mathrm{OPT}$.

Similar as before, we can reduce the height of each configuration to the next smaller integer since the horizontal items have an integral height. This introduces at most one new configuration per box, i.e., the one which is empty. In each box B to the right of each (used) configuration C there might be some free area of width $w(B) - w(C)$ and height $X_{C,B}$. This area defines one of the empty boxes $\mathcal{B}_\mathcal{S}^\mathcal{H}$. Since there are at most $|\mathcal{W}_\mathcal{H}| + |\mathcal{B}_\mathcal{H}|$ configurations and at most $|\mathcal{B}_\mathcal{H}|$ boxes for horizontal items, we introduce at most $\mathcal{O}(1/\varepsilon\delta^2)$ empty boxes $\mathcal{B}_\mathcal{S}^\mathcal{H}$. Furthermore, their total area has to be at least as large as $\mathrm{area}(\mathcal{B}_\mathcal{S}^\mathcal{H}) = \mathrm{area}(\mathcal{B}_\mathcal{H}) - \mathrm{area}(\tilde{\mathcal{H}})$ since the configurations contain exactly the total area of the rounded horizontal items.

Let us now consider the boxes for horizontal items which we create in this step. Each configuration contains at most $1/\delta$ positions for items. For each of these positions we create one box that has the rounded width of the items for these positions and (integral) height that is the sum of all the heights of the items positioned inside this box and we create one additional

box for the shifted item. Hence we introduced at most $\mathcal{O}(1/(\varepsilon\delta^3))$ boxes for horizontal items, which only contain items with the same rounded width.

□

Lemma 5.29. *It is possible to place the small items inside the boxes generated by Lemma 5.24 and the boxes generated by Lemma 5.28 and one extra box with width W and height at most $2\varepsilon^6$OPT.*

Proof. The free area inside the boxes from the partition form Lemma 5.22 for horizontal tall and vertical items is at least as large as the total area for the small items since the small items where contained in the optimal packing area. By Lemmas 5.23 and 5.28 we generate at most $\mathcal{O}(1/(\varepsilon^5\delta^2))$ empty boxes, which we can use to place the small items. These boxes have a total area that is at least as large as the empty space in the original boxes $\mathcal{B}_{\mathcal{H}}$ and $\mathcal{B}_{\mathcal{T}\cup\mathcal{V}}$ from Lemma 5.22. Hence the total area of these empty boxes is at least as large as the area of the small items.

Let $\mathcal{B}_{\mathcal{S}}$ be the set of boxes and $|\mathcal{B}_{\mathcal{S}}| = c/\delta^2\varepsilon^5$ for some constant $c \in \mathbb{N}$. We prove that we only need a small extra box to place all the items with the NFDH algorithm into these boxes.

First, we discard any box with height less than μOPT or width less than μW. The total area of each discarded box is at most μWOPT. Let us consider a box B with height and width larger than μOPT or μW respectively. In each shelf we use for the NFDH-Algorithm, we cannot use a total width of at most μW to place the items. Furthermore, the last shelf has a distance of at most μOPT to the upper border of the box. Additionally, the free area between the shelfs has a total area of at most $\mu\text{OPT} \cdot w(B)$. Therefore, the total free area in B is at most $\mu W \cdot h(B) + 2\mu\text{OPT} \cdot w(B) \leqslant 3\mu W\text{OPT}$. As a result, the total area of items that could not be placed inside the boxes is at most $3\mu W\text{OPT} \cdot c/\delta^2\varepsilon^5$. Since $\mu \leqslant \varepsilon^{11}\delta^2/k$ for some suitable constant k, it holds that $3\mu W\text{OPT} \cdot c/\delta^2\varepsilon^5 \leqslant \varepsilon^6 W\text{OPT}$ when choosing $k \geqslant c$.

These items can be place with Steinberg's algorithm [100] into a box with width W and height $2\varepsilon^6$OPT since they have a height of at most μOPT. □

In the last step, we prove that it is possible to place the medium sized items $\mathcal{M}$.

Lemma 5.30. *It is possible to place the medium items $\mathcal{M}$ into a box width W and height at most* $2\varepsilon\text{OPT}$

Proof. First, we sort them by their processing time. Afterward, we use the NFDH algorithm to place the jobs. We know that $\text{area}(\mathcal{M})$ is bounded by $\varepsilon^{13}W\text{OPT}$ and $p_{\max}$ is bounded by εOPT. Therefore, by Lemma 2.2, we can place these items with a packing height of at most $\text{NFDH}(\mathcal{M}) \leqslant 2\varepsilon^{13}\text{OPT} + \varepsilon\text{OPT} \leqslant 2\varepsilon\text{OPT}$.

□

5.5 Algorithms

In this section, we describe the three algorithms for Strip Packing Without Rotations, Strip Packing With Rotations, and Scheduling contiguous Moldable Jobs. Each optimal solution of these three problems can be rearranged such that the structure looks like the structure in Lemma 5.24. The algorithms all work roughly the same. First, we determine an upper bound for the approximation. Afterward, we use a binary search framework to find a $(5/4 + \varepsilon)$ approximation. The routine called by the framework guesses the structure of the packing and tests with a dynamic program if the guess is feasible.

5.5.1 Strip Packing Without Rotations

The steps of the algorithm can be summarized as follows:

1. Define $\varepsilon' := 1/\lceil 10/\varepsilon \rceil$, a lower bound
$$T := \max\{\text{area}(\mathcal{I})/W, \max\{h(i) | i \in \mathcal{I}\}\}$$
and an upper bound $2T$ for the approximation and use these bounds to round and scale the item heights to values in $\{1, \ldots, n/\varepsilon'\}$. As a result we gain that OPT_{scaled} is an integer in $\{n/\varepsilon', \ldots, 2n/\varepsilon' + n, \}$.
2. Try values $T' \in \{n/\varepsilon', \ldots, 2n/\varepsilon' + n, \}$ for the optimum in a binary search fashion and for each tested value perform the following steps.
3. For $\text{OPT} = T'$, $\varepsilon = \varepsilon'$, and the rounded and scaled instance perform the simplification steps as described in Section 5.4. More precisely: find the

correct values for δ and μ, and round the heights of all items with a height taller than δOPT with the techniques from Lemma 3.2, round the horizontal items using linear grouping as described in Lemma 5.28.
4. Try each possible partition of the area $W \times (5/4 + 5\varepsilon')$OPT into the at most $\mathcal{O}(1/(\delta^3 \varepsilon'^5))$ boxes from Lemma 5.24. For each of these partitions perform the following steps:
5. Using a dynamic program try to place the vertical and horizontal items inside their boxes. If this fails the partition must have been wrong and we try the next partition.
6. If the vertical and tall items could be placed inside the boxes, try to place the horizontal items using the algorithm from Lemma 5.28. If this fails discard the guessed partition. Otherwise save the packing and try the next smaller value for T' in binary search fashion.
7. If all the partitions into boxes fail, try the next larger value for T' in binary search fashion.
8. Finally (after the binary search for the correct T') place the small items inside their boxes using NFDH and place the medium sized items on top of the packing using Steinberg using the best packing found. Return the packing.

In the following we argue the correctness of these steps and go into more detail.

Step 1: A first rounding step. Given a value $\varepsilon \in (0, 1]$ and an instance I, we define $\varepsilon' := \min\{1/4, 1/\lceil 10/\varepsilon \rceil\}$ and use this ε' instead of ε in the following steps. We estimate the optimal packing height OPT by using that Steinberg's algorithm [100] can place all items into a strip with height of at most $2T := 2\max\{\text{area}(\mathcal{I})/W, \max\{h(i) | i \in \mathcal{I}\}\}$. Therefore, we know that $\text{OPT} \in [T, 2T]$.

Next we round the heights of the items arithmetically by introducing a small error. We round the item heights to multiples of $\varepsilon T/n$ and reduce the number of different heights to n/ε. By Lemma 3.1 this adds at most εT to the optimal packing height. We define the height of the optimal packing for this rounded instance as $\text{OPT}_{rounded}$ and know that $\text{OPT}_{rounded} \leqslant \text{OPT} + \varepsilon T \leqslant (1 + \varepsilon)\text{OPT}$. Without making an additional rounding error, we can assume that the items have a height in $\{1, 2, \ldots, n/\varepsilon\}$, by scaling the rounded heights with $n/(T\varepsilon)$ and scale them back when constructing

the packing. Using this scaling, we know that $\mathrm{OPT}_{scaled} \in \{n/\varepsilon, n/\varepsilon + 1, \ldots, 2n/\varepsilon + n\}$ since $T \leqslant \mathrm{OPT} \leqslant \mathrm{OPT}_{rounded} \leqslant \mathrm{OPT} + \varepsilon T \leqslant 2T + \varepsilon T$ and hence $n/\varepsilon \leqslant \mathrm{OPT}_{scaled} \leqslant 2n/\varepsilon + n$.

Step 2: Dual Approximation To find the $(5/4 + \varepsilon)$OPT approximation we use the dual approximation framework introduced by Hochbaum and Shmoys [49]. Given a value T, we can calculate a packing with height at most $(5/4 + \varepsilon)T$ or decide that there is no packing with height T. To find the smallest value for T where it is possible to find a packing with height $(5/4 + \varepsilon)T$ we can try all the appropriate values for T in $\mathcal{O}(n/\varepsilon)$. The other option is to try these values in a binary search fashion such that we are searching for the smallest value from this set such that the residual algorithm finds a packing. This search for the optimal T can then be done in $\mathcal{O}(\log(n/\varepsilon))$ using the dual approximation framework.

Step 3: Performing the simplification steps. Next, we compute the values δ and μ with the properties from Lemma 5.20 while assuming $\mathrm{OPT} = T$ for some $T \in \{n/\varepsilon, \ldots, 2n/\varepsilon + n\}$. Knowing these values, we partition and round the items accordingly, see Lemma 3.2. As a trick to maintain integer sizes for the item heights, we can scale the instance with $1/(\varepsilon\delta) \in \mathbb{N}$ before rounding the items with Lemma 3.2. In this way $\varepsilon^x T_{scaled}$ will be integral for each $x \in \{1, \ldots, \log_\varepsilon 1/(\varepsilon\delta)\}$ since T_{scaled} has the form $T/\varepsilon\delta$. We will write T instead of T_{scaled} in the following.

Step 4: Guessing the partition into boxes Afterward, we guess the structure of the transformed optimal solution via Lemma 5.24 for items in $\mathcal{L}, \mathcal{M}, \mathcal{T}$ and $\mathcal{V}$ and the boxes from Lemma 5.22 for the horizontal items $\mathcal{H}$. This partition into boxes has a height of at most $(\frac{5}{4} + 5\varepsilon)(1 + \varepsilon)T$. For each of the at most $\mathcal{O}(1/(\delta^3\varepsilon^5))$ boxes, we guess the lower left corner and the upper right corner. For each box there are at most $\mathcal{O}((W/\delta\varepsilon)^2)$ possibilities to guess these positions since the x-coordinates are in $\{0, \ldots, W-1\}$, and the y-coordinates are in $\{0, \ldots, \mathcal{O}(1/\delta\varepsilon)\}$. Therefore, there are at most $(W/\delta\varepsilon)^{\mathcal{O}(1/\delta^3\varepsilon^5)}$ possible guessing steps. A guessed structure is feasible if we can place the items into the corresponding boxes.

Step 5: The dynamic program For each of these guessed partitions of the packing area into boxes, we test if we can place the items in $\mathcal{V} \cup \mathcal{T}$ inside these boxes by using the following dynamic program: For each rounded

height h we generate a vector $(w_{h,1}, \ldots, w_{h,k_h})$. It represents the k_h boxes for items $i \in \mathcal{V} \cup \mathcal{T}$ with height $h(i) = h$. Each entry is bounded by the width of the corresponding box.

For each rounded height h, the program enumerates all items $i \in \mathcal{V} \cup \mathcal{T}$ with height $h(i) = h$. We start with the vector $(w_{h,1} = 0, \ldots, w_{h,k_h} = 0)$. For each item $i \in \mathcal{T} \cup \mathcal{V}$ with $h(i) = h$, we make k_h copies of each so far generated vector $(w_{h,1}, \ldots, w_{h,k_h})$ and add the value $w(i)$ to a different component in each copy. If we enlarge one component above its maximal value or if we get the same vector a second time, we discard this vector. The guess of boxes for items with height h was feasible if there is still a valid vector after enumerating all the items with height h.

The width of each box for tall and vertical items $\mathcal{T} \cup \mathcal{V}$ is bounded by W. Therefore, we enumerate at most $W^{\mathcal{O}(k)}$ vectors for each item, where k is the total number of boxes. Therefore, the dynamic program has a running time of at most $n \cdot W^{\mathcal{O}(1/\delta^3\varepsilon^5)}$.

Step 6: Placing residual items If it is possible to place the items in $\mathcal{L} \cup \mathcal{V} \cup \mathcal{T} \cup \mathcal{M}_{\mathcal{V}}$, we place the horizontal items with the algorithm described in the proof of Lemma 5.28 in $(\log(1/\delta)/\varepsilon)^{\mathcal{O}(1/\varepsilon\delta^3)}$. After that we check of the total area of the boxes generated for the small items is large enough, i.e., larger than the total area of small items. This step fails if that is not the case.

The small $\mathcal{S}$ and medium items $\mathcal{M}$ are placed in the final step after the binary search for the right T'. The small items are placed into their corresponding boxes with the NFDH algorithm in $\mathcal{O}(n \log(n))$ and the medium items on top of the packing using the NFDH-Algorithm from [22]. These items add at most $2\varepsilon\mathrm{OPT}$ to the packing height since each item in $\mathcal{M}$ has a height of at most $\varepsilon\mathrm{OPT}$ and the total area of these items is bounded by $(\varepsilon^{13}/k)TW$.

Break condition If one of the steps to place the items, i.e., Step 5 or Step 6, fails for a guessed partition, the guess must have been wrong and we try the next partition. If we cannot find a packing for any of the partitions, the value T' was to low and we try the next value for T'.

When we scale back the heights of the items to multiples of $\varepsilon T/\varepsilon$ the final packing has a height of at most $(5/4 + 5\varepsilon)\mathrm{OPT}_{rounded} + \varepsilon^9\mathrm{OPT}_{rounded} +$

$2\varepsilon^6 \mathrm{OPT}_{rounded} + 2\varepsilon \mathrm{OPT}_{rounded} \leqslant (5/4 + 8\varepsilon)\mathrm{OPT}_{rounded}$. Since $\mathrm{OPT}_{rounded} \leqslant \mathrm{OPT} + \varepsilon T \leqslant (1+\varepsilon)\mathrm{OPT}$ it holds that the schedule we can find has a height of at most $(5/4 + 8\varepsilon)(1+\varepsilon)\mathrm{OPT} \leqslant (5/4 + 10\varepsilon)\mathrm{OPT}$. Hence, if we use the value ε' instead of ε for these steps, the generated packing has a height of at most $(5/4 + 10\varepsilon')\mathrm{OPT} \leqslant (5/4 + \varepsilon)\mathrm{OPT}$.

Using $\delta \geqslant \varepsilon^{2^{\mathcal{O}(1/\varepsilon^{13})}}$, the running time of the algorithm can be bounded by

$$\begin{aligned}
&\mathcal{O}(n\log(n)) + \mathcal{O}(\log(n/\varepsilon)) \cdot (W/\delta\varepsilon)^{\mathcal{O}(1/\delta^3\varepsilon^5)} \\
&\qquad \cdot (n \cdot W^{\mathcal{O}(1/\delta^3\varepsilon^5)} + (\log(1/\delta)/\varepsilon)^{\mathcal{O}(1/\varepsilon\delta^3)}) \\
&\qquad \leqslant \mathcal{O}(n\log(n)) \cdot W^{1/\varepsilon^{2^{\mathcal{O}(1/\varepsilon^{13})}}} \\
&\qquad \leqslant \mathcal{O}(n\log(n)) \cdot W^{\mathcal{O}_\varepsilon(1)}.
\end{aligned}$$

5.5.2 Strip Packing With Rotations

In this scenario each item either can be positioned non rotated (rotation = 0 degrees) or it can be placed rotated by 90 degrees (rotation = 90 degrees). However, the items in an optimal solution can be rounded shifted and reordered as the items in an optimal packing for Strip Packing Without Rotations and hence the structural Lemma 5.24 holds for optimal solutions to this problem as well. Nevertheless, we run into several problems when we try to use the same algorithm for these instances as for the instances for Strip Packing Without Rotations. Since we do not know which side of the items will be its width and which side will be its height in the optimal solution, we can no longer round the height of the item as we did before. Instead for each item, we will save several rounded values, i.e., both rounded heights and both rounded widths depending on the rotation of the item. Furthermore, the partition of the item set is no longer this simple, because an item can belong to one set in one rotation and to another in the other rotation. We will handle this issue by leaving this decision to an underlying dynamic program.

For simplicity of notation we will assume that $h(i) \leqslant W$ and $w(i) \leqslant W$ as well. Note that in theory one of these values could be larger than W but in that case we cannot rotate the item. However, this would only simplify

the matter since, as described before, we then can decide in which set this item is contained and and round the height of this item etc.

Dual approximation Similar as in the algorithm without rotations, we use a binary search framework to find the packing. We know that

$$T := \max\{\text{area}(\mathcal{I}), h_{\max} := \max\{\min\{h(i), w(i)\} | i \in \mathcal{I}\}\}$$

is a lower bound on the packing height, while $2T$ is an upper bound because of Lemma 2.3. (Note that this bound has to be slightly adapted, if there are non rotatable items. When considering a non rotatable item, the definition of $h_{\max}$ needs to take into account the maximum of height and width). Therefore, given an optimal solution we would like to round the heights of the items to multiples of $\varepsilon T/n$ as before in Lemma 3.1. Since we do not know which side of the item defines the height, we calculate the rounded (and scaled) values for both sides, but remember the original one as well. Using these rounded values for the heights of the items lengthens the schedule by at most εT and again we can assume that the optimal height of the schedule is one of the values $\{n/\varepsilon, n/\varepsilon + 1, ..., 2n/\varepsilon + n\}$.

Defining δ and μ In the next step, we guess the values of δ and μ since, as discussed, we cannot determinate them because we do not know which item will be in which set in the optimal solution. There are at most $1/f(\varepsilon) = \mathcal{O}(1/\varepsilon^{13})$ possibilities for these values. Knowing these values, we can decide for a given item and its rotation (0 degrees or 90 degrees) in which set $\mathcal{L}, \mathcal{V}, \mathcal{H}, \mathcal{T}, \mathcal{M}, \mathcal{M}_V$ or $\mathcal{S}$ this item is contained in the optimal solution.

Rounding the horizontal items In the next step, we consider the horizontal items. We want to round the horizontal items similar as in Lemma 5.28. However, since we do not now in advance which items the set $\mathcal{H}$ contains, we have to guess the rounded widths: First, we guess the total height $h(\mathcal{H})$. It holds that $h(\mathcal{H}) \in \{0, 1, \ldots, 2n/\varepsilon + n\}$ since by the rounding and scaling each horizontal item has an integral height and the total packing height is bounded by $2n/\varepsilon + n$. Therefore we need at most $\mathcal{O}(n/\varepsilon)$ guessing steps to determine the correct height $h_{\mathcal{H}}$ of the stack of horizontal items.

After this guess of the height of the stack, we guess the at most $1/\varepsilon\delta^2$ items and their rotation (0 degrees or 90 degrees) that define the rounded widths of the groups. There are at most $(2n)^{1/\varepsilon\delta^2}$ possibilities to guess

these items, due to the rotation. We determine the height of a group as $h_G := \lfloor \varepsilon\delta^2 h_{\mathcal{H}} \rfloor$. Using this height there is at most one item per stack that cannot be placed inside the group since all the items have an integral height. Therefore, we place the items which we have guessed to define the widths of the groups on top of the packing in the very last step of the algorithm. Since these items have a height of at most μOPT and since $\mu \leqslant \delta^2\varepsilon^{13}$, these items add at most $\mu\text{OPT}/\varepsilon\delta^2 \leqslant \varepsilon^{12}\text{OPT}$ to the packing height. Now, we can assume in the dynamic program that each rounded width occurs with a total height of at most $h_G \in \mathcal{O}(\delta^2 n)$.

Rounding the height of items with height larger than δOPT In the following let $\tilde{h}(i)$ and $\tilde{w}(i)$ be the rounded height and rounded width for an item for a given rotation (0 degrees or 90 degrees). If an item is contained in the set $\mathcal{L} \cup \mathcal{V} \cup \mathcal{T} \cup \mathcal{M}_V$, its height is a multiple of $i\varepsilon^x\text{OPT}$ for $i \in \{1/\varepsilon, \ldots, 1/\varepsilon^2\}$ and some $x \in \mathbb{N}$, while its width remains original. If the item is in $\mathcal{H} \cup \mathcal{M} \cup \mathcal{S}$, its height is an integer from $\{1, \ldots, n/\varepsilon\}$ and its width is either its original or one of the at most $1/\varepsilon\delta^2$ guessed width values. Note that if we scale all the heights with $1/\varepsilon\delta$ all the considered heights are integral.

Guessing the partition from Lemma 5.24 In this step, we guess the structure from Lemma 5.24 using a height of at most $(\frac{5}{4} + 5\varepsilon)\text{OPT}_{scaled}$. First, we guess which items are the at most $\mathcal{O}(1/\delta^2\varepsilon)$ large $\mathcal{L}$ and medium vertical $\mathcal{M}_\mathcal{V}$ items and their rotations in the optimal packing. This can be done in $n^{\mathcal{O}(1/\delta^2\varepsilon)}$. Afterward, we guess their positions, i.e., the position of the lower left corner of these items. For the x position there are at most W possibilities, while for the y-position there are at most $\mathcal{O}(1/\varepsilon\delta)$ possibilities. Hence we can guess the positions of all the large and medium vertical items in $(W/\varepsilon\delta)^{\mathcal{O}(1/\varepsilon\delta^2)}$. Next, we guess the positions of the at most $\mathcal{O}(1/\varepsilon^5\delta^3)$ other boxes using $(W/\varepsilon\delta)^{\mathcal{O}(1/\varepsilon^5\delta^3)}$ possibilities since they start and end at multiples of $1/\varepsilon\delta$.

The modified dynamic program Again we check with a dynamic program if the guess is feasible. We introduce the vectors $w_{h_i} = (w_{h_i,1}, \ldots, w_{h_i,k_{h_i}})$ and a vector $h = (h_{w_1}, \ldots, h_{w_{1/\varepsilon\delta^2}})$ for each rounded height h_i and each rounded width w_j respectively. Furthermore, we introduce two values a_s and a_m. These values represent the total area of the small items $\mathcal{S}$

and medium sized items $\mathcal{M}$ receptively. In the dynamic program, we consider a sequence of sets D_i, $i \in \mathcal{I}$, containing vectors of the form $(h_{w_1}, \ldots, h_{w_{1/\varepsilon\delta^2}}, w_{h_1}, \ldots w_{h_{1/\delta\varepsilon}}, a_s, a_m)$. D_0 contains just the vector that is filled with zeros. Iterating over the set of items for each item $i \in I$, we determine for both possible rotations the set it would be contained in. For both rotations, we do the following steps to the set D_{i-1}.

- ▷ If the ith item is in $\mathcal{V} \cup \mathcal{T}$, we make $k_{\tilde{h}(i)}$ copies of each vector in D_{i-1}. In each copy, we add the items width $w(i)$ to another entry of the vector $w_{\tilde{h}(i)}$ and add it to the set D_i.
- ▷ If the ith item is in $\mathcal{H}$, we make a copy of each vector in D_{i-1}. In each copy, we add its height $\tilde{h}(i)$ to a the entry $h_{\tilde{w}(i)}$ and add it to the set D_i.
- ▷ If the ith item is in $\mathcal{S}$, it holds that $\tilde{h}(i) = l_i\varepsilon(1+\varepsilon)T/n$ for some $l_i \in \{1, \ldots, n/\varepsilon\}$. We make a copy of each vector in the set D_{i-1} and add $l_i \cdot w(i)$ to the value a_s in each vector and add it to the set D_i.
- ▷ If the ith item is in $\mathcal{M}$, we do the same as in the previous case with the difference that we add the value $l_i \cdot w(i)$ to a_m.

If a value in the vector exceeds its boundary, we discard this vector. Furthermore, if a specific vector is created a second time, we save it just once and discard the newly generated vector.

The values $h_{i,j}$ are bounded by the height of the rounded group, i.e., they are bounded by $\mathcal{O}(\delta^2 n)$, while the values $w_{i,j}$ are bounded by the guessed width of the corresponding box, i.e., ultimately they are bounded by W. Let A_s be the total area of the boxes for small items. Note that since each small item has an integer width and an integer height and the total area of items is bounded by $W \cdot n/\varepsilon$, the total area of small items a_s can have any integer size up to Wn/ε. Furthermore, the total area of the medium sized items is bounded by $\varepsilon^{13}W\text{OPT}$, where $\text{OPT} \in \{n/\varepsilon, \ldots, 2n/\varepsilon + n\}$. t Therefore, we bound the size of a_m by this value.

Let us analyze the running time of the linear program. The entries $w_{i,j}$ can take at most W different values, the entries $h_{i,j}$ can take at most $\delta^2 n$ different values and the entries a_s and a_n can take at most $\mathcal{O}(nW/\varepsilon)$ different values. Since each vector in D has at most $\mathcal{O}(1/\varepsilon^5\delta^3)$ entries,

the running time of the dynamic program is bounded by $(W)^{\mathcal{O}(1/\varepsilon^5\delta^3)} \cdot (\delta^2 n)^{\mathcal{O}(1/\varepsilon\delta^2)} \cdot \mathcal{O}((nW/\varepsilon)^2)$.

Placing the horizontal items After the processing of the dynamic program, we try for each feasible solution of this program whether it is possible to place the horizontal and small items. The horizontal items are placed with the algorithm from Lemma 5.28. If the resulting set of boxes for small items is large enough, we save this solution and proceed with trying the next smaller value for T'.

When we have found the correct value for T', we place the small and medium sized items using the NFDH-algorithm [22]. The overall running time of the algorithm dominated by the running time of the dynamic program and hence is bounded by $(Wn)^{1/\varepsilon^{2^{\mathcal{O}(1/\varepsilon^{13})}}}$.

5.5.3 Contiguous Moldable Task Scheduling

Again we start with a binary search framework. We use the results from Ludwig and Tiwari [83] to find an estimate U for the makespan of the optimal schedule and define $T := U/2$, i.e., we know that OPT $\in [T, 2T]$. Afterward, we use the same rounding and scaling as before, i.e., we consider only processing times in $\{1, \ldots, n/\varepsilon\}$. Then we use the binary search framework to find the correct value for T'. The guessing steps work the same as in section 5.5.2 including the guessing of rounded width of horizontal jobs. In the following, we describe how to adjust the dynamic program for this scheduling version.

Let $\psi_j(p) \in M_j$ be the minimal number of processors needed for job $j \in \mathcal{J}$ to have a processing time of at most p. We iterate over the non-placed jobs in arbitrary order. Let $j \in \mathcal{J}$. To generate the set D_j in the dynamic program we do the following steps:

- First, we determine for each of the at most $1/\varepsilon^2$ large processing times $p > 1/4\text{OPT}$ the number of needed processors $\psi_j(p)$. If this number is smaller than δW, it is feasible to schedule this job as tall job and we try each possible box for this job and processing time in the dynamic program. Otherwise the job cannot have this processing time for the current choice of large and medium sized items.

- ▷ Afterward, we determine for each of the at most $1/\varepsilon\delta$ large processing times p with $1/4\text{OPT} \geqslant p > \delta\text{OPT}$ the number of needed processors $\psi_j(p)$. If this number is smaller than μW, it is feasible to schedule this job as vertical job and we try each possible box for this job and processing time in the dynamic program. Otherwise the job cannot have this processing time for the current choice of large and medium sized items.
- ▷ Next, we consider the guessed number of processors for horizontal jobs. For two consecutive rounded numbers of machines m_i, m_{i+1}, we determine for which number of machines in $M_j \cap \{m_i, \ldots, m_{i+1}\}$ the job has the smallest processing time $p_j(m_i)$. If it is smaller than μOPT, the job qualifies to be scheduled as a horizontal job. Hence, we make a copy of each vector in D_{j-1} and add the value $\lceil np_j(m_i)/\varepsilon\text{OPT}\rceil$ to the value $h_{m_{i+1}}$.
- ▷ After that, we try to schedule the job as a small job. We determine the smallest processing time if the job uses less than μW machines. If this processing time is smaller than $\mu(1+\varepsilon)^2T$, the job can be scheduled as a small job and we try this possibility too, by adding its work to a_s to each vector from the set D_{j-1}.
- ▷ Last, we test if the job can be scheduled as a medium job. We determine the smallest processing time if the job uses between μW and δW processors. If this processing time is smaller than εT, the job can be scheduled as medium job and we add its work to a copy of each vector from the set D_{j-1}. On the other hand, the job can be scheduled as medium job, if its processing time is between $\mu(1+\varepsilon)^2T$ and $\delta(1+\varepsilon)^2T$. hence, we determine the minimal number of processors so that the job has a processing time between $\mu(1+\varepsilon)^2T$ and $\delta(1+\varepsilon)^2T$ and we add its work to a copy of each vector from the set D_{j-1}.

Each vector in the dynamic program has at most $\mathcal{O}(1/\varepsilon^5\delta^3)$ entries. The values of the entries are bounded by W, $\delta^2 n$ and nW/ε. For each job we try at most $\mathcal{O}(1/\varepsilon^5\delta^3)$ entries. Therefore, $(Wn)^{1/\varepsilon^{2^{\mathcal{O}(1/\varepsilon^{13})}}}$ is an upper bound on the running time of the dynamic program and the entire algorithm.

5.6 Conclusion

In this chapter, we have nearly closed the gap between the lower bound of the approximation ratio and best approximation ratio for the problems pseudo-polynomial Strip Packing with and without rotations and Contiguous Moldable Task Scheduling. The question whether we actually can find algorithms with approximation ratio exactly $5/4$ remains still open.

Concerning polynomial algorithms, there is still a large gap between the lower bound for an absolute approximation ratio of $3/2$ unless $P = NP$ and $5/3 + \varepsilon$ which is the best absolute approximation ratio achieved so far [44].

Furthermore, an interesting question is whether we can find better approximations for the case of monotonic moldable jobs. While the lower bound of $5/4$ holds for the general case of scheduling contiguous moldable jobs in pseudo-polynomial time, a PTAS could be possible if we consider monotonic jobs.

Chapter 6

An AFPTAS for Single Resource Constraint Scheduling

In this chapter, we consider the scheduling problem Single Resource Constraint Scheduling. We are given a set of jobs that require a certain amount of an additional resource and have to be scheduled on identical machines minimizing the makespan, while at every point in time a given capacity for the resource is not exceeded. This problem contains Bin Packing With Cardinality Constraint as a special case and, therefore, it is strongly NP-complete even for a constant number of machines of at least three, which can be proven by a reduction from 3-Partition. Furthermore, if the number of machines is part of the input, we cannot hope for an approximation algorithm with absolute approximation ratio smaller than 3/2.

For this problem, we present an AFPTAS. This algorithm provides for any $\varepsilon > 0$ a schedule of length at most $(1 + \varepsilon)$ times the optimum plus an additive term of $\mathcal{O}(p_{\max} \log(1/\varepsilon)/\varepsilon)$, and its running time is polynomial bounded in $1/\varepsilon$ and the input length. Up to now only approximation algorithms with absolute approximation ratios were known. Furthermore, the AFPTAS for resource constrained scheduling on identical parallel machines directly improves the additive term of the best AFPTAS for Bin Packing With Cardinality Constraint so far.

A preliminary version of the results of this chapter was published as [58]. Compared to that version, we improved the additive term of the AFPTAS. This improvement appeared in the journal TALG [57]. In the version presented in this chapter the running time of the algorithm was

further improved compared to the version in [57].

6.1 Results

Remember in the Single Resource Constraint Scheduling problem, we are given a set of n jobs $\mathcal{J}$, $m \in \mathbb{N}$ identical parallel machines and a renewable resource with limit $R \in \mathbb{N}$. Each job $j \in \mathcal{J}$ has a processing time $p(j) \in \mathbb{N}$. To be processed, it requires an amount of $r(j) \in \mathbb{N}$ of the given resource and one machine. A schedule of these jobs is given by a mapping $\tau : \mathcal{J} \to \mathbb{N}_{\geqslant 0}$ from jobs to starting times. It is *feasible*, if at each point in time $t \in \mathbb{N}$ there are enough machines to schedule the jobs and the total resource requirement of jobs scheduled at t does not exceed the resource limit R, i.e.:

$$\forall t \in \mathbb{N} : \sum_{j : t \in [\tau(j), \tau(j)+p(j))} r(j) \leqslant R \tag{6.1}$$

$$\forall t \in \mathbb{N} : |\{j \in \mathcal{J} | t \in [\tau(j), \tau(j) + p(j))\}| \leqslant m \tag{6.2}$$

The objective is to find a feasible schedule $\tau : \mathcal{J} \to \mathbb{N}_{\geqslant 0}$ minimizing the makespan $M := \max_{j \in \mathcal{J}}(\tau(j) + p(j))$.

This problem arises naturally in different contexts, e.g., in highly parallelized computing where simultaneously active jobs share some memory, or in production logistics where additional personnel may speed up certain tasks. From a theoretical perspective on the other hand, the problems are sensible generalizations of problems like scheduling on identical parallel machines or bin packing, and have already been studied in 1975 [34].

The problem is NP-hard and therefore there is little hope to find optimal solutions efficiently. We can even rule out the possibility of an algorithm with approximation ratio strictly smaller than $3/2$, unless $\mathrm{P} = \mathrm{NP}$, by a straight forward reduction from the Partition Problem since this problem contains Bin Packing as a special case, see Section 2.3. Therefore, a PTAS is not possible, while an approximation scheme with respect to the *asymptotic* approximation ratio still can be achieved.

To this point this problem has only been studied with respect to the absolute approximation ratio. The algorithm with the best ratio so far was presented by Niemeier and Wiese [92] and has an absolute approximation

ratio of $(2+\varepsilon)$. However, for Bin Packing With Cardinality Constraint, which is contained in Single Resource Constraint Scheduling as the special case where all the processing times are equal to one, asymptotic ratios have been studied. The so far best algorithm is an AFPTAS, which was presented by Epstein and Levin [30].

In this chapter, we present an AFPTAS for Single Resource Constraint Scheduling.

Theorem 6.1. *Let I be an instance of Single Resource Constraint Scheduling and $p_{\max}$ the biggest occurring processing time in I. For each $\varepsilon \in (0,1)$ there is an algorithm which computes a schedule with makespan bounded by*

$$(1+\varepsilon)\mathrm{OPT}(I) + \mathcal{O}(p_{\max}\log(1/\varepsilon)/\varepsilon),$$

and has a running time that of at most $\mathcal{O}(n/\varepsilon + \log(m)^3/\varepsilon^7)$.

Note that this result directly improves the additive term of the AFPTAS for Bin Packing with Cardinality Constraints by Epstein and Levin [30], which has an additive term that is exponential in $1/\varepsilon$. For a more detailed overview on the work related to the problem Single Resource Constraint Scheduling, we refer to Section 2.3.

Methodology and Organization of this Chapter

In the following, we give a brief overview on how we achieve this result. We partition the set of jobs into wide and narrow jobs based on their resource requirements and apply linear grouping for the wide jobs. To handle the narrow jobs, we adapt the notion of windows that was introduced by Epstein and Levin [30]. However, we manage to bound the number of windows to be in $\mathcal{O}(1/\varepsilon^2)$ via a second elaborate rounding step. This step leads to an AFPTAS with additive term $\mathcal{O}(p_{\max}/\varepsilon^2)$ which can be found in Section 6.2. Using geometric instead of linear grouping for the wide jobs and a new argument for the reduction of the windows, we are able to further improve the additive term to $\mathcal{O}(p_{\max}\log(1/\varepsilon)/\varepsilon)$. This algorithm is described in Section 6.3.

As a side note, we would like to point out that our techniques also work for the scenario, where the resource needs to be allocated contiguously, that is, the resource is represented by an interval of length R and each

job is assigned a subinterval whose length corresponds to the resource requirement assigned to the job. This scenario corresponds to the problem of Strip Packing With Cardinality Constraint, where the cardinality constraint is given by the requirement that at most m jobs may executed at each time step. Hence, we provide an AFPTAS for this problem as well.

6.2 A First AFPTAS

In this section, we introduce an algorithm with additive term $\mathcal{O}(p_{\max}/\varepsilon^2)$ and discuss the modifications we make to achieve the improved additive term in Section 6.3. In the following, we will call a job's resource requirement its width and its processing time its height and do the same for all introduced structures as (generalized) configurations and windows.

First, we will give a slightly more detailed high-level view of the algorithm. The steps of the algorithms are described in detail in the following sub sections. We differentiate two cases: first $1/\varepsilon < m$ and second $1/\varepsilon \geqslant m$, where the case $1/\varepsilon < m$ is more involved.

In the first case, the set of jobs $\mathcal{J}$ is partitioned into wide and narrow jobs, where wide jobs have a larger resource requirement than narrow jobs. The resource requirement of the wide jobs is rounded by linear grouping such that we have to deal with just $\mathcal{O}(1/\varepsilon^2)$ different sizes. For this rounded instance the algorithm computes a preemptive schedule using a configuration LP. Broadly speaking, a configuration is a selection of jobs that can be processed at the same time. After that, each configuration is partitioned in the wide job part and the narrow job part. The spare area (resource and machines) not used by the wide jobs will be called window, following the notation in [30]. In simplified terms, a window can be seen as the residual space (resource and machine number) that is left by a configuration of wide jobs. By constructing a preemptive schedule first, instead of solving the LP by Levin and Epstein [30] directly, we manage to choose each window's width more adjusted to the given instance. This adjustment improves the number of operations and the makespan of the solution slightly. Nevertheless, the crucial step is to reduce the number of different window sizes such that it just depends on ε by simultaneously adding (not too much) processing time to the schedule. This is achieved by

a further grouping step. After a solution with reduced number of windows is found, an integral schedule can be computed by adding some processing time to the schedule.

In the second case, partitioning the jobs by resource requirement is not mandatory and all job resource requirements can be rounded to few sizes. This simplifies the problem such that we can generate a schedule directly after generating the preemptive solution.

6.2.1 Summary for the case $1/\varepsilon < m$

Let an instance $I = (\mathcal{J}, m, R)$ and $\varepsilon > 0$, with $1/\varepsilon$, be given. In the following we sometimes write $\mathcal{J}(I)$ to refer to the set of jobs in the instance. We assume w.l.o.g. that $1/\varepsilon \in \mathbb{N}$ and $m < n$ since, otherwise, we have the problem of Scheduling Parallel Tasks, for which an AFPTAS is already known [101, 15]. The algorithm can be summarized as follows:

(i) Define ε' such that it is the largest value with $1/\varepsilon' \in \mathbb{N}$ and $\varepsilon' \leqslant \varepsilon/6$. Compute $p_{\max} = \max\{p(j) | j \in \mathcal{J}(I)\}$.

(ii) Construct a rounded instance $I_{\sup,\varepsilon'}$ with at most $1/\varepsilon'^2$ wide jobs using linear grouping.

(iii) Solve a configuration linear program LP_{split} to find a schedule x_{split} for $I_{\sup,\varepsilon'}$, where we are allowed to schedule the jobs as splittable jobs. This solution uses at most $|\mathcal{J}(I_{\sup,\varepsilon'})| + 1$ configurations and has a makespan of at most $(1+\varepsilon')^2 \text{OPT}(I)$.

(iv) Transform the obtained configurations into generalized configurations, which are composed of a configuration part for wide and a window part for narrow jobs, which reduces the number of used configurations to $\min\{|\mathcal{J}(I_{\sup,\varepsilon'})| + 1, (\frac{1}{\varepsilon'^2})^{1/\varepsilon'}\}$.

(v) Reduce the number of different windows further using a grouping step which lengthens the solution by a factor of at most $(1+\varepsilon')$. Generate a solution for $I_{\sup,\varepsilon'}$ and a generalized configuration linear program LP_W, which uses at most $5 + 3/\varepsilon'^2$ configurations and has a makespan of at most $(1+\varepsilon')^3 \text{OPT}(I)$.

(vi) Given this solution for $I_{\sup,\varepsilon'}$ and LP_W, generate an integral schedule for I obtaining an overall makespan of at most

$$(1+\varepsilon')^4 \text{OPT}_{\text{split}}(I) + (3/\varepsilon'^2 + 1/\varepsilon' + 6) p_{\max}.$$

6.2.2 Rounded Instance – Step (ii)

In this step, we describe how to generate a rounded instance $I_{\text{sup},\varepsilon}$ for a given instance I and an $\varepsilon \in \mathbb{R}$ with $1/\varepsilon \in \mathbb{N}$. The algorithm will use the rounded instance $I_{\text{sup},\varepsilon'}$.

We partition the given set of jobs $\mathcal{J}$ into a set of wide jobs $\mathcal{J}_{W\varepsilon} := \{j \in \mathcal{J} | r(j) \geqslant \varepsilon R\}$ with resource requirement at least εR and a set of narrow jobs $\mathcal{J}_{N\varepsilon} := \mathcal{J} \backslash \mathcal{J}_{W\varepsilon}$. We round the resource requirement of the wide jobs by linear grouping (a method introduced by Fernandez de la Vega and Lueker [106] for Bin Packing and extended by Kenyon and Rémila [77] to Strip Packing) as described in the following. (See also Section 3.2.) We interpret each job j as a rectangle with width $r(j)$ and height $p(j)$ and place these rectangles on top of each other, such they build a vertical stack ordered by increasing resource requirement from bottom to top, see Figure 6.1. Let $P_W := p(\mathcal{J}_{W\varepsilon})$ be the height of the stack. Consider the horizontal lines at height $i\varepsilon^2 P_W$. Each job intersecting with one of these lines is divided at the intersection and split into two new jobs. Define by $\mathcal{J}_{W,\varepsilon,i}$ the set of (possible split) jobs, which lie between the lines $(i-1)\varepsilon^2 P_W$ and $i\varepsilon^2 P_W$. We get $G := 1/\varepsilon^2$ many sets $\mathcal{J}_{W,\varepsilon,i}$, called groups, where $\mathcal{J}_{W,\varepsilon,G}$ is the group containing the jobs with the largest resource requirement. Define by $R_i := \max\{r(j) | j \in \mathcal{J}_{W,\varepsilon,i}\}$ the largest resource requirement in group i. Note that $R_i \leqslant R_{i+1}$ for all $i < G$. By Lemma 3.3, this rounding can be done in $\mathcal{O}(|\mathcal{J}_{W\varepsilon}| \log(1/\varepsilon))$.

The rounded instance $I_{\text{sup},\varepsilon}$ is generated as follows: Let $\mathcal{J}_{\text{sup},\varepsilon}$ be the set containing all jobs from $\mathcal{J}_{N\varepsilon}$ and one additional job for each $i \in \{1, \dots G\}$ with processing time $p(\mathcal{J}_{W,\varepsilon,i})$ and resource requirement R_i and let be $I_{\text{sup},\varepsilon} := (\mathcal{J}_{\text{sup},\varepsilon}, m, R)$. We denote by $\mathcal{J}_{\text{sup},W,\varepsilon}$ the set of wide jobs in $I_{\text{sup},\varepsilon}$.

6.2.3 Splittable Schedule – Step (iii)

The next step of the algorithm is to find a schedule for our rounded instance, where we schedule the jobs as splittable jobs. Remember, when scheduling jobs as splittable ones, we are allowed to interrupt and restart each job at no cost. These jobs can be restarted at any time, even at times, where other parts of this job are still processed. We denote the optimum of a preemptive schedule as $\text{OPT}_{\text{split}}(I, \varepsilon)$ if the wide jobs are wider than

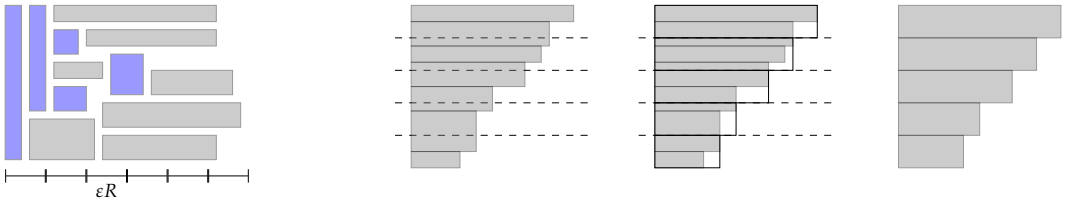


Figure 6.1. Rounding the Instance. Blue rectangles represent narrow jobs, while gray rectangles represent wide jobs. On the right, one can see the steps of the linear grouping.

εR. In this section, we prove that such a schedule of splittable jobs for any given instance $I = (\mathcal{J}, m, R)$ and given ε can be found in time polynomial in n and $1/\varepsilon$, see Lemma 6.2. Furthermore, in the end of this section, we prove that the makespan of an optimal preemptive solution of the rounded instance $I_{\text{sup},\varepsilon}$ can be bounded by the makespan of an optimal preemptive solution for I loosing a factor of $(1+\varepsilon)$, see Lemma 6.3.

A *configuration* $C \in \mathbb{N}^{\mathcal{J}}$ is a multiset of jobs which can be processed at the same point in time without violating the conditions of a feasible schedule. For a given configuration C the value $C(j)$ says how often the job j is contained in C. We allow $C(j) \in \{0,1\}$ if $j \in \mathcal{J}_{N,\varepsilon}$, and $C(j) \in \{0,\dots,1/\varepsilon\}$ if $j \in \mathcal{J}_{W,\varepsilon}$ since each job in $\mathcal{J}_{W,\varepsilon}$ has width at least εR and, therefore, it is not possible to schedule more than $1/\varepsilon$ of them at the same time. A configuration is *feasible* for a given instance I if $m(C) := \sum_{j\in\mathcal{J}} C(j) \leqslant m$ and $r(C) := \sum_{j\in\mathcal{J}} C(j)r(j) \leqslant R$. Denote by $\mathcal{C}_{I,\varepsilon}$ the set of all feasible configurations of I. An optimal solution of the following linear program $\text{LP}_{\text{split}}(I,\varepsilon)$ delivers an lower bound on an optimal schedule for any instance I.

$$\min \sum_{C\in\mathcal{C}_{I,\varepsilon}} x_C \tag{6.3}$$

$$\sum_{C\in\mathcal{C}_{I,\varepsilon}} C(j)x_C \geqslant p(j) \quad \forall j \in \mathcal{J} \tag{6.4}$$

$$x_C \geqslant 0 \quad \forall C \in \mathcal{C}_{I,\varepsilon} \tag{6.5}$$

The variable x_C denotes the processing time of configuration $C \in \mathcal{C}_{I,\varepsilon}$. Inequality 6.4 ensures that each job is scheduled.

Lemma 6.2. *For any Instance I and any $\varepsilon, \delta > 0$, we can find a basic solution $x_{\text{split},\varepsilon}$ to $\text{LP}_{\text{split}}(I, \varepsilon)$ with*

$$\sum_{C \in \mathcal{C}_{I,\varepsilon}} (x_{\text{split},\varepsilon})_C \leqslant (1+\delta)\text{OPT}_{\text{split}}(I, \varepsilon)$$

in $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + 1/\delta^2)(|\mathcal{J}_W|z + |\mathcal{J}_N| + mz'^2/\delta^2 + |\mathcal{J}|^{1.5356}))$ operations, where $z = \min\{1/\varepsilon, m\}$ and $z' = \min\{1/\delta, m\}$.

Proof. This configuration LP has the same form as the first configuration LP described in 3.3. Therefore by Lemma 3.9 we can find a basic solution to the above linear program in $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \delta^{-}2) \cdot (P(\mathcal{ABS}) + |\mathcal{J}|^{1.5356}))$, by transforming it to a Max-Min-Resource-Sharing problem.

In this approach $P(\mathcal{ABS})$ is the running time of the following problem: Given a profit function q that assigns a profit $q(j)$ to each job in $\mathcal{J}(I)$, find the configuration C that maximizes the profit of the contained jobs. For the set of configurations $\mathcal{C}_{I,\varepsilon}$, this problem can be formalized as follows:

$$\begin{aligned}
\max \sum_{j \in \mathcal{J}} \frac{q_j}{p(j)} a_j & \\
\sum_{j \in \mathcal{J}} a_j &\leqslant m \\
\sum_{j \in \mathcal{J}} r(j) a_j &\leqslant R \\
a_j \in \{0, 1\} \quad & \forall j \in \mathcal{J}_{N,\varepsilon} \\
a_j \in \{0, \ldots, 1/\varepsilon\} \quad & \forall j \in \mathcal{J}_{W,\varepsilon}
\end{aligned}$$

This ILP is equivalent to the ILP formulation of the problem Knapsack With Cardinality Constraint (kKP). This problem is similar to the Knapsack problem and has the additional constraint that at most k items are allowed to be put into the knapsack. In our case we have that $k = m$ since we can schedule at most m jobs at the same time. A difference in the formulation is that the wide jobs can be picked several times. There is an FPTAS by Mastrolilli and Hutter [86] to solve this problem. To use it, we have to duplicate each wide job $z = \min\{1/\varepsilon, m\}$ times since it can be scheduled up to z times at the same time, without violating any constraint. The computation of a $(1+\delta')$-approximate solution for a kKP instance (I, k, ε)

takes $\mathcal{O}(|I| + kz'^2/\delta'^2)$ operations, where $z' := \min\{k, 1/\delta'\}$. In our case we have $|I| = |\mathcal{J}_{W,\varepsilon}|z + |\mathcal{J}_{N,\varepsilon}|$, $k = m$ and $z' = \min\{m, 1/\delta'\}$, and hence we need at most $\mathcal{O}(|\mathcal{J}_{W,\varepsilon}|z + |\mathcal{J}_{N,\varepsilon}| + mz'^2\delta'^{-2})$ operations to find a solution a^* to the ILP. To get a configuration C^* we define $C^*(j) := a_j^*$ for each $j \in \mathcal{J}$.

As a consequence, the total running time to find the basic solution is bounded by $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + 1/\delta^2)(|\mathcal{J}_W|z + |\mathcal{J}_N| + mz'^2/\delta^2 + |\mathcal{J}|^{1.5356}))$, which concludes the proof. □

We now look at the relation between optimal solutions to I and $I_{\text{sup},\varepsilon}$:

Lemma 6.3. *It holds that* $\text{OPT}_{\text{split}}(I_{\text{sup},\varepsilon}, \varepsilon) \leqslant (1+\varepsilon)\text{OPT}_{\text{split}}(I, \varepsilon)$

Proof. Let a solution to $\text{LP}_{\text{split}}(I, \varepsilon)$ and $\text{LP}_{\text{split}}(I_{\text{sup},\varepsilon}, \varepsilon)$ each be given. Since each group $\mathcal{J}_{W,\varepsilon,i}$ has the same summed up processing time, we can split that large job in $I_{\text{sup},\varepsilon}$ with processing time $p(\mathcal{J}_{W,\varepsilon,i})$ and resource requirement R_i and schedule it instead of the jobs in $\mathcal{J}_{W,\varepsilon,i+1}$ in the solution to $\text{LP}_{\text{split}}(I, \varepsilon)$. The widest group cannot be scheduled inside other jobs, but we can shift this group on top of the schedule. This group has a makespan of at most $\varepsilon^2 P_W \leqslant \varepsilon\text{OPT}_{\text{split}}(I, \varepsilon)$. So for each solution to $\text{LP}_{\text{split}}(I, \varepsilon)$, we can generate a solution to $\text{LP}_{\text{split}}(I_{\text{sup},\varepsilon}, \varepsilon)$ which is lengthened by at most $\varepsilon\text{OPT}_{\text{split}}(I, \varepsilon)$, hence it holds that $\text{OPT}_{\text{split}}(I_{\text{sup},\varepsilon}, \varepsilon) \leqslant (1+\varepsilon)\text{OPT}_{\text{split}}(I, \varepsilon)$. □

Let us recapitulate the steps taken by the algorithm so far. In the first step it computed $I_{\text{sup},\varepsilon'}$. Next in (iii) - as described in this subsection - the algorithm computes $x_{\text{split},\varepsilon'}$ defined as the $(1+\varepsilon')$-approximate solution to $\text{LP}_{\text{split}}(I_{\text{sup},\varepsilon'}, \varepsilon')$ with at most $|\mathcal{J}_{\text{sup},\varepsilon'}| + 1$ non zero components. Since $|\mathcal{J}_{\text{sup},W,\varepsilon'}| \leqslant \varepsilon'^{-2}$ and $1/\varepsilon < m$, we can generate this solution in $\mathcal{O}(|\mathcal{J}_{\text{sup},\varepsilon'}|(\ln(|\mathcal{J}_{\text{sup},\varepsilon'}|) + \varepsilon^{-2})(|\mathcal{J}_{\text{sup},\varepsilon'}|^{1.5356} + m/\varepsilon^4))$ operations. By Lemma 6.3 we know that the solution $x_{\text{split},\varepsilon'}$ has a makespan of at most

$$\sum_{C \in \mathcal{C}_{I_{\text{sup},\varepsilon',\varepsilon'}}} (x_{pre,\varepsilon'})_C \overset{\text{L. 6.2}}{\leqslant} (1+\varepsilon')\text{OPT}_{\text{split}}(I_{\text{sup},\varepsilon'}, \varepsilon')$$

$$\overset{\text{L. 6.3}}{\leqslant} (1+\varepsilon')^2\text{OPT}_{\text{split}}(I, \varepsilon') \leqslant (1+\varepsilon')^2\text{OPT}(I).$$

6.2.4 Generalized Configurations – Step (iv)

In the following steps, we will consider the solution $x_{\text{split},\varepsilon'}$ to the linear program $LP_{pre}(I_{\text{sup},\varepsilon'}, \varepsilon')$ generated in step (iii) of the algorithm. Now the splitting point $\varepsilon' R$ between wide and narrow jobs is clear from the context and we will discard the index ε' in all occurring sets and definitions.

The number of configurations used in x_{split} still depends on the input length, i.e. n, because we did not round the sizes of the narrow jobs. For each used configuration, we have to pay one additional $p_{\max}$ when generating an integral schedule. Therefore, we achieve an additive term depending on n when computing an integral solution at this point. Since we aim to an additive term depending solely on $1/\varepsilon$, we give an additional structure to this solution. For this purpose, we use a set of containers for the narrow jobs, called windows, which where first introduced by Epstein and Levin [30].

A *window* $w = (w_r, w_m)$ is a pair consisting of a resource requirement $r(w) = w_r$ and a number of machines $m(w) = w_m$. As for a configuration, the total time a window is processed is denoted as $p(w)$ and is called its height. At each point of time in a given window w, there can be processed $m(w)$ jobs with summed up resource requirement $r(w)$. For windows w_1, w_2 we write $w_1 \leqslant w_2$ if and only if $r(w_1) \leqslant r(w_2)$ and $m(w_1) \leqslant m(w_2)$.

For a given configuration $C \in \mathcal{C}_I$, we denote by $C|_{\mathcal{J}_W}$ the configuration consisting of all wide jobs in C; and for a given set of configurations $\mathcal{C} \subseteq \mathcal{C}_I$, we define $\mathcal{C}_W := \{C|_{\mathcal{J}_W} | C \in \mathcal{C}\}$. Note that each configuration in $\mathcal{C}_W$ contains at most $1/\varepsilon'$ jobs since each of the wide jobs needs at least $\varepsilon' R$ resource. A *generalized configuration* (C, w) is a pair consisting of a configuration $C \in \mathcal{C}_W$ and a window w. (C, w) is valid for an instance I, if $m(w) \leqslant m - m(C)$ and $r(w) \leqslant R - r(C)$. For a configuration $C \in \mathcal{C}_W$ with $r(C) < R$, we define by $w(C) := (R - r(C), m - m(C))$ the *main window* for C.

Let $\mathcal{W}$ be any set of windows and $\mathcal{C}_W$ any set of configurations consisting just of wide jobs. The following linear program LP_W describes the relation between generalized configurations and jobs assigned to them. This linear program was introduced by Epstein and Levin [30] but with a different set of generalized configurations.

$\mathrm{LP}_W(I, \mathcal{C}_W, \mathcal{W})$:

$$\sum_{C \in \mathcal{C}_W} \sum_{\substack{w \in \mathcal{W} \\ w \leqslant w(C)}} C(j) x_{(C,w)} \geqslant p(j), \qquad \forall j \in \mathcal{J}_W \tag{6.6}$$

$$\sum_{w \in \mathcal{W}} y_{j,w} \geqslant p(j), \qquad \forall j \in \mathcal{J}_N \tag{6.7}$$

$$m(w) \sum_{\substack{C \in \mathcal{C}_W \\ w(C) \geqslant w}} x_{(C,w)} \geqslant \sum_{j \in \mathcal{J}_N} y_{j,w}, \qquad \forall w \in \mathcal{W} \tag{6.8}$$

$$r(w) \sum_{\substack{C \in \mathcal{C}_W \\ w(C) \geqslant w}} x_{(C,w)} \geqslant \sum_{j \in \mathcal{J}_N} r(j) y_{j,w}, \qquad \forall w \in \mathcal{W} \tag{6.9}$$

$$x_{(C,w)} \geqslant 0, \qquad \forall C \in \mathcal{C}_W, \forall w \in \mathcal{W} \tag{6.10}$$

$$y_{j,w} \geqslant 0, \qquad \forall w \in \mathcal{W}, \forall j \in \mathcal{J}_N \tag{6.11}$$

The variable $x_{(C,w)}$ denotes the processing time of the generalized configuration (C, w) and the value $y_{j,w}$ indicates which amount of job j is processed in window w. Inequalities (6.6) and (6.7) ensure that for each job there is enough processing time reserved, while the constraints (6.8) and (6.9) ensure that in each window there is enough space to schedule the contained jobs.

Given a solution (x, y) to LP_W we define

$$p(x) := \sum_{C \in \mathcal{C}_W} \sum_{\substack{w \in \mathcal{W} \\ w \leqslant w(C)}} x_{(C,w)},$$

which is the makespan of (x, y), and

$$p(w, x) := \sum_{\substack{C \in \mathcal{C}_W \\ C(w) \geqslant w}} x_{(C,w)},$$

which is the summed up processing time of a window $w \in \mathcal{W}$ in x.

Denote by $\mathcal{C}_{\text{split}} := \{C \in \mathcal{C}_I | (x_{\text{split}})_C > 0\}$ the set of configurations with a non-zero component in x_{split}, where $(x_{\text{split}})_C$ denotes the processing time of configuration C in x_{split}. Accordingly, $\mathcal{C}_{\text{split},W}$ is the set of configurations of wide jobs with respect to $\mathcal{C}_{\text{split}}$. We define $\mathcal{W}_{\text{split}} := \{w(C) | C \in \mathcal{C}_{\text{split},W}\}$ as the set of main windows for $\mathcal{C}_{\text{split},W}$ and $P_{\text{pre}} := \sum_{C \in \mathcal{C}_{\text{split}}} (x_{\text{split}})_C$ as the

makespan of the preemptive schedule.

Lemma 6.4. *Given a solution x_{split} we can find a solution $(\tilde{x}, \tilde{y})$ to the linear program $\text{LP}_W(I, \mathcal{C}_{\text{split},W}, \mathcal{W}_{\text{split}})$, which fulfills*

$$p(\tilde{x}) = P_{\text{pre}} \tag{6.12}$$

Proof. To generate this solution, we simply look at each configuration $C \in \mathcal{C}_{\text{split},W}$ and sum up the processing time of each configuration $C' \in \mathcal{C}_{\text{split}}$, which is reduced to C, meaning $C'|_{\mathcal{J}_W} = C$. Building a generalized configuration, we combine C with its main window $w(C) \in \mathcal{W}_{\text{split}}$. Hence we define

$$\tilde{x}_{(C,w(C))} := \sum_{\substack{C' \in \mathcal{C}_{\text{split}} \\ C'|_{\mathcal{J}_W} = C}} (x_{\text{split}})_{C'}.$$

Equality 6.12 holds for this choice for $\tilde{x}$ since the processing time of each configuration is added to exactly one generalized configuration. With a similar argument, one can see that inequality 6.6 holds since x_{split} fulfills inequality 6.4. Now, we have to ensure that inequalities 6.7 to 6.9 hold. For this purpose, we look at each configuration $C \in \mathcal{C}_{\text{split}}$ and consider the reduced configuration $C|_{\mathcal{J}_W}$ and its main window $w := w(C|_{\mathcal{J}_W})$. For each job $j \in \mathcal{J}_N$ we add its processing time in C, which is $C(j)(x_{\text{split}})_C$, to the window w. In total we get for each window $w \in W$ and each job $j \in \mathcal{J}_N$

$$\tilde{y}_{j,w} := \sum_{\substack{C \in \mathcal{C}_{\text{split}} \\ w(C|_{\mathcal{J}_W}) = w}} C(j)(x_{\text{split}})_C.$$

Since the configuration C was valid and the constraint 6.4 hold for x_{split} the constraints 6.7 to 6.9 hold for $(\tilde{x}, \tilde{y})$. □

6.2.5 Reducing the number of configurations – Step (v)

At this point, we already have reduced the number of used (generalized) configurations: Each wide job has a width of at least $\varepsilon' R$ and they have at most $1/\varepsilon'^2$ different sizes. Therefore the configuration part of each introduced generalized configuration has one of at most $(\frac{1}{\varepsilon'^2})^{1/\varepsilon'}$ different widths. Since in each introduced generalized configuration the configuration part is paired with the corresponding main window (which is the

same for configuration parts with the same width and number of used machines), we used at most $(\frac{1}{\varepsilon'^2})^{1/\varepsilon'}$ different generalized configurations. This number is independent of the input size. Nevertheless, this value is still too large since it is exponential in $1/\varepsilon'$ and if $|\mathcal{J}_{\text{sup}}| < (\frac{1}{\varepsilon'^2})^{1/\varepsilon'}$ not even a better bound on the number of generated configurations. (The number of generalized configurations in $(\tilde{x}, \tilde{y})$ is bounded by $\min\{|\mathcal{J}_{\text{sup}}| + 1, (\frac{1}{\varepsilon'^2})^{1/\varepsilon'}\}$). To reduce the number of generalized configurations, we round the resource requirements of the windows, i.e. the widths of the windows, in the next step. We define $P_{\text{pre}}(C) := \tilde{x}_{C,w(C)}$ for each $C \in \mathcal{C}_W$ and $P_{\text{pre}}(K) := \sum_{C \in K} P_{\text{pre}}(C)$ for each $K \subseteq \mathcal{C}_W$.

Lemma 6.5. *Given a solution $(\tilde{x}, \tilde{y})$ to $\text{LP}_W(I, \mathcal{C}_W, \mathcal{W}_{\text{split}})$, we can find a set $\mathcal{W}' \subseteq \mathcal{W}_{\text{split}}$ with $|\mathcal{W}'| \leqslant \varepsilon'^{-2} + 2$ and a solution $(\bar{x}, \bar{y})$ to $\text{LP}_W(I, \mathcal{C}_W, \mathcal{W}')$, which fulfils*

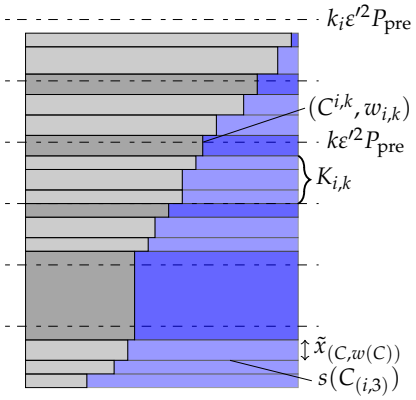
$$p(\bar{x}) \leqslant (1 + \varepsilon') P_{\text{pre}} \tag{6.13}$$

and contains at most $|\mathcal{J}_N| + |\mathcal{J}_W| + 2|\mathcal{W}'| + 1$ non zero components in at most $\mathcal{O}((|\mathcal{J}| + |\mathcal{W}'|)^{1.5356}|\mathcal{J}||\mathcal{W}'|)$ operations.

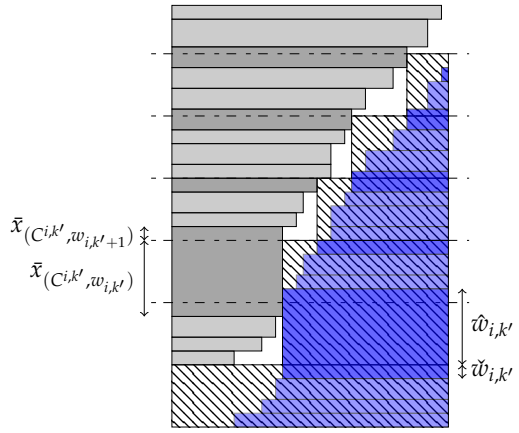
Proof. The steps to find $\mathcal{W}'$ and $(\bar{x}, \bar{y})$ are described in the following. The sets, configurations and sizes defined for these steps can also be found in the Figures 6.2a and 6.2b.

To find the set $\mathcal{W}'$, we reduce the number of window resource requirements by a further grouping step. We partition the set of generalized configurations by the number of machines in the window. For each $i \in \{1, \ldots, m\}$ we define $K_i := \{C \in \mathcal{C}_W | m(C) = i\}$ to denote the set containing all configurations using exactly i machines. Since at most $1/\varepsilon'$ wide jobs can be part of one configuration, only the sets K_1 to $K_{1/\varepsilon'}$ are not empty. For each of these sets we apply linear grouping: We number the configurations in K_i such that $r(C_{i,1}) \leqslant r(C_{i,2}) \leqslant r(C_{i,3}) \ldots$ holds. Then we stack the configurations defining start positions $s(C_{i,1}) := 0$ and $s(C_{i,j}) := s(C_{i,j-1}) + P_{\text{pre}}(C_{i,j-1})$ for each $j > 1$. Furthermore, we define a configurations end position $e(C_{i,j}) := s(C_{i,j+1})$. The summed up height of all stacks is P_{pre}. We want to get about ε'^{-2} windows. Hence, we split our stacks in ε'^{-2} pieces total and consider in each stack the multiples of $\varepsilon'^2 P_{\text{pre}}$ to group the windows.

For each $0 < i \leqslant 1/\varepsilon'$, we define $k_i := \lceil P_{\text{pre}}(K_i)/(\varepsilon'^2 P_{\text{pre}}) \rceil$ which is the



(a) A stack of the generalized configurations in K_i. The grey rectangles represent the configurations and the blue rectangles represent the windows. The dashed lines are multiplies of $\varepsilon'^2 P_{\text{pre}}$.



(b) The same stack of the generalized configurations as in figure 6.2a. But this time the windows are shifted $\varepsilon'^2 P_{\text{pre}}$ downwards. One can see that their area fits into the new windows (hatched rectangles).

Figure 6.2. Shifting the windows

first multiple of $\varepsilon'^2 P_{\text{pre}}$ which does not intersect a generalized configuration in the ith stack since it has a height of at most $P_{\text{pre}}(K_i)$. (k_i might touch the last configuration if $\lceil P_{\text{pre}}(K_i)/(\varepsilon'^2 P_{\text{pre}})\rceil = P_{\text{pre}}(K_i)/(\varepsilon'^2 P_{\text{pre}})$, but we allow this.) For each $k \in \{1, \dots, k_i - 1\}$ there is a configuration C which intersects with $k\varepsilon'^2 P_{\text{pre}}$, i.e., $s(C) < k\varepsilon'^2 P_{\text{pre}} \leqslant s(C) + P_{\text{pre}}(C)$. We denote this configuration by $C^{i,k}$ and define $w_{i,k} := w(C^{i,k})$ and $w_{i,k_i} := (0,0)$. Further, we define the set $K_{i,k}$ as the set of configurations lying between the configurations $C^{i,k-1}$ and $C^{i,k}$. The set $\mathcal{W}'$ is defined such that it contains all the windows intersected by a multiple of $\varepsilon'^2 P_{\text{pre}}$ and two default windows. More precisely, we define

$$\mathcal{W}_{K_i} := \{w_{i,k} | k \in \{1, \dots, k_i - 1\}\}.$$

the set of chosen windows from K_i and define

$$\mathcal{W}' := \bigcup_{i=1}^{1/\varepsilon'} \mathcal{W}_{K_i} \cup \{(0,0), (R,m)\}$$

It holds that $|\mathcal{W}_{K_i}| \leqslant \lfloor P_{\text{pre}}(K_i)/(\varepsilon'^2 P_{\text{pre}}) \rfloor$ and therefore

$$|\mathcal{W}'| \leqslant 2 + \sum_{i=1}^{1/\varepsilon'} \left\lfloor \frac{P_{\text{pre}}(K_i)}{\varepsilon'^2 P_{\text{pre}}} \right\rfloor \leqslant 2 + \frac{1}{\varepsilon'^2}.$$

After choosing the set of windows $\mathcal{W}'$ the next step is to generate a solution to the LP_W, which uses just windows in $\mathcal{W}'$ and needs not much further processing time. The crucial step is to shift the windows in each stack exactly $\varepsilon'^2 P_{\text{pre}}$ downwards.

Let us consider the configurations $C^{i,k}$ and $C^{i,k-1}$. The resource requirement of $C^{i,k-1}$ is less or equal to the resource requirement of $C^{i,k}$. So for each configuration $C \in K_{i,k}$ we have $w(C) \geqslant w_{i,k}$. If we shift the windows $\varepsilon'^2 P_{\text{pre}}$ downwards, the window $w_{i,k}$ is now processed alongside $C^{i,k-1}$. The windows above $w_{i,k}$ have a lesser resource requirement. We now round up the resource requirement of the windows alongside the configurations in $C \in K_{i,k}$ such that they have the same resource requirement as $w_{i,k}$. Therefore, for each $1 \leqslant 1/\varepsilon'$, for all $k \leqslant k_i$, and for each $C \in K_{i,k}$, we define $\bar{x}_{(C,w_{i,k})} := \tilde{x}_{(C,w(C))}$.

With configuration $C^{i,k}$ we have to be more careful. The lower part of this configuration should be alongside window $w_{i,k}$, the other alongside $w_{i,k+1}$. We have to find the biggest multiple of $\varepsilon'^2 P_{\text{pre}}$ the configuration $C^{i,k}$ was intersected by. In case $p(C^{i,k}) > \varepsilon'^2 P_{\text{pre}}$ the configuration $C^{i,k}$ will be intersected by more than one multiple of $\varepsilon'^2 P_{\text{pre}}$. The biggest multiple is defined by $\varepsilon'^2 P_{\text{pre}} \lfloor \frac{e(C^{i,k})}{\varepsilon'^2 P_{\text{pre}}} \rfloor$. This multiple defines the height at which the configuration is split. We define for each $i \leqslant 1/\varepsilon'$ and for each $k \leqslant k_i$:

$$\bar{x}_{(C^{i,k},w_{i,k})} := \varepsilon'^2 P_{\text{pre}} \left\lfloor \frac{e(C^{i,k})}{\varepsilon'^2 P_{\text{pre}}} \right\rfloor - s(C^{i,k})$$

$$\bar{x}_{(C^{i,k},w_{i,k+1})} := \tilde{x}_{(C^{i,k},w_{i,k})} - \bar{x}_{(C^{i,k},w_{i,k})}$$

Finally, we need some extra space for the windows, which were shifted below the lowest configuration. In each stack, it concerns window parts of total height $\varepsilon'^2 P_{\text{pre}}$. Since we have $1/\varepsilon'$ stacks, we need $\varepsilon' P_{\text{pre}}$ extra space. We round these windows up to the window (R, m). Hence, the configuration $(\varnothing, (R, m))$ is the configuration which is extended in height.

We define $\bar{x}_{(\varnothing,(R,m))} := \tilde{x}_{(\varnothing,(R.m))} + \varepsilon' P_{\text{pre}}$.

In the following, we describe how to assign the narrow jobs to the round up windows. Let us consider a configuration $C \in K_{i,k+1}$. The window $w(C)$ was shifted down such that it can be rounded up to $w_{i,k}$. To round this window up, we have to know which amount of it was processed alongside C. This amount is given by $\varphi_C := P_{\text{pre}}(C)/P_{\text{pre}}(w(C))$. So for each configuration $C \in K_{i,k+1}$ and each job $j \in \mathcal{J}_N$ we add $\varphi_C \tilde{y}_{j,w(C)}$ processing time to $\bar{y}_{j,w_{i,k}}$.

Next, we consider a window $w_{i,k}$. In general, this has to be split: as one can see in figure 6.2b the upper part of window $w_{i,k}$ stays in this window while the lower part is put into window $w_{i,k-1}$. This time we have to split the window at the smallest multiple of $\varepsilon'^2 P_{\text{pre}}$. This multiple is defined by $\varepsilon'^2 P_{\text{pre}} \lceil s(C^{i,k})/\varepsilon'^2 P_{\text{pre}} \rceil$. Again we have to know which amount of window $w_{i,k}$ has to be processed where. Let $\check{w}_{i,k} = \varepsilon'^2 P_{\text{pre}} \lceil s(C^{i,k})/\varepsilon'^2 P_{\text{pre}} \rceil - s(C^{i,k})$ be the processing time of window $w_{i,k}$ which has to be scheduled in the window $w_{i,k-1}$ and $\hat{w}_{i,k} = \tilde{x}_{(C^{i,k},w_{i,k})} - \check{w}_{i,k}$ the processing time of window $w_{i,k}$, which can stay in $w_{i,k}$. Furthermore, we have to know which fraction of the total processing time of $w_{i,k}$ is presented by these two values. We define $\hat{\varphi}_{i,k} := \hat{w}_{i,k}/P_{\text{pre}}(w_{i,k})$ and $\check{\varphi}_{i,k} := \check{w}_{i,k}/P_{\text{pre}}(w_{i,k})$. We now know that we have to add $\hat{\varphi}_{i,k}\tilde{y}_{j,w_{i,k}}$ to $\bar{y}_{j,w_{i,k}}$ and $\check{\varphi}_{i,k}\tilde{y}_{j,w_{i,k}}$ to $\bar{y}_{j,w_{i,k-1}}$. In total we have

$$\bar{y}_{j,w_{i,k}} := \sum_{C \in K_{i,(k+1)}} \varphi_C \tilde{y}_{j,w(C)} + \hat{\varphi}_{i,k}\tilde{y}_{j,w_{i,k}} + \check{\varphi}_{i,k+1}\tilde{y}_{j,w_{i,k+1}}$$

for each $i \leqslant 1/\varepsilon'$, $k \leqslant k_i$ and $j \in \mathcal{J}_N$.

We consider the window (R,m) separately. First, we have to round the main windows from all configurations in $K_{i,1}$ up to (R,m). Meaning for each $i \leqslant 1/\varepsilon'$, $C \in K_{i,1}$ and $j \in \mathcal{J}_N$ we add $\varphi_C \tilde{y}_{j,w(C)}$ to $\bar{y}_{j,(R,m)}$. Furthermore, we have to round up the lower part of window $w_{i,1}$ to (R,m). Additional jobs which where processed in window (R,m) before stay there. So in total we have

$$\bar{y}_{j,(R,m)} := \tilde{y}_{j,(R,m)} + \sum_{i=1}^{1/\varepsilon'} \left(\check{\varphi}_{i,1}\tilde{y}_{j,w_{i,1}} + \sum_{C \in K_{i,1}} \varphi_C \tilde{y}_{j,w(C)}\right)$$

for all $j \in \mathcal{J}_N$. Therefore, $(\bar{x},\bar{y})$ is a solution to $\text{LP}_W(\mathcal{W}')$. Using suit-

able data structures $(\tilde{x}, \tilde{y})$ as well as $(\bar{x}, \bar{y})$ can be computed in $\mathcal{O}((m + \log(n)/\varepsilon)n)$ operations. This linear program has $|\mathcal{J}_N| + |\mathcal{J}_{\sup W}| + 2|\mathcal{W}'| \in \mathcal{O}(|\mathcal{J}| + |\mathcal{W}'|)$ constraints and at most $|\mathcal{C}_W||\mathcal{W}'| + |\mathcal{J}_N||\mathcal{W}'| \in \mathcal{O}(|\mathcal{J}||\mathcal{W}'|)$ variables. So we can compute a solution with at most $|\mathcal{J}_N| + |\mathcal{J}_{\sup W}| + 2|\mathcal{W}'| + 1$ non zero components in $\mathcal{O}((|\mathcal{J}| + |\mathcal{W}'|)^{1.5356}|\mathcal{J}||\mathcal{W}'|)$ operations. □

In the step (v) the algorithm uses the results from Lemma 6.5 to find a solution $(\bar{x}, \bar{y})$ to LP_W which uses at most $|\mathcal{J}_N| + |\mathcal{J}_W| + 2|\mathcal{W}'| + 1 \leqslant |\mathcal{J}_N| + 3/\varepsilon'^2 + 5$ non zero components in at most

$$\mathcal{O}((|\mathcal{J}_{\sup}| + |\mathcal{W}'|)^{1.5356}|\mathcal{J}_{\sup}||\mathcal{W}'|)$$

operations. Since $|\mathcal{W}'| \in \mathcal{O}(1/\varepsilon^2)$ the total running time of the algorithms from step (i) to (v) can be bounded by

$$\mathcal{O}(|\mathcal{J}_{\sup}|(\ln(|\mathcal{J}_{\sup}|) + \varepsilon^{-2})(|\mathcal{J}_{\sup}|^{1.5356} + m/\varepsilon^4)).$$

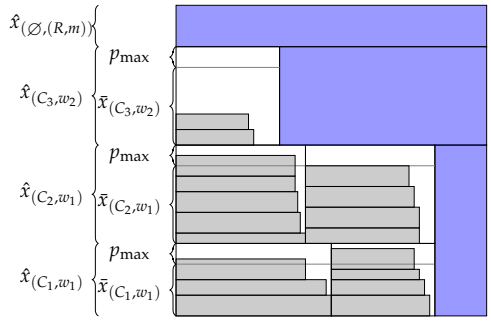
6.2.6 Integral Solution – Step (vi)

We now generate an integral schedule of the jobs in $\mathcal{J}$. The used technique to schedule the wide jobs is similar to the technique presented by Kenyon and Rémila [77] to place the wide rectangles into their fractional packing of rectangles. To place the narrow jobs, we use a similar argument as Epstein and Levin in [30].

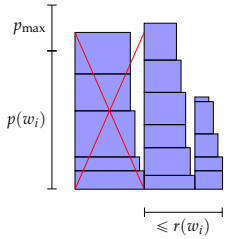
We say a narrow job in $\mathcal{J}_N$ is scheduled fractionally, if it is assigned to more than one window, hence each fractionally scheduled job corresponds to at least two non zero components. Note that $(\bar{x}, \bar{y})$ has at most $|\mathcal{J}_N| + 3/\varepsilon'^2 + 5$ non zero components. Since each job in $\mathcal{J}_N$ needs one none zero component to be scheduled there are at most $3/\varepsilon'^2 + 5$ non zero components left for configurations or narrow jobs. Hence, $(\bar{x}, \bar{y})$ contains at most $3/\varepsilon'^2 + 5$ fractionally scheduled jobs and configurations in total.

Lemma 6.6. *Let $(\bar{x}, \bar{y})$ be a basic solution to $\mathrm{LP}_W(I_{\sup}, \mathcal{C}_W, \mathcal{W}')$ such that the total number of fractional scheduled narrow jobs and used configurations is bounded by K. There is a solution which places the jobs in $\mathcal{J}(I)$ integrally and has a makespan of at most*

$$(1 + \varepsilon')p(\bar{x}) + (1 + \varepsilon'^{-1} + K)p_{\max}.$$



(a) Three generalized configurations and the configuration $\hat{x}_{(\emptyset,(R,m))}$. Wide jobs are filled in the generalized configurations. The windows still need to be filled.



(b) Stacks of small jobs assigned to a window w_i. The first stack is discarded and positioned at the end of the schedule.

Figure 6.3. Integral placement of the jobs into the generalized configurations

This solution can be found in $\mathcal{O}(K/\varepsilon + |\mathcal{J}(I)|\log(|\mathcal{J}(I)|) + |\mathcal{W}|)$ *operations.*

Proof. First, we modify the solution $(\bar{x}, \bar{y})$ to have enough space to schedule the jobs integrally. We define $\hat{x}_{(C,w)} := \bar{x}_{(C,w)} + p_{\max}$ if $\bar{x}_{(C,w)} > 0$, and $\hat{x}_{(C,w)} = 0$ otherwise, which needs $\mathcal{O}(K)$ operations. We denote by $\mathcal{J}_{frac} \subseteq \mathcal{J}_N$ the set of fractional scheduled narrow jobs in $(\bar{x}, \bar{y})$. For each $j \in \mathcal{J}_{frac}$ and each window $w \in \mathcal{W}$ we set $\hat{y}_{j,w} := 0$ and $\hat{y}_{j,(R,m)} := p(j)$. Further we add $p(\mathcal{J}_{frac}) \leqslant p_{\max}|\mathcal{J}_{frac}|$ to $\hat{x}_{(\emptyset,(R,m))}$. For every other $j \in \mathcal{J}_N \backslash \mathcal{J}_{frac}$ we set $\hat{y}_{j,w} = \bar{y}_{j,w}$. This step needs at most $\mathcal{O}(K + \mathcal{J}(I_{\text{sup}}))$ operations. In this transformation, we add at most $p_{\max}$ processing time for each fractional job and for each configuration . Hence, we have

$$p(\hat{x}) \leqslant p(\bar{x}) + Kp_{\max}.$$

Next, we place the wide jobs. We rearrange the generalized configurations such that configurations with the same window are scheduled consecutively and number them in ascending order. Since there are at most $\mathcal{O}(K)$ configurations, this step needs at most $\mathcal{O}(|\mathcal{W}'| + K)$ operations, given there is a appropriate data structure to manage the windows. We iterate over each $i < G$, i.e., over all groups, as well as all generalized configurations and fill the spaces for the job $i \in \mathcal{J}_{W,\text{sup}}$, which has a resource requirement of R_i, with jobs from $\mathcal{J}_{W,i} \cap \mathcal{J}_W$, until the height of the configuration, $\bar{x}_{(C,w)}$, is reached. Each last job in a configuration is

allowed to overlap the border $\bar{x}_{(C,w)}$ since we have added $p_{\max}$ extra space. All jobs from $\mathcal{J}_{W,i} \cap \mathcal{J}_W$ can be placed in the configurations. In addition, there is one generalized configuration (C, w) where the border $\bar{x}_{(C,w)}$ is not overlapped by a job from $\mathcal{J}_{W,i}$ since $\sum_{(C,w)\in\tilde{\mathcal{C}}} C(i)\bar{x}_{(C,w)} \geqslant p(\mathcal{J}_{W,i})$. In this particular configuration we place the job which was intersected by the multiple of $\varepsilon'^2 P_W$ in the first linear grouping and which fractions were put in $\mathcal{J}_{W,i}$ as well as in $\mathcal{J}_{W,i+1}$. Since one of its fractions was in $\mathcal{J}_{W,i}$ it fits into the reserved space. This greedy filling of the configurations can be done in $\mathcal{O}(K/\varepsilon + \mathcal{J}(I))$ since we need to consider each of the at most $1/\varepsilon$ entries in each configuration and each job only once.

To place the small jobs, we consider each window $w \in \mathcal{W}$. Let $\mathcal{J}(w)$ be the set of jobs contained in w. We order the jobs in $\mathcal{J}(w)$ by decreasing resource requirement. Since the generalized configurations containing window w are scheduled consecutively, we can build stacks of jobs form $\mathcal{J}(w)$ with total processing time $p(w, \bar{x})$. During this process, we cut the last job of the stack at the horizontal line $p(w, \bar{x})$ if it overlaps this line and schedule place the residual part at the bottom of the next step. We need at most $\mathcal{O}(|\mathcal{J}(I_{\text{sup}})| \log(|\mathcal{J}(I_{\text{sup}})|))$ operations to build these stacks.

Since $m(w)p(w, \bar{x}) \geqslant \sum_{j\in\mathcal{J}_N} \hat{y}_{j,w}$, we have to build at most $m(w)$ of these stacks. Because we have $m(w)$ free machines in each window there is a free machine for each stack. Consider the jobs $\mathcal{J}_{\text{top}}$ which are positioned at the top of their stack. These are the jobs with the lowest resource requirement of their stack. Hence, at each point in time the small jobs in the stacks have a total resource requirement of at least $r(\mathcal{J}_{\text{top}})$ and therefore $p(w, \bar{x})r(\mathcal{J}_{\text{top}}) \leqslant \sum_{j\in J(w)} r(j)p(j) \leqslant p(w, \bar{x})r(w)$. Therefore, the jobs in $\mathcal{J}_{\text{top}}$ fit into the window w.

We remove the stack with the largest resource requirement together with the second half of the potentially overlapping job. Then the total resource requirement of the jobs at the bottom of the stacks is less than $r(\mathcal{J}_{\text{top}})$ since each of these jobs was added right after a job from $\mathcal{J}_{\text{top}}$. Therefore, at each point in time in window w it holds that the resource requirement of small jobs is at most $r(\mathcal{J}_{\text{top}})$. We schedule the residual stacks such that the job, which was intersected at the end of the stack, is scheduled integral and overlaps the window at the top by at most $p_{\max}$. This overlapping adds no further height to the makespan since we extended the configuration by $p_{\max}$.

The removed stack has a total processing time of at most $p(w, \bar{x}) + p_{\max}$. Since this stack is removed in each window, jobs with summed up processing time at most $\sum_{w \in \mathcal{W} \setminus \{(0,0),(R,m)\}} (p(w, \bar{x}) + p_{\max}) \leqslant p(\bar{x}) + \varepsilon'^{-2} p_{\max}$ have to be placed at the end of the schedule. Since each of these jobs has a resource requirement of at most $\varepsilon' R$, we can schedule $1/\varepsilon'$ of them at the same time. Hence, we need an additional processing time of $\varepsilon' p(\bar{x}) + (1 + \varepsilon'^{-1}) p_{\max}$. In total we get a schedule with makespan at most

$$\begin{aligned} &\varepsilon' p(\bar{x}) + (1 + \varepsilon'^{-1}) p_{\max} + p(\bar{x}) + K p_{\max} \\ &= (1 + \varepsilon') p(\bar{x}) + (1 + \varepsilon'^{-1} + K) p_{\max}. \end{aligned}$$

Furthermore, this schedule was produced in at most

$$\mathcal{O}(K/\varepsilon + |\mathcal{J}(I)| \log(|\mathcal{J}(I)|) + |\mathcal{W}|)$$

operations. □

The solution we get from Lemma 6.5 has at most $|\mathcal{J}_N| + |\mathcal{J}_{\sup W}| + 2|\mathcal{W}'| + 1 \leqslant |\mathcal{J}_N| + 3/\varepsilon'^2 + 5$ non zero components. Since each job $j \in \mathcal{J}_N$ needs a non zero component to be scheduled we have at most $3/\varepsilon'^2 + 5$ configurations and fractional jobs. Therefore, the generated integral schedule has a makespan of

$$\begin{aligned} &(1 + \varepsilon') p(\bar{x}) + (1 + \varepsilon'^{-1} + 3/\varepsilon'^2 + 5) p_{\max} \\ &\leqslant (1 + \varepsilon')^3 \mathrm{OPT}_{\mathrm{split}}(I_{\sup}) + \mathcal{O}(1/\varepsilon^2) p_{\max} \\ &\leqslant (1 + \varepsilon')^4 \mathrm{OPT}_{\mathrm{split}}(I) + \mathcal{O}(1/\varepsilon^2) p_{\max} \\ &\leqslant (1 + \varepsilon) \mathrm{OPT}(I) + \mathcal{O}(1/\varepsilon^2) p_{\max}. \end{aligned}$$

The number of operations to build the integral schedule is dominated by the number of operations to build the first preemptive schedule, which is $\mathcal{O}(n(\ln(n) + \varepsilon^{-2})(n^{1.5356} + m\varepsilon^{-4}))$ and, hence, it represents the total running time of this AFPTAS. Note that we stack jobs in contiguous containers and hence the resources of the jobs can be allocated contiguously. Therefore, this schedule is feasible for instances, where the contiguous allocation of the resource is required, i.e., the algorithm generates a feasible solution for instances of Strip Packing With Cardinality Constraint as well.

6.2.7 The Case $m \leqslant 1/\varepsilon$

Let $\varepsilon > 0$ and an instance $I = (\mathcal{J}, m, R)$, with $1/\varepsilon \geqslant m$ be given. In this case we do not partition the set of jobs into wide and narrow jobs. We apply the linear grouping to all jobs. Again we get $G \leqslant 1/\varepsilon'^2$ groups $\mathcal{J}_i$. We denote by $\mathcal{J}_{\text{sup}}$ the set of jobs, which contains for each $i < G$ one job with processing time $p(\mathcal{J}_i)$ and resource requirement $\max\{r(j)|j \in \mathcal{J}_i\}$. We denote by $\mathcal{C}_{\text{sup}}$ the set of valid configurations of jobs in $\mathcal{J}_{\text{sup}}$. We denote by $I_{\text{sup}} := (\mathcal{J}_{\text{sup}}, m, R)$ the round up instance. To get a preemptive schedule for I_{sup} we have to solve $\text{LP}_{\text{split}}(I_{\text{sup}})$, while interpreting each job in $\mathcal{J}_{\text{sup}}$ as a wide job. By Lemma 6.2 we get a solution x_{split} with $\mathcal{O}(|\mathcal{J}_{\text{sup}}|(\ln(|\mathcal{J}_{\text{sup}}|) + \varepsilon'^{-2})) = \mathcal{O}(\varepsilon'^{-4})$ non-zero components and $\sum_{C \in \mathcal{C}_{\text{sup}}} (x_{\text{split}})_C \leqslant (1+\varepsilon')\text{OPT}_{\text{split}}$ in $\mathcal{O}(|\mathcal{J}_{\text{sup}}|(\ln(|\mathcal{J}_{\text{sup}}|) + 1/\varepsilon'^2)(|\mathcal{J}_W|z + |\mathcal{J}_N| + mz'^2/\varepsilon'^2)) = \mathcal{O}(m^3/\varepsilon'^6) = \mathcal{O}(1/\varepsilon'^9)$ operations . Since we have now a polynomial number of variables, we can construct a solution $\tilde{x}$, which has at most $1/\varepsilon'^2$ non zero components and for which it holds that $\sum_{C \in \mathcal{C}_{\text{sup}}} (x_{\text{split}})_C = \sum_{C \in \mathcal{C}_{\text{sup}}} \tilde{x}_C$, in $\mathcal{O}(|\mathcal{J}_{\text{sup}}|^{2.5356}(\ln(|\mathcal{J}_{\text{sup}}|) + \varepsilon'^{-2})) = \mathcal{O}(1/\varepsilon'^{4.5356})$ operations.

To each of these components we add $p_{\max}$ processing time. Now we place the jobs from $\mathcal{J}_i$ for each $i < G$ in the configurations as we placed the wide jobs in the first case. We get a schedule of length at most $(1+\varepsilon')\text{OPT}_{\text{split}} + \varepsilon'^{-2}p_{\max}$. The jobs in $\mathcal{J}_G$ are added in the end of the schedule. These jobs have a total makespan of at most $\varepsilon'\text{OPT}_{\text{split}}(I)$ because $p(\mathcal{J}_G) \leqslant \varepsilon'^2 p(\mathcal{J})$ and $\text{OPT}_{\text{split}}(I) \geqslant p(\mathcal{J})/m \geqslant \varepsilon' p(\mathcal{J})$. In the end, we have a schedule with a total makespan of at most

$$\begin{aligned}(1+\varepsilon')\text{OPT}_{\text{split}}(I) + \varepsilon'\text{OPT}_{\text{split}}(I) + 1/\varepsilon'^2 p_{\max} \\ \leqslant (1+\varepsilon)\text{OPT}(I) + \mathcal{O}(1/\varepsilon^2)p_{\max}.\end{aligned}$$

6.3 The improved AFPTAS

It is possible to reduce the additive term using different rounding strategies. However, we have to differentiate other cases than before: $m > 1/\varepsilon'^2$ and $m \leqslant 1/\varepsilon'^2$. Let $m > 1/\varepsilon'^2$. In the described algorithm we have two rounding steps. One for the wide jobs and one for the windows for narrow jobs.

To achieve a better additive term we have to modify both of them. Note that the general procedure remains the same as described in the algorithm overview. However, we modify the simplification and rounding steps, as well as the first solving of the linear program. In the following we only repeat the steps, where we change the algorithm.

6.3.1 Rounded Instance – Step (ii)

The first modification is a better rounding of the resource requirements of the jobs. Previously, we used linear grouping to round the wide jobs. By this rounding we received ε'^{-2} different job sizes. To improve the additive term, we have to reduce this number since the number of configurations we get by solving LP_W depends linearly on it. Karmakar and Karp [74] have introduced a second rounding strategy called geometric grouping. This technique was refined for strip packing in [15] and [101]. We describe how to apply the geometric grouping to the wide jobs:

Lemma 6.7. *For every $\varepsilon > 0$ with $1/\varepsilon \in \mathbb{N}$ and every instance I with set of wide jobs $\mathcal{J}_{W,\varepsilon}$, it is possible to reduce the number of different resource requirement of all jobs to at most $\mathcal{O}(\log(m)/\varepsilon)$ sizes in at most $\mathcal{O}(n\log(1/\varepsilon))$ operations such that each job is mapped to exactly one rounded size. Furthermore, the number of resource requirement of wide jobs is reduced to $\mathcal{O}(\log(1/\varepsilon)/\varepsilon)$. This reduction extends the optimal makespan by at most $2\varepsilon \cdot \text{area}(\mathcal{J}(I))/R + \varepsilon p(\mathcal{J}(I))/m \leqslant 3\varepsilon \text{OPT}_{\text{split}}(I,\varepsilon)$ when scheduling the jobs as splittable ones. The resulting rounded instance is called $I'_{\text{sup},\varepsilon'}$.*

Proof. First, we partition the set of jobs $\mathcal{J}$ into $\lceil \log(m) \rceil + 1$ sets. We construct the first $\lceil \log(m) \rceil$ sets such that the ith set $\mathcal{J}_i$ contains the jobs with resource requirements in the interval $(R/2^i, R/2^{i-1}]$. The last of these sets contains jobs with resource requirements in the interval $(R/2^{\lceil \log(m) \rceil}, R/2^{\lceil \log(m) \rceil - 1}]$. The residual jobs build the last set $\mathcal{J}_\perp$. All the jobs in this set have a resource requirement of at most $R/2^{\lceil \log(m) \rceil} \leqslant R/m$, and hence, we can schedule m of them at the same time without violating any constraint.

To round the jobs, we consider all the sets $\mathcal{J}_i$, $i \in \{1, \ldots, \lceil \log(m) \rceil, \perp\}$. The jobs in a set $\mathcal{J}_i$ are sorted in increasing order of their resource requirement and stacked such that the job with the smallest resource requirement

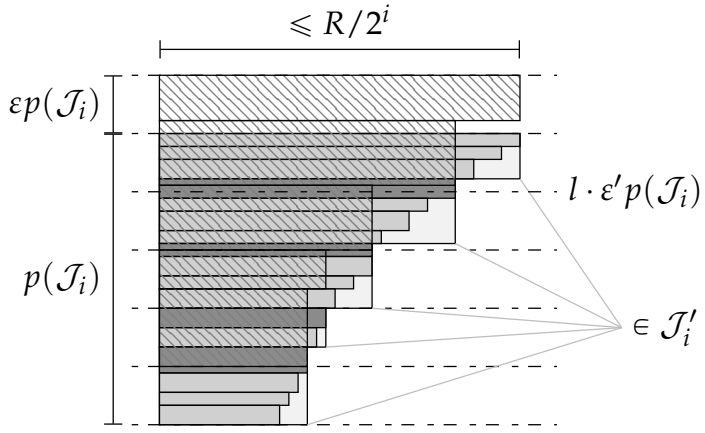


Figure 6.4. Rounding of the jobs in $\mathcal{J}_i$. The hatched rectangles represent the rounded jobs that are shifted upwards by $\varepsilon' p(\mathcal{J}_i)$.

is on the bottom, see Figure 6.4. The height of the stack is given by $p(\mathcal{J}_i) := \sum_{j \in \mathcal{J}_i} p(j)$. We will partition this stack into $1/\varepsilon$ segments. We draw a horizontal line at the height $\varepsilon p(\mathcal{J}_i)$, starting at 0. If there is no job intersected by this line, we leave the horizontal line at its position. Otherwise, we shift this line up such that it lies at the end of the cut job. We define $\mathcal{J}_{i,\text{split}}$ as the set of jobs that where previously intersected by these lines. We call the set of jobs which is contained between the lth and $(l+1)$st horizontal line group l and denote it with $\mathcal{J}_{i,l}$. To define the set $\mathcal{J}'$, we introduce for each group one job with processing time $p(\mathcal{J}_{i,l})$ and resource requirement $\max_{j \in \mathcal{J}_{i,l}} r(j)$. Hence $|\mathcal{J}'| \leqslant (1/\varepsilon)(\log(m)+2) \in \mathcal{O}(\log(m)/\varepsilon)$.

Claim 6.8. Given an optimal schedule for the jobs $\mathcal{J}$, we can find a schedule of the jobs in $\mathcal{J}'$ that uses only the area of the jobs in $\mathcal{J}$ and an extra height of at most $2\varepsilon \cdot \text{area}(\mathcal{J}(I))/R + \varepsilon p(\mathcal{J}(I))/m \leqslant 3\varepsilon \text{OPT}_{\text{split}}(I, \varepsilon)$.

Proof. Consider again the stack of the jobs in $\mathcal{J}_i$ and the stack of rounded jobs for the jobs in $\mathcal{J}_i$. Consider a horizontal line at $\tau \geqslant \varepsilon p(\mathcal{J}_i)$ through the first stack and the corresponding horizontal line at $\tau - \varepsilon p(\mathcal{J}_i)$ trough the schedule. We claim that the resource requirement of the job j_τ intersected by τ is at least as large as the resource requirement of the rounded job j'_τ at $\tau - \varepsilon p(\mathcal{J}_i)$. Let $j_\tau \in \mathcal{J}_{i,l}$. Note that j'_τ is either the rounded job constructed for $\mathcal{J}_{i,l}$ or it is the rounded job constructed for a set $\mathcal{J}_{i',l}$ with $i' < i$. In the second case the claim follows trivially since, by construction, all the jobs

in $\mathcal{J}_{i,l}$ have a resource requirement that is at least as large as the resource requirement of the job constructed for $\mathcal{J}_{i',l}$. On the other hand, if j'_τ the rounded job constructed for $\mathcal{J}_{i,l}$, the job j_τ has to be a job in $\mathcal{J}_{i,\text{split}}$. Since the jobs in $\mathcal{J}_{i,\text{split}}$ are the ones with the larges resource requirement of their group the resource requirement of j_τ has to be at least as large as j'_τ. Hence, when shifting the second stack such that it starts at $\varepsilon p(\mathcal{J}_i)$, we schedule the jobs in $\mathcal{J}'_i$ instead of the jobs that are positioned at the same height in the stack. What remains to be scheduled is a last part of height $\varepsilon p(\mathcal{J}_i)$ that cannot be scheduled instead of any job due to the shifting. These jobs will be scheduled at the end of the schedule in the next step. We call these parts to be scheduled on the top of the schedule segment S_i for the set $\mathcal{J}_i$.

First consider segment $S_\perp$. Note that $p(\mathcal{J}_{S,\perp}) \leqslant m\text{OPT}_{\text{split}}(I,\varepsilon)$ since there can be scheduled at most m jobs at the same time. Hence, the job parts in $S_\perp$ have a total processing time of $\varepsilon p(\mathcal{J}_{S,\perp}) \leqslant \varepsilon m\text{OPT}_{\text{split}}(I,\varepsilon)$. We partition this segment into subsegments of height $\varepsilon p(\mathcal{J}_{S,\perp})/m$ and schedule the job parts intersecting the segments as splittable jobs on one machine each. This does not violate the resource condition since $r(j) \leqslant R/m$ for each $j \in \mathcal{J}_{S,\perp}$. Hence we can schedule the jobs intersecting this segment with makespan at most $\varepsilon p(\mathcal{J}_{S,\perp})/m \leqslant \varepsilon p(\mathcal{J}(I))/m \leqslant \varepsilon\text{OPT}_{\text{split}}(I,\varepsilon)$ as splittable jobs.

Next we consider the residual segments S_i. The jobs in $\mathcal{J}_i$ have a resource requirement of at most $R/2^{i-1}$. Hence, we can schedule jobs in segment S_i as splittable jobs 2^{i-1} times at the same time without violating the resource constraint. Therefore, we need a processing time of $\varepsilon p(\mathcal{J}_{S,i})/2^{i-1}$ to schedule the jobs in this segment. On the other hand, the jobs in the set $\mathcal{J}_i$ have a resource requirement of at least $R/2^i$. Therefore, it holds that $\sum_{i=1}^{\lceil\log(m)\rceil+1} p(\mathcal{J}_{S,i})/2^i \leqslant \text{area}(\mathcal{J}(I))/R \leqslant \text{OPT}_{\text{split}}(I,\varepsilon)$.

Hence the total processing time we add to schedule the jobs in the segments S_i, $i \in \{1,\ldots,\lceil\log(m)\rceil,\perp\}$, is bounded by

$$\begin{aligned}&\sum_{i=1}^{\lceil\log(m)\rceil+1} \varepsilon p(\mathcal{J}_{S,i})/2^{i-1} + \varepsilon p(\mathcal{J}_{S,\perp})/m\\ &= 2\varepsilon \sum_{i=1}^{\lceil\log(m)\rceil+1} p(\mathcal{J}_{S,i})/2^i + \varepsilon p(\mathcal{J}_{S,\perp})/m\end{aligned}$$

$$\begin{aligned}&\leqslant 2\varepsilon \cdot \text{area}(\mathcal{J}(I))/R + \varepsilon p(\mathcal{J}(I))/m\\&\leqslant 3\varepsilon \text{OPT}_{\text{split}}(I,\varepsilon),\end{aligned}$$

which concludes the proof of the claim. ◁

The running time of this rounding procedure can be bounded by $\mathcal{O}(n\log(1/\varepsilon))$ since, as in the proof of Lemma 3.5, we only need to partition the jobs into the $\log(m)$ sets and, in each of these sets, we need to find only the $1/\varepsilon$ size defining jobs. The partition can be done in $\mathcal{O}(n)$, provided we can calculate the logarithm of $\lfloor R/r(j)\rfloor$ in $\mathcal{O}(1)$, while the search for the size defining jobs can be done in $\mathcal{O}(n\log(1/\varepsilon))$ for all of the sets.

Note that the assumption that we can find $\log(\lfloor R/r(j)\rfloor)$ in $\mathcal{O}(1)$ is reasonable for all $r(j) \geqslant R/m$ since, in $O(m)$, we can provide a table of size m that contains as entries the size of the corresponding logarithm. This table needs to be constructed only once and afterward the logarithm can be found in $\mathcal{O}(1)$ by a simple lookup.

Now consider the number of resource requirements larger than εT. These jobs are partitioned into at most $\mathcal{O}(\log(1/\varepsilon))$ sets and, for each of these sets, we generated at most $1/\varepsilon$ different sizes. Hence the total number of different resource requirements larger than εT is bounded by $\mathcal{O}(\log(1/\varepsilon)/\varepsilon)$. □

6.3.2 Preemptive Schedule – Step (iii)

For this rounded instance $I'_{\text{sup},\varepsilon'}$, we have to solve $\text{LP}_{\text{split}}(I'_{\text{sup},\varepsilon'},\varepsilon')$. However, since in $I'_{\text{sup},\varepsilon'}$ the narrow jobs are rounded as well, we need to change our approach. The main problem is that as the block-problem $\mathcal{ABS}$ is no longer the problem Knapsack With Cardinality Constraint. Instead, we have to consider the unbounded variant of this problem. In Section 6.4, we consider the the algorithm of Mastrolilli and Hutter [86] in more detail and show that it can be extended to the unbounded version. As a result, as stated in Theorem 6.12, the corresponding $\mathcal{ABS}$ problem can be solved in $\mathcal{O}(|\mathcal{J}(I'_{\text{sup},\varepsilon'})| + \log(m)/\varepsilon^4) = \mathcal{O}(\log(m)/\varepsilon^4)$. Therefore, we can find a

basic solution to $\mathrm{LP}_{\mathrm{split}}(I'_{\mathrm{sup},\varepsilon'}, \varepsilon')$ in at most

$$\begin{aligned}
&\mathcal{O}(|\mathcal{J}(I'_{\mathrm{sup},\varepsilon'})|(\ln(|\mathcal{J}(I'_{\mathrm{sup},\varepsilon'})|) + 1/\varepsilon^2)(\log(m)/\varepsilon^4 + |\mathcal{J}(I'_{\mathrm{sup},\varepsilon'})|^{1.5356})) \\
=&\mathcal{O}((\log(m)/\varepsilon)(\ln(\log(m)/\varepsilon) + 1/\varepsilon^2)(\log(m)/\varepsilon^4 + (\log(m)/\varepsilon)^{1.5356})) \\
\leqslant&\mathcal{O}(\log(m)^2/\varepsilon^7 + \log(m)^3/\varepsilon^5).
\end{aligned}$$

Corollary 6.9. *We can find a solution to* $\mathrm{LP}_{\mathrm{split}}(I'_{\mathrm{sup},\varepsilon'}, \varepsilon')$ *that uses at most* $\mathcal{O}(\log(m)/\varepsilon)$ *non-zero components in* $\mathcal{O}(\log(m)^2/\varepsilon^7 + \log(m)^3/\varepsilon^5)$.

6.3.3 Reducing the Number of Configurations – Step (vi)

The Step (v), where we transform the solution of $\mathrm{LP}_{\mathrm{split}}(I'_{\mathrm{sup},\varepsilon'}, \varepsilon')$ to a solution of $\mathrm{LP}_W(I, \mathcal{C}_W, \mathcal{W})$ remains the same. However, in this section, we change the rounding of the resource requirements of the windows and thus we are able to reduce the number of different windows to $1/\varepsilon' + 1$:

Lemma 6.10. *Let* $m \geqslant 1/\varepsilon'^2$. *Given a solution* $(\tilde{x}, \tilde{y})$ *to* $\mathrm{LP}_W(I, \mathcal{C}_W, \mathcal{W}_{\mathrm{split}})$, *we can find a set* $\mathcal{W}' \subseteq \mathcal{W}_{\mathrm{split}}$ *with* $|\mathcal{W}'| \leqslant 1/\varepsilon' + 1$ *and a solution* $(\bar{x}, \bar{y})$ *to* $\mathrm{LP}_W(I, \mathcal{C}_W, \mathcal{W}')$, *which fulfills*

$$p(\bar{x}) \leqslant (1 + 2\varepsilon')p(\tilde{x}) \tag{6.14}$$

and contains at most $|\mathcal{J}_N| + |\mathcal{J}_W| + 2|\mathcal{W}'| + 1$ *non zero components in*

$$\mathcal{O}((|\mathcal{J}| + |\mathcal{W}'|)^{1.5356}|\mathcal{J}||\mathcal{W}'|)$$

operations.

Proof. In the first step, we modify all windows except the window (R, m) such that they use at most $m - 1/\varepsilon'$ machines. This allows us to neglect the number of machines when we are rounding the windows resource requirement since each configuration of wide jobs needs at most $1/\varepsilon'$ machines.

Let us look at a window w. Let J_w be the set of job parts scheduled in w; more precisely for each job in $\mathcal{J}_N$ with $y_{j,w} > 0$ we define a job $j_w \in J_w$ with processing time $p(j_w) = y_{j,w}$ and resource requirement $r(j_w) = r(j)$. We sort the jobs in J_w by decreasing resource requirement and build stacks of height $p(w)$ exactly, similar as in the generation of an integral solution, but this time we allow jobs to be split horizontally.

Now, we look at the number of generated stacks. If there are less than $m - 1/\varepsilon'$ stacks, we set w_m to $m - 1/\varepsilon'$ without violating the equation (6.8). On the other hand, if there are $(m - x)$ stacks with $x \in \{0, \ldots, 1/\varepsilon' - 1\}$, we remove the $(1/\varepsilon' - x)$ stacks with the smallest resource requirement, set w_m to $(m - 1/\varepsilon')$ without violating the equation (6.8), and place the removed jobs into the window (R, m). Let $\mathcal{J}_{\text{shift}}$ be the set of (fractional) jobs we moved to window (R, m).

Claim. If we schedule the jobs $\mathcal{J}_{\text{shift}}$ in window (R, m), we have to add at most $\varepsilon' p(\tilde{x})$ processing time to $\tilde{x}_{(\emptyset,(R,m))}$ to get a feasible solution to LP_W.

Proof. The window (R, m) is the only window where we have added jobs. We have to show $m\varepsilon' p(\tilde{x}) \geqslant p(\mathcal{J}_{\text{shift}})$ and that $R\varepsilon' p(\tilde{x}) \geqslant \sum_{j \in \mathcal{J}_{\text{shift}}} r(j)p(j)$ to prove that the inequalities (6.8) and (6.9) hold which then proves the claim.

Let us look at the stacks of jobs we removed in window w. First, we show that the summed up resource requirement of the jobs at the bottom of these stacks is at most $\varepsilon' R$. Consider the jobs with smallest resource requirement per stack. Their total resource requirement R_s has to be smaller than $r(w)$. Otherwise the total resource requirement of the jobs is in window w larger than $r(w) \cdot p(w)$, which cannot be the case since equation (6.9) holds for the considered solution. The total resource requirement of all the jobs with largest resource requirement per stack, except for the first stack has a resource requirement less than R_s. As a consequence, this resource requirement is at most $r(w)$. Furthermore, we know that $r(w) \leqslant R - \varepsilon' R$ since the considered window is not the window (R, m) and hence there has to be a wide job scheduled at the same time as the window. Since the job at the bottom of the widest stack has a resource requirement of less than $\varepsilon' R$ the summed resource requirement of all widest jobs per stack is strictly smaller than R. Assume that the jobs at the bottom of the $1/\varepsilon' - x$ smallest stacks have a summed resource requirement of more than $\varepsilon' R$. In this case, one of these jobs has a resource requirement of at least $\varepsilon' R/(\varepsilon'^{-1} - x) \geqslant \varepsilon'^2 R$. Hence, the at least $m - 1/\varepsilon'$ other stacks have a job with resource requirement of at least $\varepsilon'^2 R$ at the bottom. Therefore, the total resource requirement of the jobs at the bottom of the stacks is at least $(m - 1/\varepsilon')\varepsilon'^2 R + \varepsilon' R \geqslant (1/\varepsilon'^2 - 1/\varepsilon')\varepsilon'^2 R + \varepsilon' R = R$, which is a contradiction since the total resource requirement of these stacks

is strictly less than R.

Let $\mathcal{J}_{\text{shift},w}$ be the set of jobs in the stacks we remove from w. Since the summed up resource requirement of the jobs at the bottom of these stacks is at most $\varepsilon' R$, we know that $\varepsilon' R \cdot p(w) \geqslant \sum_{j \in \mathcal{J}_{\text{shift},w}} r(j)p(j)$. The total processing time of these stacks is at most $\sum_{j \in \mathcal{J}_{\text{shift},w}} p(j) = p(\mathcal{J}_{\text{shift},w}) \leqslant x \cdot p(w) \leqslant 1/\varepsilon' \cdot p(w) \leqslant m\varepsilon' p(w)$ since $m \geqslant 1/\varepsilon'^2$. If we sum up the changes for all windows we get:

$$R\varepsilon' p(\tilde{x}) \geqslant \sum_{w \in \mathcal{W} \setminus (R,m)} R\varepsilon' p(w) \geqslant \sum_{w \in \mathcal{W} \setminus (R,m)} \sum_{j \in \mathcal{J}_{\text{shift},w}} r(j)p(j) = \sum_{j \in \mathcal{J}_{\text{shift}}} r(j)p(j)$$

and

$$m\varepsilon' p(\tilde{x}) \geqslant \sum_{w \in \mathcal{W} \setminus (R,m)} m\varepsilon' p(w) \geqslant \sum_{w \in \mathcal{W} \setminus (R,m)} p(\mathcal{J}_{\text{shift},w}) = p(\mathcal{J}_{\text{shift}}),$$

which proves the claim. ◁

For each window $w \in \mathcal{W}$ we move each job $j \in \mathcal{J}_{\text{shift},w}$ to the window (R, m), by setting $\bar{x}_{(\varnothing,(R,m))} := \tilde{x}_{(\varnothing,(R,m))} + \varepsilon' P_{\text{pre}}$ and $\bar{y}_{j,w} := 0$ and $\bar{y}_{j,(R,m)} := \bar{y}_{j,w}$. Now, we can set $m(w) := m - 1/\varepsilon'$ for all windows in $\mathcal{W}_{\text{split}}$ and adjust the solution respectively.

We look again at the resource requirement of the windows. Since all windows have the same number of machines they use, we do not need to partition the set of generalized configurations by the number of wide jobs they contain since the number of jobs is no longer a restriction to the possibility to schedule configuration and window at the same time. Therefore, we build just one stack of generalized configurations, in increasing order of resource requirements of the configurations. We now shift the windows exactly $\varepsilon' P_{\text{pre}}$ downwards like in the original rounding step and round the windows like before. By this rounding, we get $\varepsilon'^{-1} + 1$ different window sizes and have to extend the window (R, m) by $\varepsilon' P_{\text{pre}}$. The solution $(\bar{x}, \bar{y})$ and the set $\mathcal{W}'$ can be constructed analogously to the previous rounding. The number of operations is given equally as in Lemma 6.5. □

6.3.4 Integral Solution – Step (vi)

In this section, we describe how to place the jobs inside the configurations and windows. While the wide jobs can be placed as before, we need to be more careful in the analysis of placing the narrow jobs.

Lemma 6.11. *Let $(\bar{x}, \bar{y})$ be a basic solution to $\mathrm{LP}_W(I_{\mathrm{sup}}, \mathcal{C}_W, \mathcal{W}')$ such that the total number of fractional scheduled narrow jobs and used configurations is bounded by an integer K. There is a solution which places the jobs in $\mathcal{J}$ integrally and has a makespan of at most*

$$(1+\varepsilon')p(\bar{x}) + (1+\varepsilon'^{-1} + K)p_{\max}.$$

This solution can be found in at most $\mathcal{O}(\log(m)/\varepsilon \log(\log(m)/\varepsilon) + K/\varepsilon + n)$ operations.

Proof. First we modify the solution $(\bar{x}, \bar{y})$ to have enough space to schedule the jobs integrally: We define $\hat{x}_{(C,w)} := \bar{x}_{(C,w)} + p_{\max}$ if $\bar{x}_{(C,w)} > 0$, and $\hat{x}_{(C,w)} = 0$ otherwise and place the wide jobs exactly as in the Lemma 6.6. This alteration and the schedule of the wide jobs can be done in $\mathcal{O}(K/\varepsilon + |\mathcal{J}(I)|)$.

To place the narrow jobs, we first determine for each window which rounded jobs stay inside the window and which do not. For this purpose, we consider for each window the jobs that are to be scheduled (fractionally) inside them. Similar as in the proof of Lemma 6.6 for a given window w, we sort the corresponding jobs by their resource requirement and build stacks of height $p(w)$ by scheduling the job that overlaps $p(w)$ fractionally. This can be done in $\mathcal{O}(|\mathcal{J}(I_{\mathrm{sup}})| \log(|\mathcal{J}(I_{\mathrm{sup}})|) + K) = \mathcal{O}(\log(m)/\varepsilon \log(\log(m)/\varepsilon))$. As in Lemma 6.6, we move the stack containing the widest jobs to the end of the schedule.

Afterward, we replace the rounded jobs by the jobs from the set $\mathcal{J}(I)$. Consider a rounded job $j \in \mathcal{J}(I_{\mathrm{sup}})$. This job was introduced for a set of jobs $\mathcal{J}_{i,l}$. Assume the job was scheduled inside $z > 1$ windows. For each of these windows w, we greedily replace the job j by jobs from the set $\mathcal{J}_{i,l}$ in arbitrary order such that they may overlap the border $p(w)$ if the job was split at this line. Assume the next job $j' \in \mathcal{J}_{i,l}$ does not fit inside the window w, because the the area reserved for j is used up by the already placed jobs in $\mathcal{J}_{i,l}$. In this case there has to be another part of j that is not

filled up with the jobs from $\mathcal{J}_{i,l}$ (because the LP solution reserves enough processing time for the jobs in $\mathcal{J}_{i,l}$). We schedule the job j' at the end of the schedule and proceed to schedule the residual jobs from $\mathcal{J}_{i,l}$ inside the next part. Since the job j is scheduled inside at most z windows, we shift at most $z-1$ jobs to the end of the schedule.

For each fraction in a narrow job and each configuration, we add at most $p_{\max}$ processing time to the schedule and, hence, we have

$$p(\hat{x}) \leqslant p(\bar{x}) + Kp_{\max}.$$

In total these step need at most

$$\begin{aligned}&\mathcal{O}(|\mathcal{J}(I_{\sup})|\log(|\mathcal{J}(I_{\sup})|) + K/\varepsilon + |\mathcal{J}(I)|)\\ &= \mathcal{O}(\log(m)/\varepsilon \log(\log(m)/\varepsilon) + K/\varepsilon + n)\end{aligned}$$

operations. □

6.3.5 Summary of the Modified Algorithm

In the following we summarize the steps of the algorithm. First, the modified algorithm computes the rounded instance $I'_{\sup,\varepsilon'}$ using the rounding described for Lemma 6.7 in $\mathcal{O}(n/\varepsilon)$. Afterward it uses the algorithm described in the proof of Lemma 6.2 to find an approximate solution to LP_{split} with value $(1+\varepsilon')\text{OPT}_{\text{split}}(I'_{\sup,\varepsilon'},\varepsilon')$ in $\mathcal{O}(\log(m)^3/\varepsilon^7)$ that has at most $\mathcal{O}(\log(m)/\varepsilon)$ non zero components. By Lemma 6.7, we know that $(1+\varepsilon')\text{OPT}_{\text{split}}(I'_{\sup,\varepsilon'},\varepsilon') \leqslant (1+\varepsilon')(1+3\varepsilon')\text{OPT}(I)$. This solution is transformed to a solution to LP_W without losing any factor in the approximation with the techniques from Lemma 6.4. This can be done in $\mathcal{O}((\log(m)/\varepsilon)^2)$ since each configuration contains at most $\mathcal{O}(\log(m)/\varepsilon)$ different jobs. In the next step, the algorithm reduces the number of widows with the techniques from Lemma 6.10. The rounding of the window sizes is possible in $\mathcal{O}((\log(m)/\varepsilon)^3)$: We have to sort the $\mathcal{O}(\log(m)/\varepsilon)$ generalized configurations and in each generalized configuration, we have to sort the $\mathcal{O}(\log(m)/\varepsilon)$ jobs that are contained in the $\mathcal{O}(\log(m)/\varepsilon)$ windows. Since $|\mathcal{W}'| \leqslant 1+1/\varepsilon'$ and $|\mathcal{J}_W| \in \mathcal{O}(\log(1/\varepsilon')/\varepsilon')$, the linear program has $|\mathcal{J}_{\sup W}| + |\mathcal{J}_N| + 2|\mathcal{W}'| \in \mathcal{O}(\log(m)/\varepsilon')$ constraints and at most $|\mathcal{C}_W||\mathcal{W}'| + |\mathcal{J}_N||\mathcal{W}'| \in \mathcal{O}(\log(m)/\varepsilon^2)$ variables. Therefore, we can

compute a basic solution with at most $|\mathcal{J}_{\sup W}| + |\mathcal{J}_N| + 2|\mathcal{W}'| + 1$ non zero components in $\mathcal{O}((\log(m)/\varepsilon')^{1.5356} \log(m)/\varepsilon^2) \leqslant \mathcal{O}(\log(m)^{2.5356}/\varepsilon^{3.5356})$ operations. Since we have at most $|\mathcal{J}_{\sup W}| + |\mathcal{J}_N| + 2|\mathcal{W}'| + 1$ non zero components, the number of configurations and fractional scheduled narrow jobs is bounded by $\mathcal{O}(\log(1/\varepsilon')/\varepsilon')$. The makespan of this solution is bounded by $(1+\varepsilon')(1+3\varepsilon')^2\mathrm{OPT}(I)$. With Lemma 6.11 we get an integral schedule for the jobs in $I_{\sup}$ with makespan at most $(1+\varepsilon')^2(1+3\varepsilon')^2\mathrm{OPT}(I) + \mathcal{O}((\log(1/\varepsilon')/\varepsilon') \cdot p_{\max})$. If we set $\varepsilon' := \varepsilon/12$ we get a schedule with makespan of at most:

$$(1+\varepsilon)\mathrm{OPT} + \mathcal{O}(p_{\max} \log(1/\varepsilon)/\varepsilon)$$

The total running time of this algorithm is bounded by

$$\mathcal{O}(n/\varepsilon + \log(m)^3/\varepsilon^7) \leqslant \mathcal{O}(n/\varepsilon + \log(n)^3/\varepsilon^7)$$

with $n \geqslant m > 1/\varepsilon^2$.

The case $m \leqslant 1/\varepsilon'^2$ Let $m \leqslant 1/\varepsilon'^2$. Again this case is the simpler one. We redefine the sets of wide and narrow jobs. Let $\mathcal{J}_W := \{j \in \mathcal{J} | r(j) > R/m\}$ and $\mathcal{J}_N := \{j \in \mathcal{J} | r(j) \leqslant R/m\}$. Again we round all the jobs with the technique described for Lemma 6.2, which can be done in $\mathcal{O}(n/\varepsilon)$. Now we have at most $\mathcal{O}(\log(m)/\varepsilon') \leqslant \mathcal{O}(\log(1/\varepsilon')/\varepsilon')$ sizes in the rounded instance $I'_{\sup,\varepsilon'}$. By Corollary 6.9, we can find a preemptive schedule with at most $\mathcal{O}(\log(1/\varepsilon')/\varepsilon')$ non zero components and makespan of at most $(1+\varepsilon')\mathrm{OPT}_{\mathrm{split}}(I'_{\sup,\varepsilon'}) \leqslant (1+\varepsilon')(1+3\varepsilon')\mathrm{OPT}_{\mathrm{split}}(I)$ in $\mathcal{O}(\log(m)^2/\varepsilon^7 + \log(m)^3/\varepsilon^5) = \mathcal{O}(\log(1/\varepsilon')^2/\varepsilon^7)$ operations. By Lemma 6.6, we can find an integral schedule with makespan at most

$$(1+\varepsilon')(1+3\varepsilon')\mathrm{OPT}_{\mathrm{split}}(I) + \mathcal{O}(p_{\max} \log(1/\varepsilon')/\varepsilon').$$

If we set $\varepsilon' := \varepsilon/6$ we get a schedule with makespan of at most

$$(1+\varepsilon)\mathrm{OPT} + \mathcal{O}(p_{\max} \log(1/\varepsilon)/\varepsilon)$$

in at most $\mathcal{O}(n/\varepsilon + \log(1/\varepsilon')^2/\varepsilon^7)$ operations. This proves Theorem 6.1.

6.4 Remark on the Unbounded Knapsack with Cardinality Constraint

In the Problem Unbounded Knapsack With Cardinality Constraint (UkKP) we are given a set of items $I = \{1, \dots, n\}$ as well as a knapsack with capacity c and cardinality constraint k. Each item $i \in I$ has a profit $p(i)$ and a wight $w(i)$. The objective is to find a multiset of the items which can be placed into the knapsack without exceeding the capacity or the cardinality constraint and maximizes the profit, i.e. the objective is to find an optimal solution to the following ILP:

$$\begin{aligned} \max \sum_{i \in I} x_i p(i) & \\ \sum_{i \in I} x_i w(i) &\leqslant c \\ \sum_{i \in I} x_i &\leqslant k \\ x_i \in \mathbb{N} & \qquad \forall i \in I \end{aligned}$$

To our knowledge, the unbounded case has not been studied. For the $\{0,1\}$-version an FPTAS with running time of $\mathcal{O}(n + kz^2/\varepsilon^2)$, where $z = \min\{k, 1/\varepsilon\}$, was presented by Mastrollili and Hutter [86]. By applying the techniques from this algorithm to the unbounded case, we can proof the following lemma.

Theorem 6.12. *There is a PTAS for Unbounded Knapsack With Cardinality Constraint that has a running time of $\mathcal{O}(n + \log(k)z^2/\varepsilon^2)$, where $z = \min\{k, 1/\varepsilon\}$.*

We will give a short overview on the steps of the algorithm in [86] and the alterations to make such that it applies to the unbounded case as well. For the detailed algorithm we refer to [86].

The first step of the algorithm in [86] is to find a solution to the relaxed LP version of the problem, where each variable can be number in $[0,1]$. This is possible in $\mathcal{O}(n)$ using the algorithm by Megiddo and Tamir [87]. Caprara et al. [17] showed that an optimal solution to the relaxed $\{0,1\}$-version of the problem, where only two variables are nither 0 nor 1, can be transformed to a solution with profit $P \leqslant \mathrm{OPT}_{\{0,1\}} \leqslant P + p_{\max} \leqslant 2P$, where $p_{\max}$ is the largest occurring profit.

Note that a basic solution to the relaxed version of the above linear program has only two non zero components and can be found in linear time using the algorithm described in [87]. We can interpret this solution as a solution to the $\{0,1\}$-version of this problem by duplicating each of the two items corresponding to the non zero components at most k times such that all items are either not choose or used fully in the solution except for two items that can be used fractionally. This solution can then be transformed to a solution with profit $P \leqslant \mathrm{OPT}_{\{0,1\}} \leqslant P + p_{\max} \leqslant 2P$, using the same technique as in [17].

In the next step of the algorithm in [86] the item profits are rounded to $\mathcal{O}(\log(k)/\varepsilon)$. In our alteration, we use the same rounding technique. However, we have to remember just one item per profit size and not up to k as in [86], which improves the number of items in the rounded solution from $\mathcal{O}(k/\varepsilon)$ to $\mathcal{O}(\log(k)/\varepsilon)$. Afterward as in [86], the at most $\mathcal{O}(\log(z)/\varepsilon)$ items with a profit larger than $\mathcal{O}(\varepsilon)OPT$ are rounded a second time arithmetically to at most $\mathcal{O}(z/\varepsilon)$ different profit sizes in $\{i\mathcal{O}(\varepsilon/z)P \mid i = 1,\ldots,\mathcal{O}(z/\varepsilon)\}$.

The residual steps are almost identical to [86]. In a dynamic program, we compute the minimal weight to score profit $i\mathcal{O}(\varepsilon/z)P$ with l items for all possible combinations

$$(i,l) \in \{1,\ldots,\mathcal{O}(z/\varepsilon)\} \times \{1,\ldots,z\}.$$

To compute a table entry for (i,l) each large item has to be considered once and therefore at most $\mathcal{O}(\log(z)/\varepsilon)$ operations are needed. Since the dynamic program has at most $\mathcal{O}(z^2/\varepsilon)$ cells, the dynamic program needs at most $\mathcal{O}(z^2/\varepsilon \cdot \log(z)/\varepsilon)$ operations.

For each table entry, we consider the residual capacity and the residual number of items and solve the corresponding linear program for the small items in $\mathcal{O}(\log(k)/\varepsilon)$. Generating the corresponding integral solution, we loose at most $\mathcal{O}(\varepsilon)$OPT since the small items have a profit of at most $\mathcal{O}(\varepsilon)$OPT. In total, the running time is bounded by

$$\mathcal{O}(n + z^2/\varepsilon \cdot (\log(z)/\varepsilon + \log(k)/\varepsilon)) \leqslant \mathcal{O}(n + z^2 \log(k)/\varepsilon^2).$$

Chapter 7

A Tight Polynomial Time Approximation for Single Resource Constraint Scheduling

In this chapter, we consider three algorithms for Single Resource Constraint Scheduling. It is NP-hard to approximate this problem better than $3/2$. On the other hand, the best algorithm so far has an absolute approximation ratio of $2 + \varepsilon$.

First, we present an APTAS with additive term $p_{\max}$. The techniques from this APTAS are used to construct an algorithm with absolute approximation ratio $(3/2 + \varepsilon)$, which closes the gap between inapproximability and best algorithm with exception of a negligible small ε. Finally, we improve the running time of the APTAS such that ε does no longer appear in the exponent of the instance size, i.e., we present an AEPTAS for Single Resource Constraint Scheduling with additive term $p_{\max}$.

The results in the chapter have not been published so far.

7.1 Results

Remember, in the Single Resource Constraint Scheduling (SRCS), we are given m identical machines, a discrete renewable resource with a fixed size $R \in \mathbb{N}$ and a set of n jobs $\mathcal{J}$. Each job has a processing time $p(j) \in \mathbb{N}$. We define the total processing time of a set of jobs $\mathcal{J}' \subseteq \mathcal{J}$ as $p(\mathcal{J}') := \sum_{j \in \mathcal{J}'} p(j)$. To be scheduled, each job $j \in \mathcal{J}$ needs one of the machines as well as a fix amount $r(j) \in \mathbb{N}$ of the resource which it will allocate during the complete processing time $p(j)$ and which is deallocated as soon

as the job has finished its processing. Neither machine nor any part of the resource can be allocated by two different jobs at the same time. We define the area of a job as $\text{area}(j) := r(j) \cdot p(j)$ and the area of a set of jobs $\mathcal{J}' \subseteq \mathcal{J}$ as $\text{area}(\mathcal{J}') := \sum_{j \in \mathcal{J}'} r(j) \cdot p(j)$.

A schedule $\sigma : \mathcal{J} \to \mathbb{N}$ maps each job $j \in \mathcal{J}$ to a starting point $\sigma(j) \in \mathbb{N}$. We say a schedule is feasible if

$$\forall t \in \mathbb{N} : \sum_{j : t \in [\sigma(j), \sigma(j)+p(j))} r(j) \leqslant R \text{ and}$$
$$\forall t \in \mathbb{N} : \sum_{j : t \in [\sigma(j), \sigma(j)+p(j))} 1 \leqslant m.$$

When these two conditions hold, we can generate a schedule where each machine and each resource part is allocated by at most one job at a time. The objective is to find a feasible schedule, which minimizes the total length of the schedule called makespan, i.e., we have to minimize $\max_{j \in \mathcal{J}} \sigma(j) + p(j)$.

Note that the algorithm with the best absolute ratio so far is a $(2+\varepsilon)$-approximation [92]. Furthermore, they present a PTAS for the case that m is constant. As a first step, we present an APTAS for this problem, which has an approximation guarantee of $(1+\varepsilon)\text{OPT} + p_{\max}$. Note that this algorithm is a $(2+\varepsilon)$-approximation as well, but even improves on this ratio, in the case that $p_{\max}$ is strictly smaller than OPT.

Theorem 7.1. *There is an APTAS for Single Resource Constraint Scheduling with an additive term $p_{\max}$ and running time*

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

In the AFPTAS, almost all jobs are completed before $(1+\mathcal{O}(\varepsilon))\text{OPT}$, except for a small group $\mathcal{J}'$ of jobs that are all started simultaneously at $(1+\mathcal{O}(\varepsilon))\text{OPT}$, after the processing of all other jobs is finished. The processing of this set $\mathcal{J}'$ causes the additive term $p_{\max}$. We managed to find a $(3/2+\varepsilon)$-approximation, by handling a set of so called huge jobs (which have a processing time larger than $\text{OPT}/2$) more carefully and, thus, preventing that one of these jobs is contained in this group of jobs $\mathcal{J}'$.

Theorem 7.2. *There is an algorithm for Single Resource Constraint Scheduling with approximation ratio* $(3/2+\varepsilon)$ *and running time*

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

The $(3/2+\varepsilon)$-approximation highly depends on the fact that the set $\mathcal{J}'$ only contains $\mathcal{O}_\varepsilon(1)$ jobs. When we dismiss this necessity, we can improve the running time of the APTAS to an AEPTAS.

Theorem 7.3. *There is an AEPTAS for Single Resource Constraint Scheduling with additive term* $p_{\max}$ *and running time* $\mathcal{O}(n\log(1/\varepsilon) + m\cdot\mathcal{O}_\varepsilon(1))$.

Note that in all of these algorithms the exponent is at most doubly exponential in $1/\varepsilon$, i.e., it is bounded by $(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}$.

Methodology and Organization of this Chapter We will present the algorithms one after another. In Section 7.2, we will present the APTAS from Theorem 7.1. The algorithm follows the following general approach. First, we simplify the instance by rounding the processing time of the jobs and partitioning them into large, medium, and small corresponding to their processing time. Afterward, we use linear programming approaches to find a placement of these jobs inside the optimal packing. The few jobs that are placed fractional with this linear program will be placed on top of the packing.

Afterward, in Section 7.3, we present the $(3/2+\varepsilon)$-approximation and prove Theorem 7.2. However, instead of placing the fractional jobs on top of the packing, we stretch the packing by $(1/2+\mathcal{O}(\varepsilon))$OPT, and place the fractional scheduled large jobs inside a gap in this stretched schedule. This stretching allows us to define a common finishing point of all the jobs, which have a processing time larger than OPT/2 and, thus, we avoid to schedule them fractionally with the linear program.

Finally, in Section 7.4, we present the algorithm from Theorem 7.3. The key to improve the running time is a more elaborate rounding of the resource requirements of the large jobs. As a consequence of this rounding, the number of jobs that have to be discarded and scheduled in the end as the set $\mathcal{J}'$ will depend on the number of machines m, while before it only was dependent on $1/\varepsilon$. Hence this improvement of the running time is

not transferable to the $(3/2+\varepsilon)$-approximation, where we need the their number to be bounded by $\mathcal{O}_\varepsilon(1)$.

7.2 APTAS with additive term $p_{\max}$

In this section, we present an asymptotic APTAS for the Single Resource Constraint Scheduling problem which has an approximation guarantee of $(1+\varepsilon)\mathrm{OPT} + p_{\max}$ and a running time of $\mathcal{O}(n\log(n)) + (m\log(R))^{\mathcal{O}_\varepsilon(1)}$, i.e., we prove Theorem 7.1 in this section. Furthermore, we will prove that there is a PTAS for the case that m is bounded by a constant. This algorithm represents the base for the $3/2+\varepsilon$-approximation, see Section 7.3, and its running time can be improved to be efficient in the sense that $1/\varepsilon$ does not appear in the exponent of n, m or $\log(R)$, see Section 7.4. In the following sections, we assume that $1/\varepsilon \in \mathbb{N}$. Remember that we can assume that $n > m$ since, otherwise, the machine constraint can never be violated and, thus, the problem is reduced to Parallel Task Scheduling.

The general structure of the algorithm can be summarized as follows. First, we find an estimate on the makespan of the schedule and use it, to simplify the instance. This simplification is done by partitioning the jobs into large, medium, and small jobs dependent on their processing times and rounding their processing times and the resource requirement of the small jobs, see Section 7.2.1. Afterward, we use a binary search framework and a dual approximation approach. Given a target makespan T', we guess the structure of the optimal solution and try to place the jobs according to this guess. The placement of the three groups of jobs (i.e., large, medium, and small) happens independently and is described in Sections 7.2.2, 7.2.5, and 7.2.4 respectively. A more detailed summary of the algorithm can be found in Section 7.2.6.

7.2.1 Simplifying the input instance

In the first step of the algorithm, we simplify the given instance such that it has few jobs or few job sizes. Consider the lower bound on the optimal makespan $T := \min\{p_{\max}, \mathrm{area}(\mathcal{J})/R, p(\mathcal{J})/m\}$. By the analysis of the greedy List-Schedule, as described in [92], we know that the optimal

schedule has a size of at most $\frac{1}{m}p(\mathcal{J}) + \frac{2}{R}\text{area}(\mathcal{J}) + p_{\max} \leqslant 4T$.

In the first simplification step, we round up the processing times of all the jobs to multiples of $\varepsilon T/n$. By Lemma 3.1, we know that this lengthens the optimal schedule by at most εT, and hence, in this rounded instance, T is still a lower bound on the optimal makespan, but the upper bound is now $(4+\varepsilon)T$. Hence, the optimal schedule has size $i \cdot \varepsilon T/n$, for $i \in [n/\varepsilon, 4n/\varepsilon + n] \cap \mathbb{N}$. Furthermore, in this rounded instance, it holds that $\text{area}(\mathcal{J})/R \leqslant (1+\varepsilon)T$ as well as $p(\mathcal{J})/m \leqslant (1+\varepsilon)T$.

In the next step, we will create a gap between jobs with a large processing time and jobs with a small processing time, by removing a set of medium sized jobs.

Lemma 7.4. *Consider the sequence $\sigma_0 = \varepsilon$, $\sigma_{i+1} = \sigma_i\varepsilon^4$. There exists an $i \in \{1, \ldots, 1/\varepsilon\}$ such that*

$$\frac{1}{m}p(\mathcal{J}_{\sigma_i}) + \frac{2}{R}\text{area}(\mathcal{J}_{\sigma_i}) \leqslant \varepsilon\left(\frac{1}{m}p(\mathcal{J}) + \frac{2}{R}\text{area}(\mathcal{J})\right),$$

where $\mathcal{J}_{\sigma_i} := \{j \in \mathcal{J} \mid p(j) \in [\sigma_i T, \sigma_{i-1}T)\}$ and we can find this i in $\mathcal{O}(n + 1/\varepsilon)$.

Proof. This follows directly by the pigeonhole principle. Note that the $1/\varepsilon$ sets $\mathcal{J}_{\sigma_i}$ are disjoint. Assume for contradiction that it holds for each $i \in \{1, \ldots, 1/\varepsilon\}$ that

$$\frac{1}{m}p(\mathcal{J}_{\sigma_i}) + \frac{2}{R}\text{area}(\mathcal{J}_{\sigma_i}) > \varepsilon\left(\frac{1}{m}p(\mathcal{J}) + \frac{2}{R}\text{area}(\mathcal{J})\right).$$

As a consequence, it holds that

$$\sum_{i=1}^{1/\varepsilon}\left(\frac{1}{m}p(\mathcal{J}_{\sigma_i}) + \frac{2}{R}\text{area}(\mathcal{J}_{\sigma_i})\right) > \frac{1}{m}p(\mathcal{J}) + \frac{2}{R}\text{area}(\mathcal{J}).$$

This is a contradiction since

$$\begin{aligned}\sum_{i=1}^{1/\varepsilon}\left(\frac{1}{m}p(\mathcal{J}_{\sigma_i}) + \frac{2}{R}\text{area}(\mathcal{J}_{\sigma_i})\right) &= \frac{1}{m}p\left(\bigcup_{i=1}^{1/\varepsilon}\mathcal{J}_{\sigma_i}\right) + \frac{2}{R}\text{area}\left(\bigcup_{i=1}^{1/\varepsilon}\mathcal{J}_{\sigma_i}\right)\\ &\leqslant \frac{1}{m}p(\mathcal{J}) + \frac{2}{R}\text{area}(\mathcal{J}).\end{aligned}$$

Last, we argue that we can find this set in $\mathcal{O}(n + 1/\varepsilon)$. Note that in the

rounded instance there are at most n different sizes smaller than εT since $\varepsilon T/(\varepsilon T/n) = n$. Hence, for all the possible sizes smaller than εT, we can construct a table, where we save to which set this size belongs in $\mathcal{O}(n)$. Afterward, we iterate the set of jobs once and for each job we add that value $p(j)/m + 2\text{area}(j)/R$ to the size of the corresponding set $\mathcal{J}_{\sigma_i}$, if it belongs to one of these. □

Let $i \in \{1, \ldots, 1/\varepsilon\}$ be the smallest value such that $\mathcal{J}_{\sigma_i}$ has the property from Lemma 7.4 and define $\mu := \sigma_i$ and $\delta := \sigma_{i-1}$. Note that $\sigma_i = \varepsilon^{1+4i}$ and hence $\delta \geqslant \varepsilon^{4/\varepsilon+1}$. Using these values for δ and μ, we partition set of jobs into large $\mathcal{J}_L := \{j \in \mathcal{J} | p(j) \geqslant \delta T\}$, small $\mathcal{J}_S := \{j \in \mathcal{J} | p(j) < \mu T\}$ and medium $\mathcal{J}_M := \{j \in \mathcal{J} | \mu T \leqslant p(j) < \delta T\}$.

The final simplification step is to round the processing times of the large jobs using the rounding in Lemma 3.2 to multiples of $\varepsilon\delta T$. In this step, we reduce the number of different processing times of large jobs to $\mathcal{O}(\log_\varepsilon(\delta)/\varepsilon^2) = \mathcal{O}(1/\varepsilon^3)$. However, we lengthen the schedule at most by the factor $(1 + 2\varepsilon)$. Furthermore, this rounding reduces the starting times of the jobs to at most $\mathcal{O}(1/(\varepsilon\delta))$ possibilities since all the large jobs start and end at multiples of $\varepsilon\delta T$ and the optimal height of the rounded instance is bounded by $(1 + 2\varepsilon)(4 + \varepsilon)T$.

Observation 7.5. After this rounding the total number of possible processing times is bounded by $\mathcal{O}(n + 1/\varepsilon^2)$.

Since the jobs with processing time of at most εT are rounded to multiples of $\varepsilon T/n$, these jobs have at most $\varepsilon T/(\varepsilon T/n) = n$ different sizes. On the other hand, the jobs with processing time larger than εT are rounded to integral multiples of $\varepsilon^2 T$ and hence they have at most $1/\varepsilon^2$ different sizes.

Let $I_{rounded}$ be the rounded instance and $\text{OPT}_{rounded}$ its optimal packing height. In the following, we assume that the considered optimal schedule that is rounded has a height $\text{OPT}_{rounded}$ of the form $i \cdot \varepsilon T$, while making an extra error of at most εT. As a result, the total height of this optimal schedule is bounded by $T \leqslant \text{OPT}_{rounded} \leqslant (1 + 2\varepsilon)(4 + \varepsilon)T + \varepsilon T \leqslant ((1 + 2\varepsilon)(4 + \varepsilon) + \varepsilon)\text{OPT}$ and hence $i \in [1/\varepsilon, 16/\varepsilon] \cap \mathbb{N}$, for $\varepsilon \leqslant 1$.

In the following, we will prove that given a value $T' := i\varepsilon T$ for some $i \in [1/\varepsilon, 16/\varepsilon] \cap \mathbb{N}$, we either can find a schedule with height $(1 + \mathcal{O}(\varepsilon))T' + p_{\max}$, or prove that there is no schedule that schedules the large and

small jobs with makespan at most T'. In the algorithm, we will find the smallest value for i such that we find a schedule of height at most $(1+\mathcal{O}(\varepsilon))i\varepsilon T + p_{\max}$ using the following dual approximation and a binary search framework.

7.2.2 Scheduling Large Jobs

In this section, we describe how to schedule the large jobs when given the size of the makespan $T' := i\varepsilon T$ for some $i \in [1/\varepsilon, 16/\varepsilon] \cap \mathbb{N}$. When scheduling this set of jobs it comes in handy that they can only start at the multiples of $\varepsilon\delta T$. Let $\mathcal{S}$ be the set of all these points in time up to T' and let $\mathcal{P}$ be the set of all rounded processing times for large jobs. The processing time between two consecutive starting times $s_i, s_{i+1} \in \mathcal{S}$ is called layer l_i. Notice that, during the processing of a layer in a rounded optimal packing, the resource requirement and number of machines used by large jobs stays unchanged since the large jobs only start and end at the starting points in $\mathcal{S}$. In this section, we assume that $m \geqslant 3|\mathcal{S}| = (1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}$. In Section 7.2.3, we describe how we can schedule the large jobs, if this is not the case.

In the first step, we partition the set of large jobs into wide and narrow jobs. Let $\alpha \in (0,1)$. Wide jobs $j \in \mathcal{J}_{L,W}$ have a resource requirement of $r(j) \geqslant \alpha R$ and narrow jobs $j \in \mathcal{J}_{LN}$ have a resource requirement of $r(j) < \alpha R$. We will specify the value of α later. Note that α might be chosen differently for the AEPTAS or the $(3/2+\varepsilon)$-approximation. There can be at most $\mathcal{O}(1/\alpha\delta)$ wide large jobs and at most $\mathcal{O}(m/\delta)$ narrow large jobs since $\text{area}(\mathcal{J})/R \leqslant (1+\varepsilon)T$ and $p(\mathcal{J})/m \leqslant (1+\varepsilon)T$. Hence, it is possible to enumerate all combinations of starting positions of wide large jobs in $\mathcal{O}((1/\varepsilon\delta)^{1/\alpha\delta})$.

To schedule the narrow large jobs with a given Makespan T', we guess for each possible starting point $s \in \mathcal{S}$ the required number of machines $m_s^{(L)}$ and the resource requirement $R_s^{(L)}$ of the large jobs that are processed between this starting point and the next in a rounded optimal solution. Since we have at most $1/\varepsilon\delta$ starting points, there are at most $(mR)^{1/\varepsilon\delta}$ possible guesses.

However, since R can be exponential in the input size, this number is too large. Instead of guessing the exact resource requirement, we guess the interval $(R/(1+1/m)^{t+1}, R/(1+1/m)^t]$ in which this re-

source requirement lies, i.e., if the resource requirement lies in the interval $(R/(1+1/m)^{t+1}, R/(1+1/m)^t]$, we define $R_s^{(L)} := R/(1+1/m)^t$. There are at most $\lceil \log_{(1+1/m)}(R) \rceil \in \mathcal{O}(\log(R)m)$ such intervals intersecting $[1, R]$ and, therefore, at most $\mathcal{O}((\log(R)m)^{\mathcal{O}(1/\varepsilon\delta)})$ possible guesses for the resource requirement.

For a given guess $(m_s^{(L)}, R_s^{(L)})_{s\in\mathcal{S}}$, we solve the following linear program $\mathrm{LP}_{\mathcal{J}_{LN},(m_s^{(L)},R_s^{(L)})_{s\in\mathcal{S}}}$.

$$\sum_{j\in\mathcal{J}_{LN}} \sum_{s'=s-p(j)+\varepsilon\delta T}^{s} r(j)x_{j,s'} \leqslant R_s^{(L)} \qquad \forall s \in \mathcal{S} \qquad (7.1)$$

$$\sum_{j\in\mathcal{J}_{LN}} \sum_{s'=s-p(j)+\varepsilon\delta T}^{s} x_{j,s'} \leqslant m_s^{(L)} \qquad \forall s \in \mathcal{S} \qquad (7.2)$$

$$\sum_{s=0}^{T'-p(j)} x_{j,s} = 1 \qquad j \in \mathcal{J}_{LN} \qquad (7.3)$$

$$x_{j,s} \geqslant 0 \qquad \forall s \in \mathcal{S}, j \in \mathcal{J}_{LN} \qquad (7.4)$$

The variable $x_{j,s}$ represents which fraction of job j starts at time s. The first two conditions ensure that in each layer the positioned jobs do not exceed the guessed number of resources or machines. The third condition ensures that each job is scheduled. If there is an optimal schedule with resource requirement and machine number such that the values $(m_s^{(L)}, R_s^{(L)})_{s\in\mathcal{S}}$ are an upper bound, we can transform it to a solution of this linear program by setting $x_{j,s} = 1$ and $x_{j,s'} = 0$ for $s = \sigma(j)$ and $s' \in \mathcal{S}\backslash\{s\}$. This linear program has $2|\mathcal{S}| + |\mathcal{J}_{LN}|$ conditions and $|\mathcal{S}||\mathcal{J}_{LN}|$ variables. Hence, a basic solution has at most $2|\mathcal{S}| + |\mathcal{J}_{LN}|$ non zero components. Since the factors on the right hand side are bounded by R, we can find a basic solution to this linear program in $(|\mathcal{S}||\mathcal{J}_{LN}|\log(R))^{\mathcal{O}(1)} = (m\log(R)/\delta)^{\mathcal{O}(1)}$ using the Ellipsoid-Method or other polynomial time algorithms for linear programs.

In the end of the algorithm, after the binary search part, we transform the solution x to $\mathrm{LP}_{\mathcal{J}_{LN},(m_s^{(L)},R_s^{(L)})_{s\in\mathcal{S}}}$ to an integral solution, where no job is scheduled fractionally. Further, since we rounded the resource requirement of the narrow large jobs per layer, we have to remove some of the jobs from their starting points. However, we have to be careful to remove as few as

possible since we are only allowed to add one $p_{\max}$ to the makespan of the schedule and we have a machine constraint of m.

Lemma 7.6. *Given a solution x to $\mathrm{LP}_{\mathcal{J}_{LN},(m_s^{(L)},R_s^{(L)})_{s\in\mathcal{S}}}$ that has only $2|\mathcal{S}| + |\mathcal{J}_{NL}|$ non-zero components, we have to remove at most $3|\mathcal{S}|$ jobs to guarantee an integral schedule of all the large jobs and a schedule of all the small jobs. The removed jobs have a total resource requirement of at most R for $\alpha := \varepsilon\delta/3$ and they can be found in $\mathcal{O}(|\mathcal{S}| \cdot |\mathcal{J}_{NL}|)$.*

Proof. We prove this lemma in two steps. First, we consider the jobs that are fractional scheduled in the solution x and their resource requirement. Afterward, we consider how many jobs we have to remove to meet the rounded down resource requirement and hence enable a schedule of the small jobs that does not violate the resource constraint.

Claim. A basic solution for the above linear program has at most $2|\mathcal{S}|$ fractional scheduled jobs and their total resource requirement is bounded by $2\alpha R/\varepsilon\delta$.

Proof. A job $j \in \mathcal{J}_{LN}$ is scheduled fractional if there are at least two points $s, s' \in S$ such that $x_{j,s} > 0$ and $x_{j,s'} > 0$. Each job needs at least one non zero components to be scheduled and hence $|\mathcal{J}_{LN}|$ non zero components are used by different jobs. Therefore, each fractional scheduled job needs one of the $2|\mathcal{S}|$ residual non zero components, i.e., there are at most $2|\mathcal{S}|$ fractional scheduled jobs. Hence, we have to remove at most $2|\mathcal{S}|$ jobs to schedule all jobs integral. Since each job has a resource requirement of at most αR the total resource requirement of removed jobs is bounded by $2\alpha R/\varepsilon\delta$. ◁

In addition to the at most $2|S|$ fractional scheduled jobs, we have to remove further jobs to compensate for the use of a rounded resource requirement in the linear program.

Claim. To meet the resource constraint, we have to remove at most one job per layer.

Proof. Let be $s \in S$ and $R_s^{(L)} = R/(1+1/m)^t$, i.e., we had guessed that the resource requirement of the narrow large jobs scheduled in this layer is contained in the interval $(R/(1+1/m)^{t+1}, R/(1+1/m)^t]$. To

make sure that we do not exceed the given resource, we reduce the total resource requirement of narrow large jobs scheduled in this layer to $R/(1+1/m)^{t+1}$.

If the total resource requirement of all narrow large jobs scheduled in layer s is at most $R/(1+1/m)^{t+1}$, we do not need to remove any job and hence the claim is trivially true for this layer. Otherwise, the widest job has a resource requirement of at least $(R/(1+1/m)^{t+1})/m$ since there are at most m jobs scheduled in this layer. When we remove the widest job, the total resource requirement of the residual jobs is bounded by

$$\begin{aligned}&R/(1+1/m)^t - R/m(1+1/m)^{t+1}\\ =&m(1+1/m)R/m(1+1/m)^{t+1} - R/m(1+1/m)^{t+1}\\ =&R/(1+1/m)^{t+1}.\end{aligned}$$

◁

Therefore, the total number of narrow jobs to be removed is bounded by $3|S| \in \mathcal{O}(1/(\varepsilon\delta))$. If $m \geqslant 3|\mathcal{S}| \in (1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}$ and $R \geqslant (3\alpha/\varepsilon\delta)R$, we can schedule the removed jobs at the same time at the end of the schedule without violating the resource or machine constraint. This step adds $p_{\max}$ to the makespan T'. Hence, we choose $\alpha := \varepsilon\delta/3$ in this algorithm.

□

7.2.3 Scheduling Large Jobs: The case $m \leqslant 3|\mathcal{S}|$

If $m < \mathcal{O}(1/\varepsilon\delta)$, we can guess the starting time of all the large jobs in $(1/\varepsilon\delta)^{m/\delta} = (1/\varepsilon)^{(1/\varepsilon)^{\mathcal{O}(1/\varepsilon^2)}}$ since there are at most m/δ large jobs. In this case, no large job needs to be scheduled in a later step. Since to schedule the medium and small jobs adds at most $\mathcal{O}(\varepsilon)T$ to the makespan, we end up with a schedule of height at most $(1+\mathcal{O}(\varepsilon))T'$ in this case.

7.2.4 Scheduling Small Jobs

In this section, we describe how to schedule the small jobs. To schedule these jobs, we use the techniques we used for the AFPTAS, see Chapter 6. To gain a running time that is linear in n, the first step is to round the

resource requirements of the small jobs by using the rounding technique as discussed in Lemma 6.7. As a result, we gain a set of rounded jobs $\mathcal{J}'_S$ such that $|\mathcal{J}'_S| \in \mathcal{O}(\log(m)/\varepsilon)$. This rounding can be done in $\mathcal{O}(n \log(1/\varepsilon))$ and the set of rounded jobs can be scheduled as splittable jobs instead of the jobs in $\mathcal{J}_S$ and some extra height of at most $2\varepsilon \cdot \text{area}(\mathcal{J}(I))/R + \varepsilon p(\mathcal{J}(I))/m \leqslant 3\varepsilon T$.

We will schedule these jobs in $\mathcal{J}'_S$ inside the layers using the residual free resources and machines given by the guess for the large jobs. We define $m_s^{(S)}$ as the number of free machines in layer s and analogously define $R_s^{(S)}$ as the number of free resources during the processing of this layer. Note that we have to determine the values $m_s^{(S)}$ and $R_s^{(S)}$ differently depending on the case $m > 3|\mathcal{S}|$ or $m \leqslant 3|\mathcal{S}|$. If $m \leqslant 3|\mathcal{S}|$, we can determine $m_s^{(S)}$ and $R_s^{(S)}$ by simply counting the jobs and their resource requirement per layer. On the other hand, if $m > 3|\mathcal{S}|$, we have to act slightly different. While we can determine $m_s^{(S)}$ directly by the guess for the large wide jobs and the guess for the large narrow jobs, we have to be more careful, when we calculate $R_s^{(S)}$. Let $R_{s,W}$ be the total resource requirement of the wide large jobs intersecting this layer and let $R_s^{(L)} = R/(1+1/m)^t$ be the guess for the narrow jobs. Since we rounded up the resource requirement of the narrow jobs, we define the amount of free resource in the layer s as $R_s^{(S)} := R - R_{s,W} - R_s^{(L)}/(1+1/m)$.

Solving a Configuration LP We will schedule the jobs in $\mathcal{J}'_S$ using a configuration LP that allows for each layer a certain set of configurations. This linear program has the same form as the second LP described in Section 3.3, and hence we can use the same technique to solve it. In the following, we will describe the configurations with more detail.

A configuration of jobs in $\tilde{\mathcal{J}}_S$ for a layer s is a multiset $C := \{a_j : j \mid j \in \mathcal{J}'_S\}$ such that $r(C) := \sum_{j \in \mathcal{J}_S} a_j r(j) \leqslant R_s^{(S)}$ and $m(C) := \sum_{j \in \mathcal{J}} a_j \leqslant m_s^{(S)}$; i.e., a configuration defines a multiset of jobs in $\tilde{\mathcal{J}}_S$ which can be scheduled at the same time, without violating the resource or machine constraint. We define $\mathcal{C}_s$ as the set of configurations for layer s. We introduce a new layer $\top$ with processing time $3\varepsilon T$. This is necessary since we have rounded the

resource requirements of the jobs, and therefore have an extra processing time of $3\varepsilon T$ at the end of the schedule. We define $\mathcal{S}_\top := \mathcal{S} \cup \{\top\}$ and $\mathcal{C}_s$ as the set of configurations for layer $s \in \mathcal{S}_\top$. Note that $\mathcal{C}_\top$ contains all configurations C with $r(C) \leqslant R$ and $m(C) \leqslant m$ since there is no large job scheduled in this layer.

Consider the following linear program $LP_{\mathcal{S},\mathcal{J}_S}$

$$\sum_{c \in \mathcal{C}_\top} x_{\top,C} = 3\varepsilon T \tag{7.5}$$

$$\sum_{c \in \mathcal{C}_s} x_{s,C} = \varepsilon\delta T \qquad \forall s \in S \tag{7.6}$$

$$\sum_{s \in \mathcal{S}_\top} \sum_{C \in \mathcal{C}_s} C_j x_{s,C} = p(j) \qquad \forall j \in \mathcal{J}'_S \tag{7.7}$$

$$x_{s,C} \geqslant 0 \qquad \forall s \in S_\top, C \in \mathcal{C}_s. \tag{7.8}$$

$$\tag{7.9}$$

The variable $x_{s,C}$ denotes the total processing time of the configuration $C \in \mathcal{C}_s$ in layer s. The first two conditions ensure that the total processing time of a layer is not exceeded by the total processing time of the configurations in this layer. The third condition ensures that each job is scheduled completely. Note that a rounded optimal solution can be transformed into a solution of this linear program by determining which sets of rounded small jobs are processed at the same time in each layer. We are going to find a solution to the relaxed version of this linear program, where we enlarge the right hand side of the equations (7.5) and (7.6) by the factor $(1+\varepsilon)$.

Lemma 7.7. *If there is a solution to the linear program $LP_{\mathcal{S},\mathcal{J}_S}$, we can find a solution x to the relaxed version of the linear program in at most $\mathcal{O}(\log(m)^3) \cdot (1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}$ that uses at most $\mathcal{O}(|\mathcal{S}| + |\mathcal{J}'_S|)$ non-zero components.*

Proof. This linear program has $|\mathcal{S}| + 1 + |\mathcal{J}'_S|$ constraints and at most $(\log(m)/\varepsilon)^m \cdot 1/\varepsilon\delta$ variables. By Lemma 3.9, we can solve this linear program approximately in $\mathcal{O}(|\mathcal{S}||\mathcal{J}'_S|(\ln(|\mathcal{J}'_S|) + 1/\varepsilon^2) \cdot (P(\mathcal{ABS}) + (|\mathcal{J}'_S| + |\mathcal{S}|)^{1.5356}))$, where $P(\mathcal{ABS})$ is the running time of the corresponding block-problem. This block problem asks for a given profit function for the jobs and a given layer which configuration is most profitable with respect to the contained jobs. As described in Section 6.3.2, this block problem

corresponds to the problem Unbounded Knapsack With Cardinality Constraint and hence can be solved in $\mathcal{O}(|\mathcal{J}'_S| + \log(m)/\varepsilon^4) = \mathcal{O}(\log(m)/\varepsilon^4)$. Therefore, the total running time of the algorithm is bounded by

$$\begin{aligned}&\mathcal{O}((1/\varepsilon\delta)(\log(m)/\varepsilon)(\ln(\log(m)/\varepsilon) + 1/\varepsilon^2)\\ &\cdot((\log(m)/\varepsilon^4) + ((\log(m)/\varepsilon) + (1/\varepsilon\delta))^{1.5356}))\\ \leqslant&\mathcal{O}(\log(m)^3/\varepsilon^8\delta^3).\end{aligned}$$

Note that in the solution generated by the algorithm in Lemma 3.9 has at most $(|\mathcal{S}| + 1 + |\mathcal{J}'_S|) = \mathcal{O}(\log(m)/\varepsilon^2\delta)$ non zero components. Furthermore, it is only a solution to a relaxed version of the linear program, where we extend the right hand side of equation (7.5) and (7.6) by the factor $(1+\varepsilon)$. □

Reducing the Number of Configurations In the next step, we reduce the number of non-zero components some further since scheduling the large jobs inside these configurations would add up to $\mathcal{O}(\mu T \log(m)/\varepsilon^2\delta)$ to the makespan, which is too large. First, we partition the set of rounded small jobs $\mathcal{J}'_S$ into wide and narrow jobs. We say a job $j \in \mathcal{J}'_S$ is wide, if $r(j) \geqslant \varepsilon R$ and narrow otherwise. Let $\mathcal{J}'_{SW}$ be the set of wide and $\mathcal{J}'_{SN}$ be the set of narrow jobs.

To reduce the number of non-zero components, we use the same techniques as in the AFPTAS from the previous section, i.e., we introduce windows and generalized configurations. Let x_{split} be the solution for the linear program generated by the algorithm from Lemma 7.7. Let $\mathcal{C}_{\text{LP}}$ be the set of configurations that have a non-zero component in the considered solution and let $\mathcal{C}_{\text{LP},s}$ be the set of non-zero component configurations for layer s. For a configuration $C \in \mathcal{C}_{\text{LP},s}$, we denote by $p_s(C) := x_{s,C}$ the total processing time of this configuration inside the layer s. For a configuration $C \in \mathcal{C}_{\text{LP}}$, we define $C|_{\mathcal{J}'_{SW}}$ as the configuration where we removed all the narrow jobs. Furthermore, we denote by $\mathcal{C}_{\text{LP},s,W}$ the set of all configurations in $\mathcal{C}_{\text{LP},s}$ that are reduced to their wide jobs. A *window* $w = (w_r, w_m)$ is a pair consisting of a resource requirement $r(w) = w_r$ and a number of machines $m(w) = w_m$. As for a configuration, the total time a window is processed inside a layer s is denoted as $p_s(w)$ and is called its height. At each point of time in a given window w, there

can be processed $m(w)$ jobs with summed up resource requirement $r(w)$. For windows w_1, w_2, we write $w_1 \leqslant w_2$ if and only if $r(w_1) \leqslant r(w_2)$ and $m(w_1) \leqslant m(w_2)$. A *generalized configuration* (C, w) for a layer s is a pair consisting of a configuration $C \in \mathcal{C}_{\mathrm{LP},s,W}$ and a window w such that $m(w) \leqslant m_s - m(C)$ and $r(w) \leqslant R_s - r(C)$. For a configuration $C \in \mathcal{C}_{\mathrm{LP},s,W}$, we define by $w_s(C) := (R_s - r(C), m_s - m(C))$ the *main window* for C. We define $\mathcal{W}_s$ as the set of all main windows for the configurations in $\mathcal{C}_{\mathrm{LP},s}$, and $\mathcal{W}$ be the set of all the generated widows.

Consider the following linear program: $\mathrm{LP}_W(\mathcal{S}_\top, \mathcal{J}'_S)$:

$$\sum_{s\in\mathcal{S}_\top} \sum_{C\in\mathcal{C}_{\mathrm{LP},s,W}} \sum_{\substack{w\in\mathcal{W}_s \\ w\leqslant w_s(C)}} C(j)x_{C,w,s} = p(j) \qquad \forall j \in \mathcal{J}'_{SW} \quad (7.10)$$

$$\sum_{s\in\mathcal{S}_\top} \sum_{w\in\mathcal{W}_s} y_{j,w,s} = p(j) \qquad \forall j \in \mathcal{J}'_{SN} \quad (7.11)$$

$$m(w) \sum_{\substack{C\in\mathcal{C}_{\mathrm{LP},s,W} \\ w_s(C)\geqslant w}} x_{C,w,s} \geqslant \sum_{j\in\mathcal{J}_{SN}} y_{j,w,s} \qquad \forall s \in \mathcal{S}_\top, w \in \mathcal{W}_s \quad (7.12)$$

$$r(w) \sum_{\substack{C\in\mathcal{C}_{\mathrm{LP},s,W} \\ w_s(C)\geqslant w}} x_{C,w,s} \geqslant \sum_{j\in\mathcal{J}_{SN}} r(j)y_{j,w,s} \quad \forall s \in \mathcal{S}_\top, w \in \mathcal{W}_s \quad (7.13)$$

$$x_{C,w,s} \geqslant 0 \qquad \forall s \in \mathcal{S}_\top, C \in \mathcal{C}_{\mathrm{LP},s,W}, w \in \mathcal{W} \quad (7.14)$$

$$y_{j,w,s} \geqslant 0 \qquad \forall s \in \mathcal{S}_\top, w \in \mathcal{W}_s, j \in \mathcal{J}_{SN} \quad (7.15)$$

The variable $x_{C,w,s}$ denotes the processing time of the generalized configuration (C, w) in the layer s and the value $y_{j,w,s}$ indicates which amount of job j is processed in window w in the layer s. Inequalities (7.10) and (7.11) ensure that for each job there is enough processing time reserved, while equalities (7.12) and (7.13) ensure that in each window there is enough space to schedule the contained jobs.

Given a solution (x, y) to LP_W, we define

$$p_s(x) := \sum_{C\in\mathcal{C}_{\mathrm{LP},s,W}} \sum_{\substack{w\in\mathcal{W}_s \\ w\leqslant w_s(C)}} x_{C,w,s},$$

which is the processing time of (x, y) in the layer s, and

$$p_s(w, x) := \sum_{\substack{C \in \mathcal{C}_{\mathrm{LP},s,W} \\ C(w) \geqslant w}} x_{C,w,s},$$

which is the summed up processing time of a window $w \in \mathcal{W}$ in x in layer s.

Lemma 7.8. *Given a solution x_{split} to the relaxed version of $LP_{\mathcal{S}}$, we can find a solution $(\tilde{x}, \tilde{y})$ to the linear program LP_W, which fulfills*

$$p_s(x_{\top,C}) \leqslant (1+\varepsilon)3\varepsilon T \tag{7.16}$$

$$p_s(x_{s',C}) \leqslant (1+\varepsilon)\varepsilon\delta T \qquad \forall s' \in S. \tag{7.17}$$

Proof. To generate this solution, we look at each layer s and each configuration $C \in \mathcal{C}_{\mathrm{LP},s,W}$ and sum up the processing time of each configuration $C' \in \mathcal{C}_{\mathrm{LP},s}$, which is reduced to C, i.e., $C'|_{\mathcal{J}'_{SW}} = C$. Building a generalized configuration, we combine C with its main window $w(C) \in \mathcal{W}_{\mathrm{split}}$. More precisely for each $C \in \mathcal{C}_{\mathrm{LP},s,W}$, we define

$$\tilde{x}_{(C,w(C))} := \sum_{\substack{C' \in \mathcal{C}_{\mathrm{LP},s} \\ C'|_{\mathcal{J}'_{SW}} = C}} x_{\mathrm{split}_{C',s}}.$$

Equations (7.16) and (7.17) hold for this choice for $\tilde{x}$ since the processing time of each configuration is added to exactly one generalized configuration and x_{split} fulfills equations (7.5) and (7.6). With a similar argument, one can see that inequality (7.10) holds since x_{split} fulfills equation (7.7).

On the other hand, we have to ensure that inequalities (7.11) to (7.13) hold. For this purpose, we look at each layer s and each configuration $C \in \mathcal{C}_{\mathrm{LP},s}$ and consider the reduced configuration $C|_{\mathcal{J}_{SW}}$ and its main window $w := w(C|_{\mathcal{J}_{SW}})$. For each job $j \in \mathcal{J}_N$, we add its processing time in C, which is given by $C(j)(x_{\mathrm{split}})_{C,s}$, to the window w. More precisely for each layer s, each window $w \in \mathcal{W}_s$, and each job $j \in \mathcal{J}_{SN}$, we define

$$\tilde{y}_{j,w,s} := \sum_{\substack{C \in \mathcal{C}_{\mathrm{LP},s} \\ w(C|_{\mathcal{J}'_{SW}}) = w}} C(j) x_{\mathrm{split}_{C,s}}.$$

Since the configuration C was valid and Equations (7.5) (7.6), and (7.7)

hold for x_{split}, the Equations (7.11) to (7.13) hold for $(\tilde{x}, \tilde{y})$ as a direct consequence. □

Let $(\tilde{x}, \tilde{y})$ be the solution to LP_W generated for x_{split} by Lemma 7.8. Note that the number of non-zero components in $(\tilde{x}, \tilde{y})$ is bounded by $\mathcal{O}(\log(m)/\varepsilon^2\delta)$ since in x_{split} there are at most this many non-zero components and, for each of these components, we generate at most two non-zero components in $(\tilde{x}, \tilde{y})$. Furthermore, since each configuration in $\mathcal{C}_{\text{LP},W}$ contains at most $1/\varepsilon$ jobs and there are at most $\mathcal{O}(\log(1/\varepsilon)/\varepsilon)$ different wide jobs, the number of configurations and their corresponding main windows in $(\tilde{x}, \tilde{y})$ is bounded by $\mathcal{O}(|\mathcal{S}_\top| \cdot (\log(1/\varepsilon)/\varepsilon)^{1/\varepsilon}) = \mathcal{O}((1/\varepsilon\delta) \cdot (\log(1/\varepsilon)/\varepsilon)^{1/\varepsilon}) \leqslant \mathcal{O}(1/\varepsilon^{2/\varepsilon+1}\delta)$. As a consequence, a basic solution to LP_w, where we add the Equations (7.16) and (7.17) has at most $|\mathcal{J}'_{S,W}| + |\mathcal{J}'_{S,N}| + \mathcal{O}(1/\varepsilon^{2/\varepsilon+1}\delta)$ non-zero components and can be computed in $(1/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}}$. Since each job in $\mathcal{J}'_{S,N}$ uses at least one non zero component in y and $|\mathcal{J}'_{SW}| \leqslant \mathcal{O}(1/\varepsilon^2)$, this basic solution uses at most $\mathcal{O}(1/\varepsilon^{2/\varepsilon+1}\delta)$ generalized configurations or fractionally scheduled jobs from the set $\mathcal{J}'_{SN}$.

At this point, we could proceed to find an integral solution since the number of non-zero components for configurations and fractional scheduled jabs is bounded by $\mathcal{O}_\varepsilon(1)$. However, the $\mathcal{O}(1/\varepsilon^{2/\varepsilon+1}\delta)$ generalized configurations or fractionally scheduled jobs lead to a running time of the form $\mathcal{O}(n\log(1/\varepsilon)) + (m\log(R)/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}}$. In the following steps, we will reduce the number of generalized configurations or fractionally scheduled jobs to be in $1/\delta\varepsilon^{\mathcal{O}(1)}$ and, thus, reduce the running time to $\mathcal{O}(n\log(1/\varepsilon)) + (m\log(R)/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}}$.

Lemma 7.9. *Given a solution $(\tilde{x}, \tilde{y})$ to $\text{LP}_W(\mathcal{S}_\top, \mathcal{J}'_S)$, we can find a solution $(\bar{x}, \bar{y})$ to $\text{LP}_W(\mathcal{S}_\top, \mathcal{J}'_S)$ with $p(\bar{x}) \leqslant (1+\varepsilon)p(\tilde{x})$ and has at most $\mathcal{O}(1/\varepsilon^3\delta) + |\mathcal{J}_{S,N}|$ non zero components. This solution can be found in at most $\mathcal{O}(\log(m)^3) \cdot (1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}$ operations.*

Proof. Consider a layer $s \in \mathcal{S}$ and the set of windows $\mathcal{W}_s$ occurring in this layer in the considered solution $(\tilde{x}, \tilde{y})$. At the moment there can be up to $\mathcal{O}((\log(1/\varepsilon)/\varepsilon)^{1/\varepsilon})$ different windows in $\mathcal{W}_s$. We will reduce this number to at most $\mathcal{O}(1/\varepsilon^2)$ similar as in Section 6.2.5.

We partition the windows by the size of $m(w)$ for each window $w \in \mathcal{W}_s$. Since the generalized configuration can contain at most $1/\varepsilon$ wide jobs, there are at most $1/\varepsilon + 1$ different values for $m(w)$ in $\mathcal{W}_s$. However, there is only one window that has the largest value $m(w)$ since there is no wide job scheduled next to this window. For the residual $1/\varepsilon$ sets, we stack the generalized configuration corresponding to the windows in sorted order such that the widest window is at the bottom and the most narrow one at the top. The partition and the corresponding stack of windows can be found in $\mathcal{O}((\log(1/\varepsilon)/\varepsilon)^{2/\varepsilon})$.

Let P be the height of one of these stacks. We shift down the windows by εP while the configuration part of the generalized configuration is not moved. We partition the stack into segments of the same height εP and assign the jobs contained in a window from segment i to the most narrow window in the segment below i and change window corresponding to the generalized configuration accordingly as in the proof of Lemma 6.5. The jobs from the bottom-most segment are assigned to the window (R, m), which will be processed at the end of the schedule. This reassignment of jobs to windows can be done in $\mathcal{O}((\log(1/\varepsilon)/\varepsilon)^{2/\varepsilon} \cdot |\mathcal{J}_{S,N}|)$.

Since we remove windows with total processing time at most εP from each stack, we remove windows with processing time at most $\varepsilon p(\tilde{x})$ total. After this rounding for each stack, we have at most $1/\varepsilon$ windows left. Since we have at most $1/\varepsilon$ stacks per layer and at most $\mathcal{O}(1/\varepsilon\delta)$ layer, the total number of windows is reduced to $\mathcal{O}(1/\varepsilon^3\delta)$.

Given this solution, we transform it to a basic solution using the algorithm by Ke et al. [75] in at most $\mathcal{O}((|\mathcal{S}| \cdot |\mathcal{W}| + |\mathcal{J}_S|)^{1.5356}|\mathcal{S}| \cdot |\mathcal{W}| \cdot |\mathcal{J}_S|/\varepsilon) = \mathcal{O}(\log(m)^3) \cdot (1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}$. □

Generating an Integral Schedule In the final step of the algorithm, after the binary search framework has found the correct value for T', we use this relaxed solution to schedule the original jobs in $\mathcal{J}_S$. Again, we use the same techniques as described in Sections 6.2.6 and 6.3.4.

Lemma 7.10. *Given a solution (x, y) to $\mathrm{LP}_W(\mathcal{S}_\top, \mathcal{J}'_S)$ that has at most $|\mathcal{J}'_{SN}| + \mathcal{O}(1/\varepsilon^{2/\varepsilon+1}\delta)$ non zero components, we can find an integral schedule of the jobs in $\mathcal{J}_S$ that has a makespan of at most $(1 + \mathcal{O}(\varepsilon))p(x)$ in at most*

$$\mathcal{O}((\log(m)/\varepsilon)\log(\log(m)/\varepsilon) + 1/\varepsilon^{2/\varepsilon+2}\delta + |\mathcal{J}_S|)$$

operations.

Proof. Since the generalized configurations in each layer use up to $(1+\varepsilon)\varepsilon\delta T$ processing time, we extend the layers by the factor $(1+\varepsilon)$ in the first step. Furthermore, let k_s be the number of generalized configurations inside layer s that have a non zero component in x. We extend the layer s by further μT to allow the integral schedule of the jobs in $\mathcal{J}_{SW}$.

Afterward, we schedule the wide jobs as described in Lemma 6.6 and the narrow jobs as described in Lemma 6.11. In total, we extend the schedule by at most $\mu T \cdot \mathcal{O}(1/\varepsilon^3\delta) + \mathcal{O}(\varepsilon)p(x) \leqslant \mathcal{O}(\varepsilon)p(x)$ because $\mu = \delta\varepsilon^4$, we have to extend the schedule for each configuration and fractional scheduled narrow job by at most μT, and we have to add at most $\mathcal{O}(\varepsilon)p(x)$ to schedule the narrow jobs that have to be discarded from their windows. To find this integral schedule, we need at most $\mathcal{O}(n + \log(m)/\varepsilon \log(\log(m)/\varepsilon) + 1/\varepsilon^{2/\varepsilon+1}\delta)$ operations.

□

Hence, when scheduling the small jobs integral, we extend the given schedule by a factor of at most $(1+\mathcal{O}(\varepsilon))$.

7.2.5 Scheduling Medium Jobs

In the final step, we schedule the medium jobs. In this section, we proof that it is possible to schedule them in linear time with a makespan of at most $\mathcal{O}(\varepsilon)T$. Consider the following algorithm:

- ▻ First, we sort all the jobs by height.
- ▻ Afterward, we consider the jobs with $r(j) \geqslant R/m$. We schedule the jobs in round robin manner such that the ith job is positioned on the machine with number i mod m. We define the point in time where the last of these jobs ends as $T_{R/m}$.
- ▻ Afterward, starting at $T_{R/m}$, we use the NFDH algorithm to schedule the residual jobs using $r(j)$ as the width and $p(j)$ as the height of the job. We define T_{res} as the point in time where the last job ends.

Lemma 7.11. *When scheduling the medium sized jobs $\mathcal{J}_M$ with the above-described algorithm, we need a running time of at most $\mathcal{O}(n)$ and get a makespan of at most $5\varepsilon T$.*

Proof. First, note that the constructed schedule is feasible. For the first set of jobs, we will not violate the machine constraint by the way we schedule these jobs. On the other hand, we will not violate the resource constraint since at each point in time up to $T_{R/m}$, we schedule at most m jobs with resource requirement at most R/m. For the second set of jobs, we do not violate the resource constraint since we use the NFDH algorithm, while we do not violate the machine constraint since the jobs have a resource requirement of at least R/m and hence no more jobs than m fit next to each other without violating the resource constraint.

Since we use the round robin algorithm to schedule the jobs with resource requirement of at most R/m, we know that $T_{R/m} \leqslant p(\mathcal{J}_M)/m + \max_{j \in \mathcal{J}_M} p(j)$. Furthermore, since we use the NFDH algorithm to schedule the residual jobs, we know by Lemma 2.2 that

$$T_{res} - T_{R/m} \leqslant 2\text{area}(\mathcal{J}_M)/R + \max_{j \in \mathcal{J}_M} p(j).$$

In total, the makespan is bounded by

$$\begin{aligned} & p(\mathcal{J}_M)/m + 2\text{area}(\mathcal{J}_M)/R + 2\max_{j \in \mathcal{J}_M} p(j) \\ & \leqslant \varepsilon(p(\mathcal{J})/m + 2\text{area}(\mathcal{J})/R) + 2\varepsilon T \\ & \leqslant 5\varepsilon T. \end{aligned}$$

We need a running time of at most $\mathcal{O}(n)$ to sort the jobs using Bucket-Sort since these jobs have a height of at most εT and hence at most $\varepsilon T/(\varepsilon T/n) = n$ possible sizes. The algorithms to schedule the first and second set of jobs need to iterate the sorted set of jobs only once. Hence the algorithm has a total running time of at most $\mathcal{O}(n)$. □

7.2.6 Summary of the Algorithm

In this section, we summarize the steps of the APTAS. Given an instance $I = (\mathcal{J}, m, R)$ of Single Resource Constraint Scheduling and an $\varepsilon \in (0, 1)$ the algorithm performs the following steps:

Step 1: Initialization First, we define an $\varepsilon' \in (0,1)$ such that $1/\varepsilon' \in \mathbb{N}$ and $\varepsilon' \leqslant \varepsilon/c$ for a $c \in \mathbb{R}$ large enough such that the following $(1 + \mathcal{O}(\varepsilon'))$ approximation is an $(1+\varepsilon)$ approximation. Further, we define $T := \min\{p_{\max}, \text{area}(\mathcal{J})/R, p(\mathcal{J})/m\}$ and know that $T \leqslant \text{OPT} \leqslant 4T$.

Step 2: Simplification In the next step, we simplify the given instance I, by partitioning and rounding the jobs. First, we round all processing times to multiples of $\varepsilon' T/n$, which lengthens the schedule by at most $\varepsilon' T$ and reduces the number of different possible processing times to n/ε' defining a first rounded instance $I_{r,1}$ This rounding can be done in $\mathcal{O}(n)$ and, as a result, we know that $\text{OPT}(I) \leqslant \text{OPT}(I_{r,1}) \leqslant \text{OPT}(I) + \varepsilon' T$. Next, we find the values δ and μ in $\mathcal{O}(n + 1/\varepsilon)$ as described in Lemma 7.4 and partition the set of jobs $\mathcal{J}(I_{r,1})$ into large, medium and small jobs in $\mathcal{O}(n)$. The large jobs are rounded to at most $\mathcal{O}(1/\varepsilon^4)$ different processing times using Lemma 3.2. We call this rounded instance $I_{r,2}$ and it holds that $\text{OPT}(I) \leqslant \text{OPT}(I_{r,2}) \leqslant (1+2\varepsilon)(1+\varepsilon)\text{OPT}(I)$. In the final simplification step, we round the resource requirements of the small jobs to at most $\log(m)/\varepsilon'$ sizes, using the rounding in Lemma 6.7 generating a further rounded instance $I_{r,3}$. This instance can be found in $\mathcal{O}(n \log(1/\varepsilon))$ and it holds that $\text{OPT}(I) \leqslant \text{OPT}(I_{r,3}) \leqslant ((1+2\varepsilon)(1+\varepsilon)+3\varepsilon)\text{OPT}(I) = (1 + 6\varepsilon + 2\varepsilon^2)\text{OPT}(I)$. Note that the instance $I_{r,3}$ contains at most $\mathcal{O}(m/\delta)$ different jobs. In the last simplification step, we partition the set of large jobs into narrow and wide ones as well as the small jobs into these both categories.

Step 3: Binary Search Framework For the rounded instance $I_{r,3}$, we will find the value $T' = (1 + i\varepsilon')T$, for $i \in [0, 4/\varepsilon + 8] \cap \mathbb{N}$ such that $T' - \varepsilon' T \leqslant \text{OPT}(I_{r,3}) \leqslant T'$ and give a schedule with makespan at most $(1 + \mathcal{O}(\varepsilon'))T' + p_{\max} = (1 + \mathcal{O}(\varepsilon'))\text{OPT}(I) + p_{\max}$. We use dual approximation and try the values T' in binary search fashion in the next step.

Step 4: Guessing Step Given a value $T' := i\varepsilon' T$ we determine the corresponding set $\mathcal{S}$. Afterward, we try each possibility to schedule the large wide jobs $\mathcal{J}_{LW}$ and each possibility for the vector $(m_s^{(L)}, R_s^{(L)})_{s \in \mathcal{S}}$. There are at most $(\log(R)m/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}}$ possibilities for these guesses. For a given guess, we try to solve the linear program $\text{LP}_{\mathcal{J}_{LN}, (m_s^{(L)}, R_s^{(L)})_{s \in \mathcal{S}}}$, which can be done in $(\log(R)m/\delta)^{\mathcal{O}(1)}$. If the linear program is not solvable, we discard

the current guess. Otherwise, we try to solve the linear program $LP_{\mathcal{S},\mathcal{J}_S}$ to place the small jobs. By Lemma 7.7 a solution to the relaxed version can be found in $\mathcal{O}(\log(m)/\varepsilon^8\delta^3)$ if $LP_{\mathcal{S},\mathcal{J}_S}$ has a solution. If we find such a solution, we save the guess and both LP-solutions and try the next smaller value for T' in binary search fashion. Otherwise, we try the next guess. If all the guesses deny the solvability of one of both linear programs, we try the next larger value for T' in binary search fashion.

Step 5: Scheduling the Original Jobs When the binary search procedure stops, we have a guess how to schedule the large jobs and both LP solutions. By the Lemmas 7.6 and 7.10, we can transform these solutions to schedules for the jobs in $\mathcal{J}(I_{r,2})$ with an additive loss of at most $\mathcal{O}(\varepsilon')T' + p_{\max}$ in the approximation ratio. This transformation needs at most

$$\begin{aligned}&\mathcal{O}((1/\varepsilon)^{\mathcal{O}(1/\varepsilon)} \cdot |\mathcal{J}_{NL}| + \log(m)^3(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}\\ &+ (\log(m)/\varepsilon)\log(\log(m)/\varepsilon) + 1/\varepsilon^{2/\varepsilon+2}\delta + |\mathcal{J}_S|))\\ &= \mathcal{O}(n + m(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)})\end{aligned}$$

operations. To this schedule, we add the medium sized jobs $\mathcal{J}_M$ to the schedule using the algorithm described in Section 7.2.5. By Lemma 7.11 this adds at most $\mathcal{O}(\varepsilon')\mathrm{OPT}(I)$ to the makespan and needs at most $\mathcal{O}(n)$ operations. This schedule for the instance $I_{r,2}$ is then transformed by replacing the jobs in $\mathcal{J}(I_{r,2})$ by the original jobs in $\mathcal{J}(I)$. Note that this is possible since the processing time of these jobs is smaller than the processing time of the jobs in $\mathcal{J}(I)$. Finally, we can transform the schedule to one where each job is started at an integral starting point by iterating the jobs one by one and reduce the starting time of this jobs to the next smaller integral if it is not integral already. After this step, the schedule still fulfills all the constraints since now all the jobs end at integral points as well.

In this section, we have presented an algorithm with approximation guarantee $(1+\varepsilon)\mathrm{OPT} + p_{\max}$ and a running time that can be bounded by $\mathcal{O}(n\log(1/\varepsilon) + (\log(R)m/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}})$. Hence, in this section, we have proven the Theorem 7.1. Note that in the case where $m \leqslant f(1/\varepsilon)$, we can guess the starting times of all the large jobs and therefore have an $(1+\varepsilon)$-approximation with running time $\mathcal{O}(n\log(1/\varepsilon) + (1/\varepsilon)^{f(1/\varepsilon)\cdot 1/\varepsilon^{\mathcal{O}(1/\varepsilon)}})$

which corresponds to an EPTAS.

Note that the only step, where ε appears in the exponent of the size of the input is the step, where we guess the vector $(m_s^{(L)}, R_s^{(L)})_{s\in\mathcal{S}}$. In Section 7.4, we will prove that we can reduce the running time by making a rougher estimate on the values in $(m_s^{(L)}, R_s^{(L)})_{s\in\mathcal{S}}$. However, a consequence of this weaker estimate is that we have to remove more jobs and shift them to the top. More precisely, we have to shift up $\mathcal{O}(\varepsilon)m$ jobs. This larger number of shifted jobs is a problem in the following algorithm where we exploit the feature that the number of jobs that have to be shifted is in $\mathcal{O}_\varepsilon(1)$. Therefore, we were not able to guarantee an efficient running time in the $(3/2+\varepsilon)$ approximation.

7.3 A $(3/2+\varepsilon)$-Approximation

In this section, we design an algorithm with approximation ratio $(3/2+\varepsilon)$. The algorithm uses the techniques described for the APTAS. However, we have to be more careful which large jobs can be scheduled fractional and shifted to the end of the schedule and which cannot.

We aim to find a schedule with makespan $(3/2+\mathcal{O}(\varepsilon))T'$, where T' is the assumed optimal makespan given by a binary search framework. If we discard a job larger than $T'/2+\mathcal{O}(\varepsilon)T$ and schedule it after T', we exceed the aspired approximation ratio of $(3/2+\mathcal{O}(\varepsilon))T$. We call this set of critical jobs huge jobs, i.e., $\mathcal{J}_H := \{j \in \mathcal{J} | p(j) > T'/2\}$. As a consequence, we redefine the set of large jobs as $\mathcal{J}_L := \{j \in \mathcal{J} | \delta T \leqslant p(j) \leqslant T'/2\}$ respectively.

Notice that the processing of all huge jobs has to intersect the time $T'/2$ in each schedule with makespan at most T'. Therefore, each machine can contain at most one of these jobs. If we could guess the starting positions of these huge jobs, the discarded jobs in the linear program $\mathrm{LP}_{\mathcal{J}_{LN},(m_s^{(L)},R_s^{(L)})_{s\in\mathcal{S}}}$ would have a height of at most OPT/2 and could be placed on top of the schedule. Sadly this guessing step is not possible in polynomial time since there are up to m of these jobs and iterating all combinations of their starting position needs $\Omega((1/\varepsilon\delta)^m)$ operations. Our idea is to let almost all the huge jobs end at a common point in time, e.g. T', and thus avoid the

guessing step. To solve the violation of the resource or machine condition, we shift up all the jobs which start after $\lceil T'/2 \rceil_{\varepsilon\delta T}$ by $\lceil T'/2 \rceil_{\varepsilon\delta T}$ such that they now start after T', where we denote by $\lceil T'/2 \rceil_{\varepsilon\delta T}$ the integer multiple of $\varepsilon\delta T$ that is the first which has a size of at least $T'/2$.

While this shift fixes the start positions of the huge jobs, the large jobs are again placed with the linear program from Section 7.2.2. Since there are some large jobs which could be scheduled fractional by the linear program, we need to find a gap in the shifted schedule where we can place them. In the following, we will consider optimal schedules and the possibilities to rearrange the jobs. Depending on this arrangement, we can find a gap for the fractional scheduled large jobs of height $\lceil T'/2 \rceil_{\varepsilon\delta T}$.

Let us assume that we have to schedule $k \in \mathcal{O}_\varepsilon(1) \leqslant m/4$ jobs with total resource requirement at most $k\alpha R \leqslant R/4$. We consider an optimal schedule, after applying the simplification steps, i.e., we consider the rounded instance $I_{r,3}$ and the corresponding transformed optimal schedule, where each large job starts at a multiple of $\varepsilon\delta T$ and each huge job starts at a multiple of $\varepsilon^2 T$. Furthermore, we will assume that there are more than $4k = \mathcal{O}_\varepsilon(1)$ huge jobs. Otherwise, we can guess their starting positions in $\mathcal{O}((1/\varepsilon\delta)^{4k})$ and place the fractional scheduled large jobs on top of the schedule.

In the following, we will prove that by extending it by $\lceil T'/2 \rceil_{\varepsilon\delta T}$, we can transform the rounded optimal packing $\mathrm{OPT}_{\mathrm{rounded}}$ such that all the huge jobs, except for $\mathcal{O}(k)$ of them, start at a common point in time and we can place k further narrow large jobs without violating the machine or the resource constraint.

Lemma 7.12. *Given a rounded optimal schedule* $\mathrm{OPT}_{\mathrm{rounded}}$ *with makespan at most* T'*, we can find a transformed schedule* $\mathrm{OPT}_{\mathrm{shift}}$ *with makespan at most* $\mathrm{OPT}_{\mathrm{rounded}} + \lceil T'/2 \rceil_{\varepsilon\delta T}$*, where all huge jobs except of at most* $2k$ *end at the same point in time* γ*, and we can schedule further* k *jobs with processing time at most* $\lceil T'/2 \rceil_{\varepsilon\delta T}$ *and resource requirement at most* αR*. Further, there is an injective function which maps each layer* s *in* $\mathrm{OPT}_{\mathrm{rounded}}$ *with* $m_{s,S}$ *machines and* $R_{s,S}$ *resources reserved for the small jobs to a layer in* $\mathrm{OPT}_{\mathrm{shift}}$ *where we reserve at least as many machines and resources for the small jobs.*

Proof. We will prove this lemma by a careful analysis of the structure of the schedule $\mathrm{OPT}_{\mathrm{rounded}}$. First, however, we introduce some notations. Let

$s \in \mathcal{S}$, with $s > T'/2$ be any starting point of large jobs. We say a job $j \in \mathcal{J}$ intersects s or is intersected by s if $\sigma(j) < s < \sigma(j) + p(j)$. We will differentiate between the jobs that start before $T'/2$ and those that start at or after $T'/2$. We will add the attribute $_{\text{pre}}$ to sets of jobs that contain only jobs starting before $T'/2$, and the attribute $_{\text{post}}$ to those that contain only jobs that start at or after $T'/2$. Furthermore, we will identify the sets of jobs that intersect certain points of time. We will add the attribute $_{s,\uparrow}$ to denote a set of jobs that is processed up to s, i.e., we denote by $\mathcal{J}_{s,\uparrow,\text{pre}} := \{j \in \mathcal{J} | p(j) \geqslant \delta T, \sigma(j) < T'/2, \sigma(j) + p(j) \geqslant s\}$ the set of large and huge jobs starting before $T'/2$ and ending at or after s. On the other hand, if we are only interested in the jobs that intersect the time s, we add the attribute $_{s,\updownarrow}$ to the set and mean $\mathcal{J}_{s,\updownarrow,\text{pre}} := \{j \in \mathcal{J} | p(j) \geqslant \delta T, \sigma(j) < T'/2, \sigma(j) + p(j) > s\}$. Finally, we will indicate if the set contains only huge or only large jobs, by adding the attribute $_H$ or $_L$.

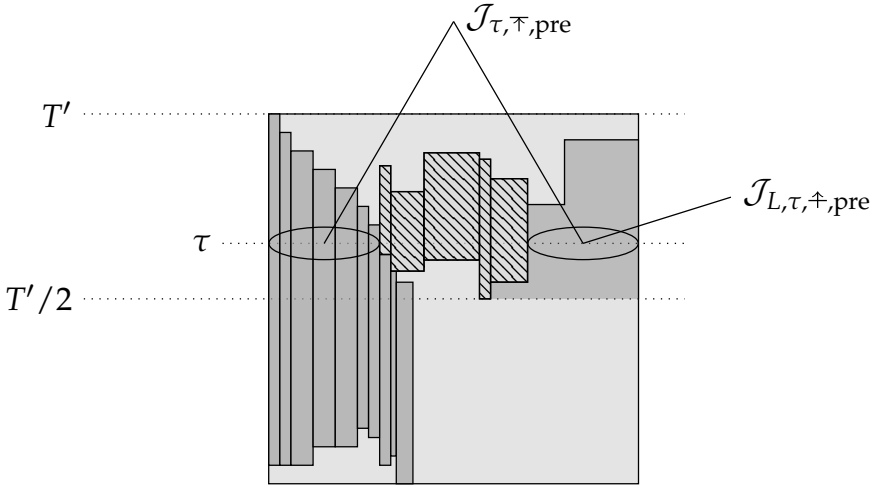


Figure 7.1. An optimal packing. The hatched rectangles are the jobs that start after $T'/2$ and intersect τ, the dark gray area corresponds to large jobs, which start before $T'/2$ and end after τ and the dark gray rectangles on the left are huge jobs.

Let $\tau \in \{s | s \in S, T'/2 \leqslant s \leqslant T'\}$ be the smallest value such that there are at most $m - k$ jobs (huge or large) that start before $T'/2$ and intersect τ, i.e., end after τ. We partition the set of large jobs intersected by τ into two sets. Let $\mathcal{J}_{L,\tau,\updownarrow,\text{pre}} := \{j \in \mathcal{J}_L | \sigma(j) < T'/2, \sigma(j) + p(j) > \tau\}$ be the set of large jobs which start before $T'/2$ and end at or after τ. Further let

$\mathcal{J}_{L,\tau,\Uparrow,\text{post}} := \{j \in \mathcal{J}_L | T'/2 \leqslant \sigma(j) < \tau, \sigma(j) + p(j) > \tau\}$ be the set of large jobs, which are started at or after $T/2$ but before τ and end after τ, see Figure 7.1.

Note that by the choice of τ in each point between τ and $T'/2$ in the schedule there are more than $m-k$ machines used by $\mathcal{J}_{\tau,\Uparrow,\text{pre}}$. As a result there are at most $k-1$ machines used by jobs starting after $T'/2$ at each point between $T'/2$ and τ, implying $|\mathcal{J}_{L,\tau,\Uparrow,\text{post}}| < k$.

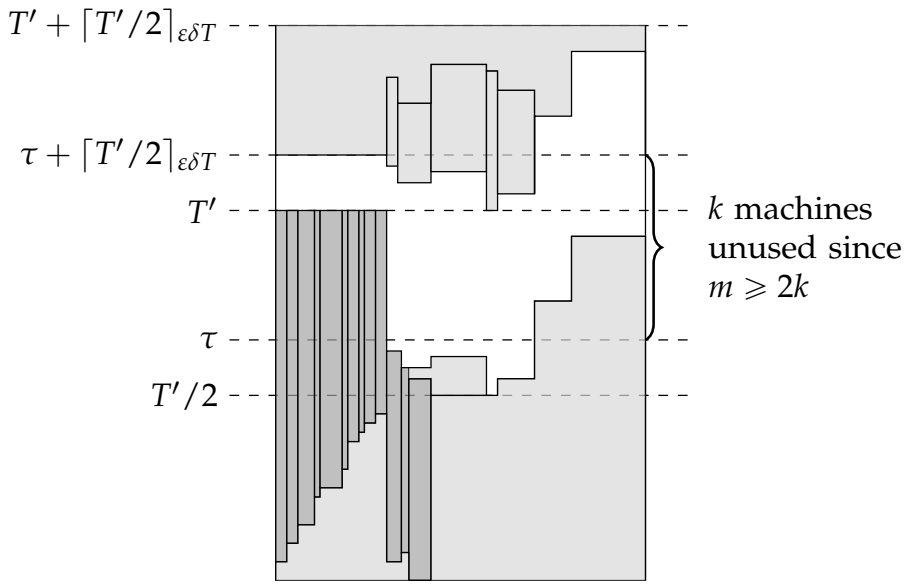


Figure 7.2. The shifted schedule

We now construct a shifted schedule. Starting times in this schedule will be denoted by σ'. We shift each job $j \in \mathcal{J}$ with $\sigma(j) \geqslant T'/2$ and $\sigma(j) + p_j \geqslant \tau$ exactly $\lceil T'/2 \rceil_{\varepsilon\delta T}$ upwards, i.e., we define $\sigma'(j) := \sigma(j) + \lceil T'/2 \rceil_{\varepsilon\delta T}$ for these jobs. Furthermore, each huge job $j \in \mathcal{J}_H$ intersecting τ is shifted upwards such that it ends at T', i.e., we define $\sigma'(j) := T' - p_j$ for these jobs j, see Figure 7.2. Note that there are at most k huge jobs ending strictly before τ. If the total number of huge jobs ending before or at τ is larger than k, we choose arbitrarily from the set of jobs ending at τ and shift them until there are exactly k huge jobs ending before or at τ.

Claim. After this shift there are at least k machines at each point between τ and $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ that are not used by any other job.

Up to T', there are k free machines, because there is no new job starting between τ and T' since we shifted all of them up such that they start after $\lceil T'/2 \rceil_{\varepsilon\delta T}$. On the other hand, only jobs from the set $\mathcal{J}_{L,\tau,\uparrow,\text{post}}$ are processed between T' and $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$. Since $|\mathcal{J}_{L,\tau,\uparrow,\text{post}}| < k$ and $m - k \geqslant k$ this leaves k free machines which proves the claim.

We would like to place the gap at τ since there are enough free machines. Sadly it can happen that at a point between τ and $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there is not enough free resource for the gap. In the following, we carefully analyze where we can place the k jobs, dependent on the structure of the optimal schedule $\text{OPT}_{\text{rounded}}$.

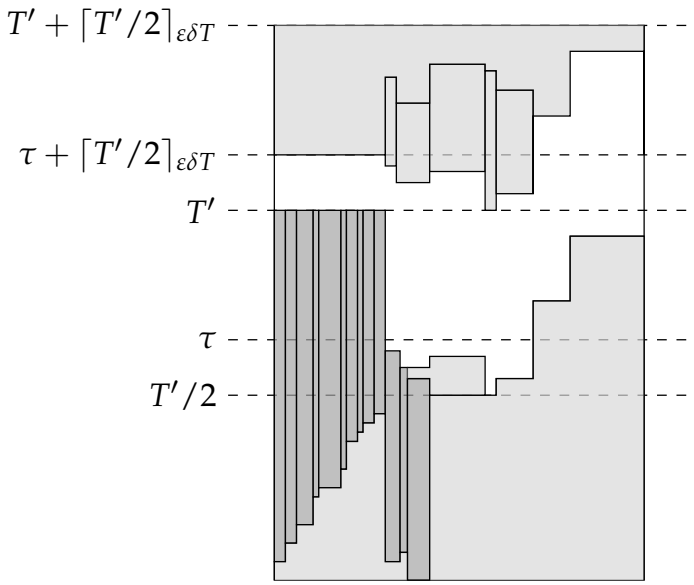


Figure 7.3. Case 1: Between τ and T' there are k free machines and $k\alpha R$ free resources.

Case 1: $r(\mathcal{J}_{\tau,\mp,\text{pre}}) \leqslant R - k\alpha R$ In this case there are at least $k\alpha R$ free resources at each point in the shifted schedule between τ and T' since there are no jobs starting between these points of time.

To place the k fractional scheduled jobs, we have to generate a gap of height $\lceil T'/2 \rceil_{\varepsilon\delta T}$. In this gap there have to be k unused machines and $k\alpha R$ unused resources. For the time between τ and T', we have this guarantee, while for the time between T' and $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$, we have k free machines,

but might have less than $k\alpha R$ free resources. The only jobs overlapping in this time window are the jobs from the set $\mathcal{J}_{L,\tau,\Uparrow,\text{post}}$, see Figure 7.3. If these jobs have a small resource requirement, we have found our gap, see Case 1.1. and, otherwise, we have to look more careful at the schedule.

Case 1.1: $r(\mathcal{J}_{L,\tau,\Uparrow,\text{post}}) \leqslant R - k\alpha R$. In this case, there is a gap between τ and $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$, see Figure 7.4. In this shifted optimal schedule there are at most k huge jobs ending before τ. In the algorithm, we will guess τ dependent on a given fractional solution for the large jobs and guess these k huge jobs and their start points in $\mathcal{O}(m^k S^k)$, which is polynomial in the input size, see Section 7.3.1 for an overview.

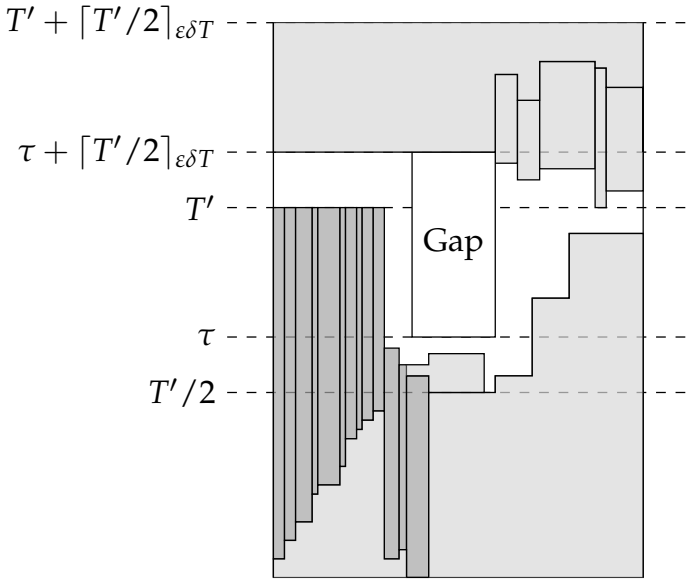


Figure 7.4. The shifted schedule and the gap between τ and $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ in Case 1.1

Case 1.2: $r(\mathcal{J}_{L,\tau,\Uparrow,\text{post}}) \geqslant R - k\alpha R$. In this case, there is a point $t \in [\tau, T']$ such that after this point there are less than $k\alpha R$ free resources. Therefore, we need another position to place the fractionally scheduled jobs. To generate a gap, we shift down the widest job in $\mathcal{J}_{L,\tau,\Uparrow,\text{post}}$ back to its start position.

Claim. When choosing $\alpha \leqslant \frac{1}{k(k+1)}$, there exist at least one large job $i \in \mathcal{J}_{L,\tau,\Uparrow,\text{post}}$ with a resource requirement of $r(j) \geqslant k\alpha R$.

Proof. In $\mathcal{J}_{L,\tau,\dot{+},\text{post}}$ there are at most $k-1$ large jobs using more than $R - k\alpha R$ resource total. Since $\alpha \leqslant \frac{1}{k(k+1)}$, it holds that

$$\frac{R - k\alpha R}{k} \geqslant \frac{R(1 - k\frac{1}{k(k+1)})}{k} = \frac{R(\frac{k+1}{k+1} - \frac{1}{k+1})}{k} = Rk\frac{1}{k(k+1)} \geqslant k\alpha R.$$

Hence, by the pigeon principle, one of the jobs must have a resource requirement of at least $k\alpha R$. □

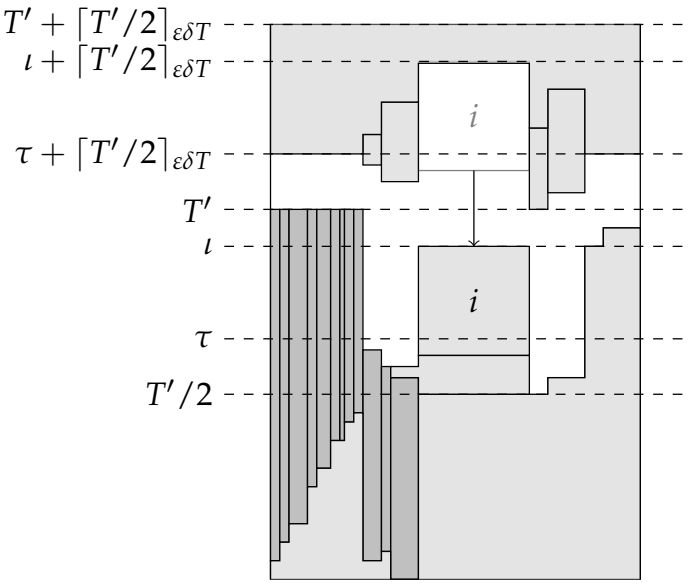


Figure 7.5. Case 1.2: The job with resource requirement at least $k\alpha R$ is shifted down. At each point between ι and $\iota + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there are at least $k\alpha R$ unused resources.

We shift this job i down such that it starts at its primarily starting position. Let $\iota := \sigma(i) + p(i)$ be its end position.

Claim. As a result of this shift, there are at least $k\alpha R$ free resources at each point between ι and $\iota + \lceil T'/2 \rceil_{\varepsilon\delta T}$.

Proof. At each point between ι and T' there were $k\alpha R$ unused resources before. Each job which starts between T' and $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ is an element of $\mathcal{J}_{L,\tau,\dot{+},\text{post}}$ and was therefore scheduled with the wide job at the same time. So between T' and $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there is at least $k\alpha R$ resource unused.

From $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ to $\iota + \lceil T'/2 \rceil_{\varepsilon\delta T}$ the wide job i was scheduled, hence there is at least $k\alpha R$ free resource. □

We now have to differentiate if there are at least k machines unused between $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ and $\iota + \lceil T'/2 \rceil_{\varepsilon\delta T}$, see Figure 7.5. Let $\rho \in \{s | \tau \leqslant s \leqslant T, s \in S\}$ be the first point in the schedule where at most k jobs from $\mathcal{J}_{\tau,\mp,\text{pre}}$ are scheduled in the given optimal schedule (not the shifted one), i.e., ρ is the first point in time where $|\mathcal{J}_{\rho,\mp,\text{pre}}| \leqslant k$. Note that as a consequence $|\mathcal{J}_{\rho,\mp,\text{pre}}| \geqslant k$ since otherwise there would have been a point in time before ρ, where at most k machines are used by jobs starting before $T'/2$. We know that between T' and $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there always will be k machines unused since before the first shift they were blocked by jobs in $\mathcal{J}_{\tau,\mp,\text{pre}}$.

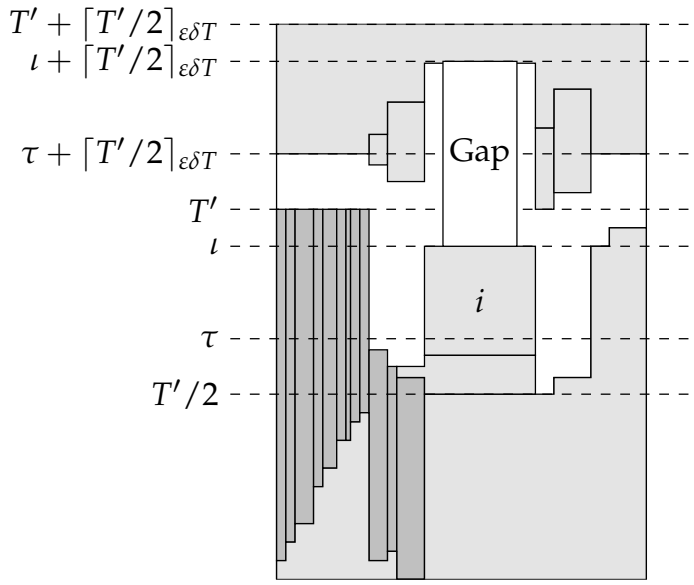


Figure 7.6. The shifted schedule and the position of the gap in Case 1.2.1

Case 1.2.1: $\rho \geqslant \iota$. In this case, at each point between ι and $\iota + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there are k machines unused. Between ι and T' there are k free machines by the choice of τ and between T' and $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there are k free machines by the choice of ρ. Therefore, there is a gap between ι and $\iota + \lceil T'/2 \rceil_{\varepsilon\delta T}$, see Figure 7.6.

Case 1.2.2: $\rho < \iota$. Let $\mathcal{J}_{H,\rho} := \{j \in \mathcal{J}_H | s_j + p_j > \rho\}$ be the set of huge jobs, which are still scheduled after ρ. It holds that $|\mathcal{J}_{H,\rho}| \leqslant k$. As a consequence, it is possible to guess their starting positions in polynomial time and, hence, we schedule each job in $\mathcal{J}_{H,\rho}$ as in the original simplified schedule $\text{OPT}_{\text{rounded}}$. The other huge jobs, which end between τ and ρ, are scheduled such that they end at ρ, i.e., we define $\sigma'(j) := \rho - p(j)$ for each of these huge jobs j. Next, we shift the all the jobs j with starting time $\sigma'(j) \geqslant \rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$ downwards such that they start as they had started before the first shift. As a result between T' and $T' + \lceil T'/2 \rceil_{\varepsilon\delta T}$, there are just jobs left which overlap the time from $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ to $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$, see Figure 7.7.

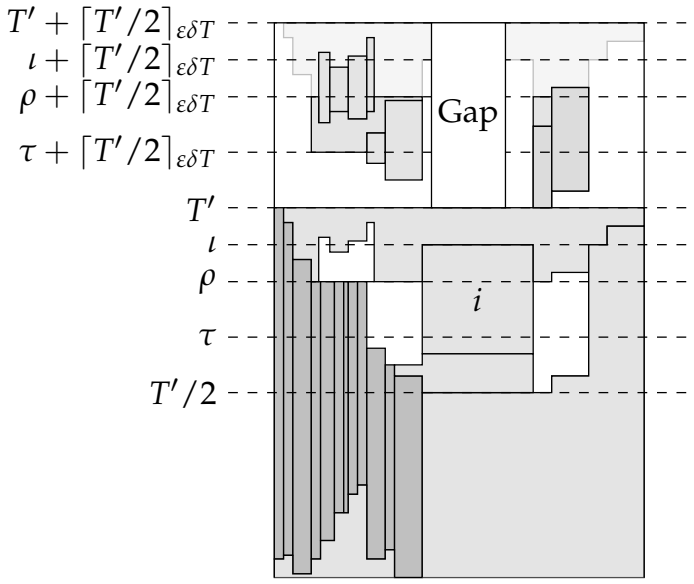


Figure 7.7. The shifted schedule and the position of the gap in Case 1.2.2

By the choice of ρ and τ at each point between $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ and $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there are at most $m - k$ jobs which use at most $R - k\alpha R$ resource since the job i was scheduled there before. Since each job between T' and $T' + \lceil T'/2 \rceil_{\varepsilon\delta T}$ overlaps this area there are at least k free machines and $k\alpha R$ free resources in this area. Hence, we position the gap at T'.

In the algorithm, we will guess τ and ρ dependent on a given fractional solution for the large jobs and guess the at most k jobs ending before τ

and the k jobs ending after ρ in $\mathcal{O}(m^{2k-1})$. For each of these jobs, we have to guess its starting time out of at most $S/2$ possibilities.

Case 2: $r(\mathcal{J}_{\tau,\mp,\text{pre}}) > R - k\alpha R$. In this case, the gap has to start strictly after τ since at τ there is not enough free resource. Let $\mathcal{J}_{L,T'/2}$ be the set of large jobs intersecting the point in time $T'/2$. Remember that $\mathcal{J}_{\tau,\mp,\text{pre}}$ contains huge and large jobs. Since $r(\mathcal{J}_{\tau,\mp,\text{pre}}) > R - k\alpha R$, at least one of these sets of jobs (huge or large) has to contribute a large resource requirement to $r(\mathcal{J}_{\tau,\mp,\text{pre}})$. In the following, we will find the gap, depending on which of both sets contributes a suitable large resource requirement.

Case 2.1: $r(\mathcal{J}_{L,T'/2}) \geqslant 2k\alpha R$. Let $\tau' \in \{s \in S | \tau \leqslant s \leqslant T'\}$ be the first point in time where $r(\mathcal{J}_{L,T'/2}) - r(\mathcal{J}_{L,\tau',\mp,\text{pre}}) \geqslant k\alpha R$. Note that $\tau \leqslant \tau'$ since, otherwise, there would be $k\alpha R$ free resources at τ.

Claim. By this choice at each point between τ' and $\tau' + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there are at least $k\alpha R$ free resources.

Between τ' and T' there are $k\alpha R$ free resources since jobs form $\mathcal{J}_{L,T'/2}$ with a resource requirement of at least $k\alpha R$ end before τ'. On the other hand, before the shift there was at least $k\alpha R$ resource blocked by jobs from $\mathcal{J}_{L,T'/2}$ between $T'/2$ and τ' and hence after the shift there is at least $k\alpha R$ free resource at any time between T' and $\tau' + \lceil T'/2 \rceil_{\varepsilon\delta T}$.

Moreover, as in Case 1.2, let $\rho \in \{s | \tau \leqslant s \leqslant T', s \in S\}$ be the first point in the schedule where $|\mathcal{J}_{\rho,\mp,\text{pre}}| \leqslant k$, i.e., where at most k jobs are scheduled that start before $T'/2$.

Claim. By this choice at each point between τ and $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there are at least k unused machines.

From τ to T' there are k unused machines, by the choice of τ. On the other hand, at each point in time between T' and $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there were k machines blocked by jobs that started before $T'/2$ and these machines are now unused.

Similar as in Cases 1.2.1 and 1.2.2, we will find the gap dependent of the relation between τ' and ρ.

Case 2.1.1: $\rho \geqslant \tau'$. In this case between τ' and $\tau' + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there are at least k unused machines. Therefore, we have a gap between these two points, which is large enough, see Figure 7.8. In the algorithm, we have to

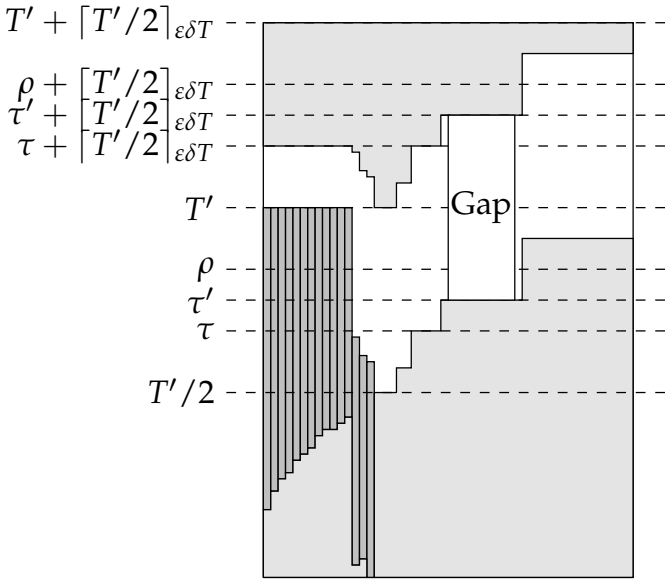


Figure 7.8. The shifted schedule and the position of the gap in Case 2.1.1

guess the k huge jobs, which end before τ and their start point, as well as the points τ, τ' and ρ.

Case 2.1.2: $\rho < \tau'$. In this case, we act like in Case 1.2.2 and shift all huge jobs, but the at most k jobs ending after ρ, downwards such that they end at ρ, see Figure 7.9. Furthermore, we shift all jobs starting after $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$ back downwards such that they again start at their primary start position. Now after T' there are just jobs having their start or end position between $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ and $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$. At each point between these two points there are at least k unused machines and $k\alpha R$ unused resource with the same arguments as in Case 1.2.2. Hence, we have a gap with the right properties between T' and $T' + \lceil T'/2 \rceil_{\varepsilon\delta T}$.

Case 2.2: $r(\mathcal{J}_{L,T'/2}) < 2k\alpha R$. Since we have $r(\mathcal{J}_{\tau,\overline{\tau},\text{pre}}) > R - k\alpha R$ (by Case 2.) and it holds that $(\mathcal{J}_H \cap \mathcal{J}_{\tau,\overline{\tau},\text{pre}}) \cup (\mathcal{J}_{L,T'/2} \cap \mathcal{J}_{\tau,\overline{\tau},\text{pre}}) = \mathcal{J}_{\tau,\overline{\tau},\text{pre}}$, we get that $r(\mathcal{J}_H \cap \mathcal{J}_{\tau,\overline{\tau},\text{pre}}) \geqslant R - 3k\alpha R$.

Similar as before, let $\rho \in \{s | \tau \leqslant s \leqslant T, s \in S\}$ be the first point in the schedule where less than k jobs are scheduled that start before $T'/2$. By the same argument as in Case 2.1, we know that at every point between τ and $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$ there are at least k unused machines in the shifted

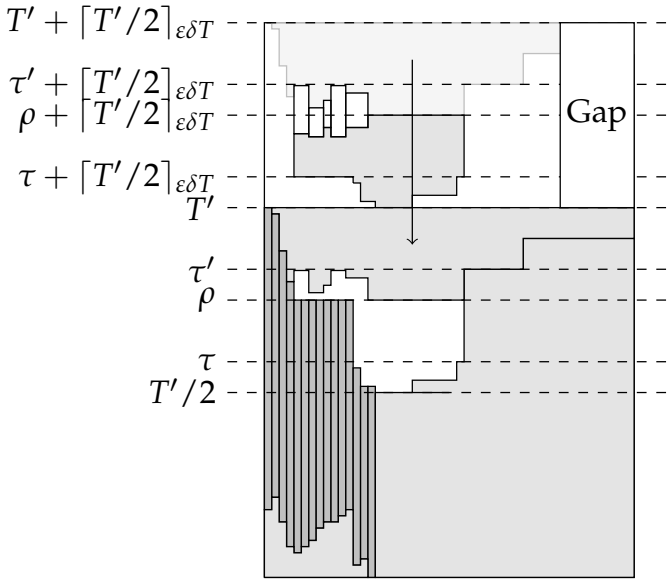


Figure 7.9. The shifted schedule and the position of the gap in Case 2.1.2

schedule.

Case 2.2.1: $r(\mathcal{J}_{\rho,\mathrm{T},\mathrm{pre}}) \geqslant k\alpha R$. In this case, we can construct a schedule in the same way as in case 1.2.2 or 2.1.2 by shifting down the jobs that start after $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$ and positioning the gap at T', see Figure 7.10. This is possible because the jobs that are scheduled between $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ and $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$ can use at most $R - k\alpha R$ resources in this case since $r(\mathcal{J}_{\rho,\mathrm{T},\mathrm{pre}}) \geqslant k\alpha R$ and hence at least $k\alpha R$ resources are blocked by the jobs in $\mathcal{J}_{\rho,\mathrm{T},\mathrm{pre}}$.

Case 2.2.2: $r(\mathcal{J}_{\rho,\mathrm{T},\mathrm{pre}}) < k\alpha R$. Let $\rho' \in \{i\delta^2 | \tau/\delta^2 \leqslant i \leqslant \rho/\delta^2, i \in \mathbb{N}\}$ be the smallest value, where $r(\mathcal{J}_{\rho',\mathrm{T},\mathrm{pre}}) \leqslant k\alpha R$. Remember, we had $r(\mathcal{J}_H \cap \mathcal{J}_{\tau,\mathrm{T},\mathrm{pre}}) \geqslant R - 3k\alpha R$ so huge jobs with summed resource requirement of at least $R - 4k\alpha R$ are finished till ρ'. We partition the huge jobs that finish between τ and ρ by their processing time. Since each job has a processing time of at least $\lceil T'/2 \rceil_{\varepsilon\delta T}$, we get at most $\mathcal{O}(1/\varepsilon\delta) \leqslant |\mathcal{S}|/2$ sets. As seen in Section 7.2.2, we have to discard at most $k \leqslant 3|\mathcal{S}|$ large jobs, which have to be placed later on.

Claim 7.13. When choosing $\alpha \leqslant 2/(k(k+8))$ and assuming $k = 3|\mathcal{S}|$, there is a set in the partition, which uses at least $3k\alpha R$ resource total.

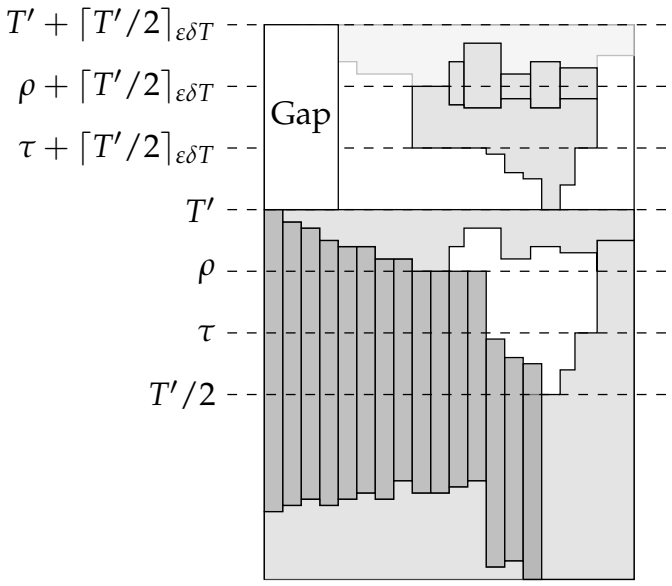


Figure 7.10. The shifted schedule and the position of the gap in Case 2.2.1

Choosing α as claimed, it holds that

$$\frac{R - 4k\alpha R}{|\mathcal{S}|/2} \geqslant \frac{(1 - 4k\frac{2}{k(k+8)})R}{k/6} = \frac{6(\frac{k+8}{k+8} - \frac{8}{k+8})}{k} R = 3k\frac{2}{k(k+8)} R = 3k\alpha R.$$

Therefore, by the pigeon principle, there must be one set in the partition, which has summed resource requirement of at least $3k\alpha R$. We sort the jobs in this partition by non increasing order of resource requirement. We greedily take jobs from this set, till they have a summed resource requirement of at least $k\alpha R$ and schedule them such that they end before ρ'. If there was a job with more than $k\alpha R$ resource requirement, it had to be finished before ρ' since the resource requirement of huge jobs finishing after ρ' is smaller than $k\alpha R$. In the other case, we greedily choose jobs with summed resource requirement between $k\alpha R$ and $2k\alpha R$.

Since the considered set has a summed resource requirement of at least $3k\alpha R$, jobs of this set with summed resource requirement at least $2k\alpha R$ end before ρ'. Therefore, we do not violate any constraint by shifting down these jobs such that they end at ρ', see Figure 7.11.

Note that since we only use the free area (machines and resources) to

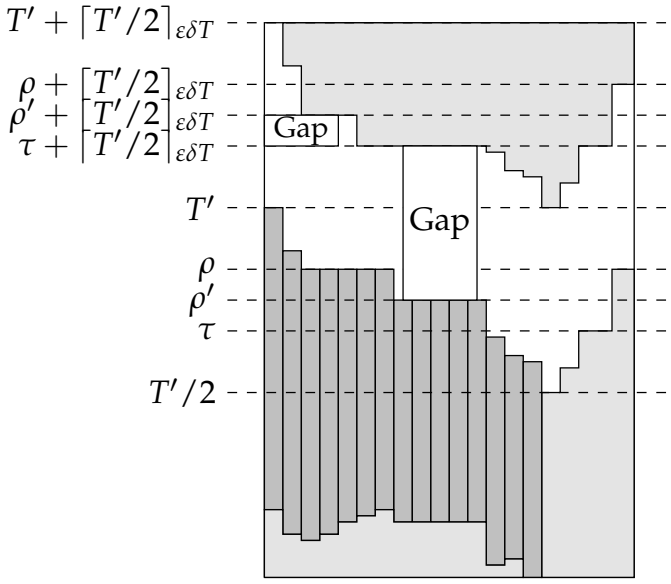


Figure 7.11. The shifted schedule and the position of the gap in Case 2.2.2. Note that in this figure the gap is not displayed continuously. However, by swapping the used resource, we can make it continuous. We only need the fact that at each point in time there are enough free resources and machines.

schedule the k large jobs inside the gap, there is a layer s' in the shifted schedule, for each layer $s \in \mathcal{S}$ that has at least as many machines and resources reserved for the small jobs as the layer s. □

7.3.1 Algorithm

The algorithm works similar to the algorithm in Section 7.2.6. The only step that we change is the guessing step (Step 4). In the following, we describe the altered step. Given a value $T' := i\varepsilon' T$, we determine the set $\mathcal{S}$ and guess the starting positions of the large jobs $\mathcal{J}_{L,W}$ as well as the vector $(m_s^{(L)}, R_s^{(L)})_{s\in\mathcal{S}}$. Note that this vector defines the resource and machine requirements of only the large and not the huge jobs. For each of these guesses, we try to solve the linear program $\mathrm{LP}_{\mathcal{J}_{L,N},(m_s^{(L)},R_s^{(L)})_{s\in\mathcal{S}}}$ which needs $(m\log(R)/\delta)^{\mathcal{O}(1)}$ operations. Note that neither $\mathcal{J}_{L,W}$ nor $\mathrm{LP}_{\mathcal{J}_{L,N}}$ contain huge jobs.

If the linear program has a solution, we remove the at most $3|S|$ large jobs, which we have to remove to guarantee a feasible schedule, and guess and identify the case for the huge jobs. More precisely, we have to guess τ and ρ and the at most $2k$ huge jobs ending before or after these values and their starting positions. Then, we identify the case and the other variables dependent on the previous guesses and the solution of the linear program.

In detail, we perform the following steps: First, we guess τ as well as the at most $k-1$ huge jobs ending before τ and their starting positions ($\mathcal{O}((1/\varepsilon^2)^k)$ possibilities). Afterward, we identify the applying case:

Case 1.1 We schedule the removed jobs at τ and shift the jobs as described.

Case 1.2 We identify i and guess ρ. For each of these guesses, we identify the resulting case:

- Case 1.2.1. We shift the jobs as described and schedule the removed large jobs at ι.
- Case 1.2.2. We guess the at most k huge jobs ending after ρ and their starting positions. For each of these guesses we shift the jobs as described and schedule the removed jobs at T'.

Case 2. We guess ρ and the at most k huge jobs ending after ρ as well as their starting positions. For each of these guesses, we identify the corresponding case 2.1, 2.2.1 or 2.2.2:

- Case 2.1. We find τ' and schedule the jobs as described in Case 2.1.1. or Case 2.1.2. depending on the relation of τ' and ρ.
- Case 2.2.1. We schedule the jobs as described.
- Case 2.2.2. For each guess of ρ', we try to schedule the jobs as described.

For each of the resulting schedules of huge and large jobs, we first verify its feasibility. Afterward (for each feasible guess), we determine the residual machines and resources per layer and schedule the small jobs by solving the linear program $LP_{\mathcal{S}',\mathcal{J}_S}$, where $\mathcal{S}'$ additionally contains the start points that we added by the shift. If this linear program has a solution, the guess for the case was correct and we save the guess, the position of the gap, and both LP solutions and try the next smaller value

for T'. Otherwise, we try the next guess for the case. If all the guesses to schedule the huge jobs fail, we try another guess for the large jobs and finally a larger value for T' if all the guesses fail.

To bound the total number of guesses that we add by this procedure, note that we have to guess τ, ρ and ρ' from at most $\mathcal{O}(|\mathcal{S}|)$ possibilities. Further, we for each of these guesses, we have to guess the starting positions of at most $2k$ jobs from at most $\mathcal{O}(1/\varepsilon^2)$ possibilities since huge jobs start at multiples of $\varepsilon^2 T$. Therefore, the total number of guesses for the large jobs is bounded by $(1/\varepsilon)^{\mathcal{O}(k)} \cdot \mathcal{O}(|\mathcal{S}|^3)$. Since $k \leqslant 3|\mathcal{S}|$, this guess for the huge jobs lengthens the running time by a factor of at most $(1/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon^2)}}$. This concludes the proof of Theorem 7.2.

7.4 Improving the APTAS to an AEPTAS

In this section, we improve the running time of the APTAS described in Section 7.2 such that $1/\varepsilon$ no longer appears in the exponent of the input size, i.e., it does not appear in the exponent of n, m, or $\log(R)$. Remember the only critical step in the APTAS, where we use a running time of the form $(n, m, \log(R))^{f(1/\varepsilon)}$ is the guess for the large jobs. We will improve the running time by using a rougher estimate in the vector $(m_s^{(L)}, R_s^{(L)})_{s\in\mathcal{S}}$. To make this guessing step possible without adding too many jobs at the end of the schedule, we have to consider the large jobs more carefully and slightly change the schedule of the small jobs. However, the scheduling of the medium jobs remains the same.

7.4.1 Large Jobs

We partition the set of large jobs into wide, medium and narrow jobs. Like in the previous sections, wide jobs $j \in \mathcal{J}_{LW}$ have a resource requirement of $r(j) \geqslant \alpha R$ for a given value $\alpha \in (0,1)$ which differs from the choices before and is specified later. Medium large jobs are in the set $\mathcal{J}_{LM} := \{j \in \mathcal{J}_L | \alpha R > r(j) > R/m\}$ and narrow jobs $j \in \mathcal{J}_{LN}$ have a resource requirement of $r(j) < R/m$. As seen before, we can enumerate all possible combinations of starting positions of wide large jobs in $\mathcal{O}((1/\varepsilon\delta)^{1/\alpha\delta})$.

To speed up the running time, we have to find a better handling of the medium and narrow large jobs.

Narrow Large Jobs. First, we consider the narrow jobs in $\mathcal{J}_{LN}$ with resource requirement at most R/m. For each rounded processing time $p \in \mathcal{P}$, we sort the jobs in descending order of resource requirement and partition the jobs into sets with $m\varepsilon/|\mathcal{P}| = \mathcal{O}(m\varepsilon^4)$ jobs each such that the group containing the jobs with the smallest resource requirement may have fewer jobs. Since there are at most m/δ jobs, there are at most $\mathcal{O}((m/\delta)/(m\varepsilon^4) + |\mathcal{P}|) \in \mathcal{O}(1/\varepsilon^4\delta)$ groups total. To find the partition into the groups it is enough to find the $\mathcal{O}(1/\varepsilon^4\delta)$ jobs with the largest resource requirement in each group. Using the techniques from Lemma 3.3, i.e., the modified median algorithm, it is possible to find these jobs in $\mathcal{O}((m/\delta) \cdot \log(1/\varepsilon^4\delta))$.

Lemma 7.14. *By discarding at most* $2\varepsilon m$ *jobs with total resource requirement at most* $2\varepsilon R$*, we can find the optimal placement of rounded narrow large jobs in at most* $(1/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}}$ *guesses.*

Proof. For each processing time $p \in \mathcal{P}$, we round up the requirements of each other job to the largest one occurring in his group. Consider an optimal schedule for the jobs that have a rounded processing time but no rounded resource requirement. In this schedule, it is possible to replace the jobs from one group with the rounded jobs in the next group without violating the resource constraint, since these jobs have a smaller resource requirement as the previously scheduled jobs. The widest group per processing time cannot be scheduled instead of other jobs in the considered optimal schedule. Hence, we discard these jobs from the schedule and place them in a later step at the end of the schedule. Since we discarded one group per processing time in $\mathcal{P}$ and each group has at most $\varepsilon m/|\mathcal{P}|$ jobs, their number is bounded by εm. Since these jobs have a resource requirement of at most R/m the discarded jobs have a total resource requirement of at most εR.

Next, we discuss how we can guess the start points of the rounded jobs in few guesses total. Let $\mathcal{R}_{L,N}$ be the set of rounded resource requirements. Its size $|\mathcal{R}_{L,N}|$ is bounded by $\mathcal{O}(1/\varepsilon^4\delta)$. Further, let $\mathcal{J}'_{L,N}$ be the set of

rounded jobs. We guess for each starting time s and each rounded resource requirement $r \leqslant R/m$ how many jobs $j \in \mathcal{J}'_{L,N}$ with $r(j) = r$ start at s in an optimal solution. However, we do not guess the exact number. Instead, for a $\lambda \in \mathcal{O}_\varepsilon(1)$, we guess the largest multiple of λm smaller than the actual number and discard the residual jobs. Hence, we have at most $(1/\lambda)^{|S|\cdot|\mathcal{R}_{L,N}|} \leqslant (1/\lambda)^{4/\varepsilon^5\delta^2}$ guesses. The total number of jobs we discard is bounded by $\lambda m \cdot |\mathcal{S}| \cdot |\mathcal{R}_{L,N}| \leqslant \mathcal{O}(\lambda m/\varepsilon^5\delta^2)$. Hence, if we choose $\lambda \in \mathcal{O}(\varepsilon^6\delta^2)$ small enough, we remove at most εm jobs total. These jobs have a total resource requirement of at most $\varepsilon m \cdot R/m \leqslant \varepsilon R$.

The total number of jobs which are discarded to handle the narrow jobs is bounded by $2\varepsilon m$ and their resource requirement is bounded by $2\varepsilon R$. The number of operations for the simplification steps for this set of jobs is bounded by $\mathcal{O}(m \cdot (1/\varepsilon)^{\mathcal{O}(1/\varepsilon)})$ while the total number of possible guesses is bounded by $\mathcal{O}((1/\lambda)^{4/\varepsilon^5\delta^2}) = (1/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}}$. □

Medium Large Jobs. Last, we consider the medium large jobs, i.e., the set $\mathcal{J}_{LM}$. We schedule these jobs similar to the scheduling of the narrow large jobs in Section 7.2.2. However, we change the guess for the vector $(m_s^{(L)}, R_s^{(L)})_{s\in\mathcal{S}}$.

In the first step, we round the resource requirement of these jobs. Note that this rounding is not necessary to find the AEPTAS, but improves the running time for the medium large jobs from being linear dependent on m to a logarithmic dependence. Let $j \in \mathcal{J}_{LM}$ with $r(j) \in [2^t R/m, 2^{t+1}R/m]$. We round its resource requirement to the next larger multiple of $\alpha \cdot 2^t R/m$. Let $\tilde{r}(j)$ be this rounded resource requirement of j. It holds that $\tilde{r}(j) \leqslant (1+\alpha)r(j)$ and, hence in the considered optimal schedule, the resource requirement has increased by at most αR per layer. The interval $[2^t R/m, 2^{t+1}R/m)$ contains at most $2/\alpha$ multiples of $2^t R\alpha/m$, and there are at most $\log(m)$ such intervals intersecting the interval $[R/m, \alpha R)$. Hence after this rounding, there are at most $\mathcal{O}(\log(m)/\alpha)$ different resource requirements.

Let $\mathcal{R}_{LM}$ be the set of rounded resource requirements of the jobs in $\mathcal{J}_{LM}$ and let $n(r,p)$ be the number of jobs in $\mathcal{J}_{LM}$ with rounded resource requirement $r \in \mathcal{R}_{LM}$ and rounded processing time $p \in \mathcal{P}$.

For each starting point s, we guess the total number $m_s^{(L)}$ of jobs in $\mathcal{J}_{L,M}$

which are processed in the corresponding layer and their total rounded resource requirement $R_s^{(L)}$. More precisely, we guess $R_s^{(L)}$ as the next larger multiple of αR in the interval $[0, R + 2\alpha R]$ and $m_s^{(L)}$ as the next larger multiple of γm in the interval $[0, m + \gamma m]$, where $\gamma \in \mathcal{O}_\varepsilon(1)$. Hence due to the rounding, we later have to remove jobs with total resource requirement of at most $2\alpha R$ per layer. In total, this leaves $\mathcal{O}((1/\gamma\alpha)^{1/\varepsilon\delta})$ guesses for $(m_s^{(L)}, R_s^{(L)})_{s\in\mathcal{S}}$.

For a given guess $(m_s^{(L)}, R_s^{(L)})_{s\in\mathcal{S}}$, we solve the following linear program $\mathrm{LP}_{\mathcal{P},\mathcal{R},(m_s^{(L)},R_s^{(L)})_{s\in\mathcal{S}}}$:

$$\sum_{r\in\mathcal{R}}\sum_{p\in\mathcal{P}}\sum_{\substack{s'\in\mathcal{S}\\ s-p<s'\leqslant s}} r x_{r,p,s'} \leqslant R_s^{(L)} \qquad \forall s \in S \tag{7.18}$$

$$\sum_{r\in\mathcal{R}}\sum_{p\in\mathcal{P}}\sum_{\substack{s'\in\mathcal{S}\\ s-p<s'\leqslant s}} x_{r,p,s'} \leqslant m_s^{(L)} \qquad \forall s \in S \tag{7.19}$$

$$\sum_{p\in\mathcal{P}}\sum_{s=0}^{T-p} x_{r,p,s} = n(r,p) \qquad \forall p \in \mathcal{P}, r \in \mathcal{R}_{L,M} \tag{7.20}$$

$$x_{r,p,s} \geqslant 0 \qquad \forall p \in \mathcal{P}, r \in \mathcal{R}_{L,M}, s \in S \tag{7.21}$$

The variable $x_{r,p,s}$ represents the number of jobs with rounded processing time p and rounded resource requirement r, which start at the starting point s. The first two conditions ensure that in each layer we do not exceed the guessed number of resources or machines. The third condition ensures that each job is scheduled. It is easy to see that if there is an optimal schedule with resource requirement and machine number such that the values $(m_s^{(L)}, R_s^{(L)})_{s\in S}$ are an upper bound, we can transform it to a solution of this linear program. On the other hand, if there is no solution, we can discard the current guess. This linear program has $2|S| + |\mathcal{R}_{L,M}||\mathcal{P}|$ conditions and $|S||\mathcal{R}_{L,M}||\mathcal{P}|$ variables. Note that by scaling the condition (7.18) with m/R, all the values on the right hand side are integers in the interval $[1, 2m]$. Hence, we can find a basic solution with at most $2|S| + |\mathcal{R}_{L,M}||\mathcal{P}|$ non zero components in at most $(\log(2m)|S||\mathcal{R}_{L,M}||\mathcal{P}|)^{\mathcal{O}(1)} = (\log(m)/\gamma\varepsilon\delta)^{\mathcal{O}(1)}$ operations using the Ellipsoid-Method or any other polynomial time algorithm to solve linear programs.

The non zero components concerning one resource requirement and processing time always add up to an integer. Therefore, if for a given resource requirement r and a processing time p the sum is split into k values, the fractional parts can add up to at most $k-1$. Hence, we have to remove at most $2|S|$ jobs to schedule all jobs integral. Since each job has a resource requirement of at most αR the total resource requirement of the removed jobs is bounded by $2\alpha R/\varepsilon\delta \leqslant \varepsilon R$, if $\alpha \leqslant \varepsilon^2\delta/2$. Furthermore, the number of removed jobs is bounded by $2/\varepsilon\delta \leqslant \varepsilon m$ if $m \geqslant 2/\varepsilon^2\delta$. This integral solution can be found in at most $\mathcal{O}((m/\delta) + (|S| + |\mathcal{R}_{L,M}||\mathcal{P}|))$ operations, by iterating all the non zero components.

When the algorithm has found a solution to the linear program, it proceeds with trying to schedule the small jobs. However, when the binary search is finished, the algorithm has to produce a schedule. In Section 7.4.3, we discuss the number of jobs and their total resource requirement, which have to be removed to make the schedule feasible, despite the rounding of the total resource requirement and the rounding of the total machine requirement per layer.

7.4.2 Small Jobs

Since we already can solve the linear program for small jobs in an efficient number of operations, the simplification steps remain the same for this set of jobs. What changes is the number of machines and resource that can be used for small jobs in the linear program. When the algorithm schedules the small jobs, the number of machines and the resource amount per layer is given by the residual capacities due to the guess for the large jobs. However, we have to take into account that we had rounded the resource and machine values. Therefore, we use a little bit more than the residual amounts. Let $m_s^{(L)} := t\gamma m$ and $R_s^{(L)} := t'\alpha R$ for a given layer s, and $m_{s,L}$ be the number of wide large jobs and $R_{s,L}$ be their resource requirement in this layer. Furthermore, let $m_{s,N}$ be the number of jobs with resource requirement smaller than R/m, scheduled in this layer and $R_{s,N}$ be their total resource requirement. Then, we will use at most $m_:^{(S)} = m - (t-1)_+\gamma m - m_{s,L} - m_{s,N}$ machines and at most $R_s^{(S)} := R - (t'-2)_+\alpha R - R_{s,L} - R_{s,N}$ resource for small jobs in this layer.

After we have determined the vector $(m_s^{(S)}, R_s^{(S)})_{s\in\mathcal{S}}$, we solve the configuration LP for small jobs $LP_{\mathcal{S},\mathcal{J}_S}$, see Section 7.2.4. By Lemma 7.7, we can find a solution with at most $\mathcal{O}(|\mathcal{S}| \cdot |\mathcal{J}'_S|)$ non zero components in at most $\mathcal{O}(\log(m)^3 \cdot (1/\varepsilon)^{\mathcal{O}(1/\varepsilon)})$ operations. Since we allowed the usage of up to γm additional machines per layer, we have to remove some of the jobs in a later step. In a layer where we have reserved more than $m/2$ machines for small jobs, we remove some of them to make the schedule feasible. In the other layers, we remove large jobs.

Lemma 7.15. *By extending the schedule by at most $\varepsilon p(x)$, we can remove the smallest γm jobs from each configuration of small jobs that contains more than $m/4$ jobs.*

Proof. Consider a configuration for small jobs that uses at least $m/4$ machines. The γm most narrow jobs in this configuration can use at most $\gamma m \cdot R/(m/4) = 4\gamma R$ resource. We remove these jobs and build a new configuration with them, which has the same height, as the configuration, from which these jobs were removed. The total height of all configurations generated by removing the jobs has a height of at most $p(x)$. Since each configuration contains at most γm jobs which have a total resource requirement of at most $4\gamma R$, we can partition this stack of configurations into $1/4\gamma$ stacks of height $4\gamma p(x)$ and schedule these stacks at the same time. Since $4\gamma < \varepsilon$, we expand the schedule by at most $\varepsilon p(x)$. □

Therefore, we can reduce each $m_s^{(S)} \geqslant m/4$ to $m_s^{(S)} - \gamma m$ and there will still be a solution to LP_S, when we extend the right hand side condition (7.5) as follows $\sum_{c\in\mathcal{C}_\top} x_{\top,c} \leqslant 3\varepsilon T + \varepsilon p(x) \leqslant 3\varepsilon T + \varepsilon(1+\varepsilon)T'$.

7.4.3 Meeting the resource and machine constraints.

To meet the machine constraint, we show that it is possible to remove up to γm large jobs from the layers where there are at least $m/4 - \gamma m$ large jobs. Note that these layers are the only ones where we have to remove jobs to meet the machine constraint since in all the other layers, we can remove enough small jobs to meet this constraint. Further in this section, we prove that it is possible to remove large jobs from the configurations to

meet the resource constraint. Note that this step will be performed after the binary search for the right value for T' is done.

Lemma 7.16. *To meet the resource constraint, we have to remove at most $2\alpha m$ jobs per layer, with total resource requirement of at most $3\alpha R$.*

Proof. We consider the layer s. If the total resource requirement of the medium large jobs intersecting this layer is bounded by $(R_s^{(L)} - 2\alpha R)_+$, we do not remove any job. Otherwise, we remove medium large jobs until the residual jobs need at most $R_s^{(L)} - 2\alpha R$ resource. Since each medium large job has a requirement of at most αR resources, we remove jobs with a total resource requirement of at most $3\alpha R$. Furthermore, each medium large job has a resource requirement of at least R/m and, hence, we remove at most $2\alpha m$ jobs. □

After the resource constraint is fulfilled in each layer, we remove jobs to fulfill the machine requirement. Note that the machine constraint cannot be exceeded by more than γm since we rounded the machine requirement of the medium large jobs to these values.

Lemma 7.17. *To meet the machine constraint, we remove at most γm large jobs with a total resource requirement of at most $2\gamma R$ per layer.*

Proof. Consider a layer s containing a point in time τ, where at least $m + 1$ jobs are scheduled but at most $(1 + \gamma)m$. In this layer there cannot be more than $m/4$ machines used by small jobs since we had reduced the value $m_s^{(S)}$ in this case.

If there are at least γm narrow large jobs, i.e., jobs with a resource requirement smaller than R/m, we can remove γm of them. The removed jobs then have a resource requirement of at most γR.

Otherwise, at least $m - \gamma m - m/4 - 1/\alpha \geqslant m/2$ machines are used by medium large jobs since there are at most $1/\alpha$ wide large jobs per layer and we can choose γ such that $\gamma m + 1/\alpha \leqslant m/4$. For example, we can choose $\gamma \leqslant \varepsilon/8$ since we can assume that $1/\alpha \leqslant m/8$. Hence, the γm most narrow jobs have a total resource requirement of at most $\gamma m \cdot R/(m/2) \leqslant 2\gamma R$ and we remove them. Note that we can find these γm most narrow jobs in $\mathcal{O}(|\mathcal{J}_{LM}|)$ by using the a linear selection algorithm to find the ith element as in [11]. □

In the following, we summarize the total number and resource requirements of large jobs we removed from the schedule to speed up the running time of the algorithm. We did not remove any wide large job. To round the narrow large jobs, we remove at most $2\varepsilon m$ jobs with total resource requirement at most $2\varepsilon R$ by Lemma 7.14. To allow the rougher estimate on $(m_s, R_s)_{s\in\mathcal{S}}$, we remove at most $(2\alpha m + \gamma m) \cdot \mathcal{O}(\varepsilon\delta)$ jobs with resource requirement at most $3\alpha R + 2\gamma R \cdot \mathcal{O}(\varepsilon\delta)$, by Lemmas 7.16 and 7.17.

Remark 7.18. The total number of removed large jobs is bounded by $2\varepsilon m + (2\alpha m + \gamma m) \cdot \mathcal{O}(\varepsilon\delta) \leqslant 3\varepsilon m$ and their resource requirement by $2\varepsilon R + (3\alpha R + 2\gamma R) \cdot \mathcal{O}(\varepsilon\delta) \leqslant 3\varepsilon R$, when we choose $\gamma, \alpha \in \mathcal{O}(\varepsilon^2\delta)$ adequately.

Hence, it is possible to start all the removed jobs at the same time, without violating the resource or the machine constraint if $\varepsilon \leqslant 3$. Note that we assume that $m > 8/\alpha$ in the algorithm. Otherwise, we can guess the start times of all the large jobs $(1+\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}}$ and hence have an EPTAS for this case.

7.4.4 Summary of the Algorithm

In the following, we summarize the algorithm. The main steps of the algorithm are the same as for the algorithm in Section 7.2.6, hence, for a given instance $I = (\mathcal{J}, m, R)$ of Single Resource Constraint Scheduling and an $\varepsilon \in (0, 1)$ we only discuss the steps of the algorithm that have changed:
Step 2: Simplification We simplify the given instance I by partitioning and rounding the jobs. First, we round all processing times to multiples of $\varepsilon' T/n$. Next, we find the values δ and μ in $\mathcal{O}(n + 1/\varepsilon)$ as described in Lemma 7.4 and partition the set of jobs $\mathcal{J}(I_{r,1})$ into large, medium and small jobs in $\mathcal{O}(n)$. Then we generate the rounded instances $I_{r,2}$ and $I_{r,3}$ as before in $\mathcal{O}(n \log(1/\varepsilon))$. Next, we partition the large jobs in wide medium and narrow ones and round the narrow jobs as described in Lemma 7.14 using $\mathcal{O}((m/\delta) \cdot \log(1/\varepsilon^4\delta))$ operations and round the medium large jobs as described using at most $\mathcal{O}((m/\delta))$ operations. We call this instance $I_{r,4}$. The total number of operations in this step is bounded by $\mathcal{O}(n \log(1/\varepsilon) + m(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)})$.

Step 3: Binary Search Framework For the rounded instance $I_{r,4}$, we will find the value $T' = (1 + i\varepsilon')T$, for $i \in [0, 4/\varepsilon + 8] \cap \mathbb{N}$ such that $T' - \varepsilon' T \leqslant$

$\mathrm{OPT}(I_{r,4}) \leqslant T'$ and give a schedule with makespan at most $(1+\mathcal{O}(\varepsilon'))T' + p_{\max} = (1+\mathcal{O}(\varepsilon'))\mathrm{OPT}(I) + p_{\max}$. We use dual approximation and try the values T' in binary search fashion in the next step. As a consequence, we try at most $\mathcal{O}(\log(1/\varepsilon))$ different values for T'.

Step 4: Guessing Step Given a value $T' := i\varepsilon' T$, we determine the corresponding set $\mathcal{S}$. Afterward, we try each possibility to schedule the large wide jobs $\mathcal{J}_{LW}$ and each possibility for the vector $(m_s^{(L)}, R_s^{(L)})_{s\in\mathcal{S}}$ and each guess for start points of the large narrow jobs. The number of possibilities to schedule these large jobs is bounded by $(1/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}}$ thanks to the improvements in Section 7.4.1. For a given guess, we try to solve the linear program $\mathrm{LP}_{\mathcal{P},\mathcal{R},(m_s^{(L)},R_s^{(L)})_{s\in\mathcal{S}}}$ for the medium large jobs. We can find a solution to this linear program with at most $2|\mathcal{S}| + |\mathcal{P}|$ non zero components in at most $(\log(m)/\gamma\varepsilon\delta)^{\mathcal{O}(1)}$ operations, or decide that there is no solution to this linear program.

If the linear program is unsolvable, we discard the current guess. Otherwise, we try to solve the linear program $LP_{\mathcal{S},\mathcal{J}_S}$ to place the small jobs. In this linear program, we use the modified values $(m_s^{(S)}, R_s^{(S)})_{s\in\mathcal{S}}$, where we already reduced the number of allowed jobs in layers with more than $m/4$ machines for small jobs and where we modified the first condition (7.5) to $\sum_{c\in\mathcal{C}_\top} x_{\top,c} \leqslant 3\varepsilon T + \varepsilon p(x) \leqslant 3\varepsilon T + \varepsilon(1+\varepsilon)T'$. By Lemma 7.7, a solution to the relaxed version can be found in $\mathcal{O}(\log(m)/\varepsilon^8\delta^3)$ or we can decide that it has no solution.

If we find such a solution, we save the guess and both LP-solutions and try the next smaller value for T' in binary search fashion. Otherwise, we try the next guess. If all the guesses deny the solvability of one of the linear programs, we try the next larger value for T' in binary search fashion. The total running time of this step is bounded by $\log(m)^{\mathcal{O}(1)} \cdot (1/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}}$.

Step 5: Scheduling the Original Jobs When the binary search procedure is finished, we have a guess how to schedule the large jobs and both LP solutions. By the Lemmas 7.6, 7.10, 7.16, and 7.17, we can transform these solutions to schedules for the jobs in $\mathcal{J}(I_{r,2})$ with a loss of at most $\mathcal{O}(\varepsilon')T' + p_{\max}$ in the makespan. As before, this transformation needs at most $\mathcal{O}(n + m(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)})$ operations.

To this schedule, we add the medium sized jobs $\mathcal{J}_M$ to the schedule using the algorithm described in Section 7.2.5. By Lemma 7.11 this adds at most $\mathcal{O}(\varepsilon')\mathrm{OPT}(I)$ to the makespan and needs at most $\mathcal{O}(n)$ operations.

This schedule for the instance $I_{r,2}$ is then transformed by replacing the jobs in $\mathcal{J}(I_{r,2})$ by the original jobs in $\mathcal{J}(I)$. Note that this is possible since the processing time of these jobs is smaller than the processing time of the jobs in $\mathcal{J}(I)$.

Finally, we can transform the schedule to one where each job is started at an integral starting point, by iterating the jobs one by one and reduce the starting time of this jobs to the next smaller integral if it is not integral already. After this step, the schedule still fulfills all the constraints since now all the jobs end at integral points as well. The total running time of this step is bounded by $\mathcal{O}(n + (1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}m)$.

In total the running time of the algorithm is bounded by

$$\begin{aligned}&\mathcal{O}(n\log(1/\varepsilon) + m(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)} + \log(m)^{\mathcal{O}(1)} \cdot (1/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}})\\ &= \mathcal{O}(n\log(1/\varepsilon)) + \mathcal{O}(m)\mathcal{O}_\varepsilon(1).\end{aligned}$$

7.5 Conclusion

In this chapter, we presented an algorithm for Single Resource Constraint Scheduling with absolute approximation ratio $(3/2 + \varepsilon)$, which closes the gap between inapproximation result $3/2$ and the best algorithm. Furthermore, we presented an AEPTAS for this problem.

It would be interesting to see if the techniques used in these algorithms can be extended to the moldable case of the problem; i.e., the case where the processing time of a job depends on the number of resources that are assigned to it.

Finally, note that the AEPTAS from Section 7.4 does schedule the jobs in a way that might make it impossible for the jobs to use the resource contiguously, while the AFPTAS described in Chapter 6 can guarantee this property. It would be interesting to see if the AEPTAS can be extended to have the property that all the jobs can be scheduled contiguously.

Chapter 8

A Toolbox for Linear Time Approximations on Multiple Clusters

In this chapter, we study multi cluster variants of Parallel Task Scheduling (PTS), Strip Packing (SP), and Single Resource Constraint Scheduling (SRCS), called Multiple Cluster Scheduling (MCS), Multiple Strip Packing (MSP), and Single Resource Constraint Multiple Cluster Scheduling (SRCMCS). For these variants, there is no algorithm with approximation ratio better than 2 unless P = NP. In this paper, we present algorithms with approximation ratio 2 and running time $\mathcal{O}(n)$ for these problems. While 2 approximations were known for Multiple Cluster Scheduling (MCS) and Multiple Strip Packing (MSP), the running time of these algorithm is at least $\Omega(n^{256})$ in the worst case. Therefore, $\mathcal{O}(n)$ algorithms are surprising and the best possible. We achieve this result by calling corresponding AEPTASs, which have an approximation guarantee of $(1+\varepsilon)\mathrm{OPT} + p_{\max}$ and running time of the form $\mathcal{O}(n \log(1/\varepsilon) + f(1/\varepsilon))$, with a constant ε to schedule the jobs on a single cluster. This schedule is then distributed on the N clusters in $\mathcal{O}(n)$. Moreover, this distribution technique can be applied to any variant of of Multi Cluster Scheduling for which there exists an AEPTAS with additive term $p_{\max}$.

While the above result is strong from a theoretical point of view, it might not be very practical due to a large hidden constant caused by calling an AEPTAS with a constant $\varepsilon \geqslant 1/6$ as subroutine. Nevertheless, we point out that the general approach of finding first a schedule on one cluster and then distributing it onto the other clusters might come in handy in practical approaches. We demonstrate this by presenting a practical

algorithm with running time $\mathcal{O}(n \log(n))$, without hidden constants that is a 9/4-approximation for one third of all possible instances, i.e, all instances where the number of clusters is dividable by 3, and has an approximation ratio of at most 2.3 for all instances with at least 9 clusters.

Some of the results presented in this chapter are published in [64] and can be found as a preprint on arXiv [65].

8.1 Results

In the optimization problem Multiple Cluster Scheduling (MCS), we are given $n \in \mathbb{N}$ parallel jobs $\mathcal{J}$ and $N \in \mathbb{N}$ clusters. Each cluster consists of $m \in \mathbb{N}$ identical machines and each job $j \in \mathcal{J}$ has a processing time $p(j) \in \mathbb{N}$ and a machine requirement $m(j) \in \mathbb{N}_{\leqslant m}$. A schedule $S = (\sigma, \rho)$ of the jobs consists of two functions; $\sigma : \mathcal{J} \to \mathbb{N}$ which assigns jobs to starting points, and $\rho : \mathcal{J} \to \{1, \dots N\}$ which assigns jobs to the clusters. The objective is to find a feasible schedule of all the jobs, which minimizes the makespan, i.e., which minimizes $\max\{p(j) + \sigma(j) | j \in \mathcal{J}\}$. A schedule is feasible if at every time $\tau \in \mathbb{N}$ and any Cluster $i \in \mathbb{N}$ the number of used machines is bounded by m, i.e., if

$$\sum_{j \in \mathcal{J}, \sigma(j) \leqslant \tau < \sigma(j) + p(j), \rho(j) = i} m(j) \leqslant m \qquad \forall i \in \{1, \dots, N\}, \tau \in \mathbb{N}.$$

If the number of clusters is bounded by one, the problem is called Parallel Task Scheduling (PTS). We can assume that $n > N$ since otherwise an optimal schedule would place each job alone on one cluster and thus the problem is not hard.

The other considered problem is a closely related variant of MCS called Multiple Strip Packing (MSP). The difference is that the jobs have to be allocated on contiguous machines. We are given $n \in \mathbb{N}$ rectangular items $\mathcal{I}$ and $N \in \mathbb{N}$ strips (also called clusters). Each strip has an infinite height and the same width $W \in \mathbb{N}$. Each item $i \in \mathcal{I}$ has a width $w(i)$ and a height $h(i)$. The objective is to find a feasible packing of the items into the strips such that the packing height is minimized. A packing is feasible if all the items are placed overlapping free into the strips. If the number of strips is bounded by one, the problem is called Strip Packing (SP).

Finally, we will consider a cluster variant of Single Resource Constraint Scheduling (SRCS) which we call Single Resource Constraint Multiple Cluster Scheduling (SRCMCS). Here each cluster consists of $m \in \mathbb{N}$ machines and has an additional resource $R \in \mathbb{N}$. To be processed, each job requires one machine and an amount of the resource $r(j) \leqslant R$ both belonging to the same cluster.

Strip Packing and Parallel Task Scheduling are classical optimization problems and the extension of these problems to multiple strips or clusters comes natural. Furthermore, these problems can be motivated by real world problems. One example, as stated in [107], is the following: In operating systems, MSP arises in the computer grid and server consolidation [85]. In the system supporting server consolidation on many-core chip multi processors, multiple server applications are deployed onto virtual machines. Every virtual machine is allocated several processors and each application might require a number of processors simultaneously. Hence, a virtual machine can be regarded as a cluster and server applications can be represented as parallel tasks. Similarly, in the distributed virtual machines environment, each physical machine can be regarded as a strip while virtual machines are represented as rectangles. It is quite natural to investigate the packing algorithm by minimizing the maximum height of the strips. This is related to the problem of maximizing the throughput, which is commonly used in the area of operating systems.

The considered problems cannot be approximated better than 2 unless P = NP, see [107]. On the other hand, the best algorithms so far are 2-approximations, see [14] for MSP and [70] for MCS. Both algorithms have a large worst case running time of $\Omega(n^{256})$ since they use an algorithm with running time $n^{\Omega(1/\varepsilon^{1/\varepsilon})}$ with constant $\varepsilon = 1/4$ as a subroutine. Because of this running time, efforts have been made to speed up the running time in expense of the approximation ratio, see for example [13], [107], and [16]. We refer to Section 2.4 for a more detailed overview on the work related to all considered versions of Multiple Cluster Scheduling. For MCS and MSP, we present 2-approximations, where we managed to improve the running time drastically with regard to the $\mathcal{O}$-notation.

Theorem 8.1. *There are algorithms for MCS, MSP and SRCMCS with approximation ratio* 2 *and running time* $\mathcal{O}(n)$ *if* $N > 2$.

Furthermore, there are algorithms for MCS and MSP with approximation ratio 2 *and running time* $\mathcal{O}(n\log(n))$ *and* $\mathcal{O}(n\log^2(n)/\log(\log(n)))$ *respectively if* $N \in \{1,2\}$.

Note that the running time of these algorithms is the best possible from a theoretical point of view with respect to the $\mathcal{O}$-notation for $N \geqslant 3$. Since we need to assign a start point to each job, we cannot assume that there is an algorithm for MCS with running time strictly faster than $\Omega(n)$.

To achieve these results, we use as a subroutine an AEPTAS for the optimization problems Parallel Task Scheduling (PTS), Strip Packing (SP), and Single Resource Constraint Scheduling (SRCS) respectively. Regarding PTS, we improved the running time of an algorithm by Jansen [50] and developed an AEPTAS. For SP, we find an AEPTAS as well. However, the running time depending on $1/\varepsilon$ is worse than in the AEPTAS for PTS. Note that this algorithm is the first AEPTAS for SP that has an additive term of $h_{\max}$.

Theorem 8.2. *There are AEPTASs for PTS and SP with additive term* $p_{\max}$ *and* $h_{\max}$ *respectively.*

Note that for Single Resource Constraint Scheduling we have seen such an AEPTAS in the previous section. These algorithms can be used to find an AEPTAS for each of the problems MCS, MSP, and SRCMCS by cutting the solution for one cluster or strip into segments of height $(1+\varepsilon)\mathrm{OPT}$. The jobs overlapping the cluster borders add further $p_{\max}$ to the approximation ratio resulting in a additional algorithm for MCS with approximation guarantee $(1+\varepsilon)\mathrm{OPT} + p_{\max}$.

Theorem 8.3. *There are AEPTASs for MCS, MSP, and SRCMCS with additive term* $p_{\max}$ *or* $h_{\max}$ *respectively.*

Note that this result implies an algorithm for SRCMCS with approximation ratio $(2+\varepsilon)$ and running time $\mathcal{O}(n)\cdot\mathcal{O}_\varepsilon(1)$ for the case $N \in \{1,2\}$. The algorithms from Theorem 8.1 use the algorithms from Theorem 8.2 as a subroutine with a constant value $\varepsilon = 1/6$ if $N = 2$, $\varepsilon = 1/5$ if $N = 5$, $\varepsilon = 1/3$ for all $N \in 3\mathbb{N}$ and $\varepsilon = 1/4$ for all other $N \geqslant 2$. As a result, the running time of the algorithm can be rather large, while the $\mathcal{O}$-notation suggests otherwise since it hides all the constants. Due to this fact, we

have developed a truly fast algorithm for MCS where the most expensive part is sorting the jobs. However, this improved running time yields a loss in the approximation factor.

Theorem 8.4. *There is a fast $\mathcal{O}(n \log(n))$ algorithm for MCS with approximation ratio $9/4$ if $N = 3i$, $(9i+5)/(4i+2) \leqslant 9/4 + \frac{3}{8N}$ if $N = 3i+1$, and $(9i+10)/(4i+4) \leqslant 9/4 + \frac{3}{4N}$ if $N = 3i+2$ for some $i \in \mathbb{N}$.*

Note that the approximation ratio of the algorithm from Theorem 8.4 is worse than $7/3$ for the cases that $N \in \{2, 5\}$ and exactly $7/3$ for the case that $N \in \{4, 8\}$. However if $N \geqslant 9$, the approximation ratio is bounded by 2.3, and $9/4 + \frac{3}{8N}$ as well as $9/4 + \frac{3}{4N}$ converge to $9/4$ for $N \to \infty$.

Organization of this Chapter

The $\mathcal{O}(n)$ algorithm consists of two steps. First, we use an AEPTAS for MCS or MSP to find a schedule on two clusters, one with makespan at most $(1+\varepsilon)N\text{OPT}$ and the other with makespan at most $p_{\max} \leqslant \text{OPT}$. This schedule on the two clusters is then distributed onto the N clusters using a partitioning technique, as we call it. This partitioning technique is the main accomplishment of this chapter and presented in Section 8.2. The AEPTAS for Parallel Task Scheduling can be found in Section 8.3 while the AEPTAS for Strip Packing can be found in Section 8.5. In Section 8.4, we present the algorithm from Theorem 8.4 that finds an approximation without the need to call the AEPTAS as a subroutine but uses the partitioning technique as well.

8.2 Partitioning Technique

In this section, we describe the central idea which leads to a linear running time algorithm. Note that MSP can be interpreted as a scheduling problem, where the jobs have to be scheduled on contiguous machines by calling the items jobs (where the processing time corresponds to the height) and calling the strips clusters. Hence, in this section, we will speak of jobs and clusters for all the considered problem settings. In this spirit, we define the work of a job j as $\text{work}(j) := p(j) \cdot m(j)$ and the work of a set of jobs

$\mathcal{J}'$ as $\text{work}(\mathcal{J}') := \sum_{j \in \mathcal{J}'} \text{work}(j)$. In the following, let OPT be the height of an optimal schedule on N clusters for a given instance I.

The basic idea of the algorithm can be summarized as follows. Instead of scheduling the jobs on N clusters, we first schedule them on only two clusters C_1 and C_2, such that C_1 contains almost all the jobs, while C_2 contains some residual jobs and has a makespan bounded by OPT. In a second step, we distribute the scheduled jobs to the N clusters by cutting the schedule on C_1 into pieces of height at most 2OPT. This simple partitioning technique works, if $N \geqslant 3$. In the case that $N = 2$ we need to make stronger assumptions about the given schedule on C_1 and C_2.

8.2.1 The case $N > 2$

If $N > 2$, we can partition the schedule as described in the proof of the following lemma.

Lemma 8.5. *Let $N > 2$ and C_1 and C_2 be two clusters with m machines each. Given a schedule on C_1 and C_2 such that the makespan of C_1 is given by $T \leqslant (N + \lfloor N/3 \rfloor)\text{OPT}$ and C_2 has a makespan of at most* OPT*, we can find a schedule on N clusters with makespan at most* 2OPT *in at most $\mathcal{O}(n)$ additional steps.*

Proof. In the following, we describe how to distribute the given schedule to N new clusters. The partitioning algorithm distinguishes three cases: $N = 3i$, $N = 3i + 1$ and $N = 3i + 2$ for some $i \in \mathbb{N}_{\geqslant 1}$. When speaking of a schedule, the processing time is always displayed on the vertical axis while the machines are next to each other on the horizontal axis, see Figure 8.1.

In the following distributing algorithm, we draw horizontal lines at each multiple of $2T_A$, where $T_A \leqslant \text{OPT}$ is a value which depends on the makespan of the given schedule on C_1 and will be specified dependent on N later. We say a job j is cut by a horizontal line at $i \cdot 2T_A$ if it starts before and ends after it, i.e. if $\sigma(j) < i \cdot 2T_A < \sigma(j) + p(j)$. Let $i \in \mathbb{N}$ and consider the jobs which start at or after $2iT_A$ and end at or before $2(i+1)T_A$. We remove these jobs from C_1 and schedule them on a new cluster while maintaining their relative position. We say these new clusters have type A. Obviously the makespan of this cluster is bounded by $2T_A \leqslant 2\text{OPT}$. Next, consider the set of jobs cut by the horizontal line at $2iT_A$. These jobs have a

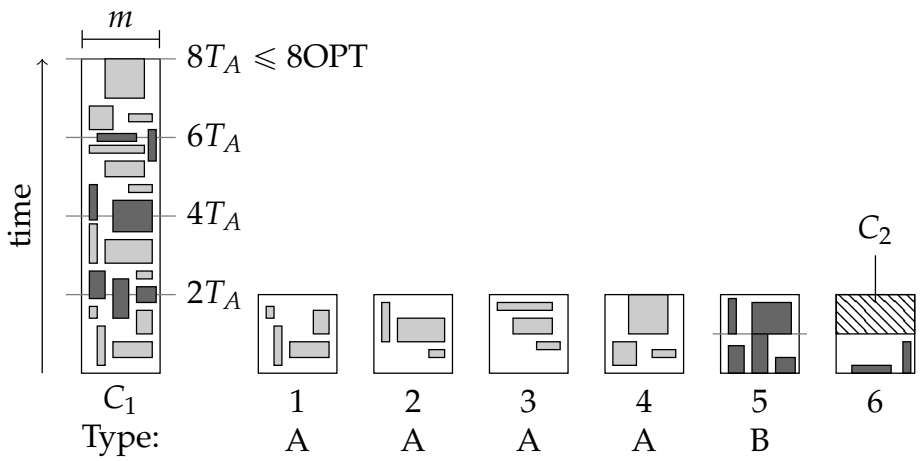


Figure 8.1. An example for $N = 3i$ and $i = 2$. The schedule generated on C_1 can be seen on the left followed by its partition onto the six clusters. The schedule on C_1 has a height of at most $(N + \lfloor N/3 \rfloor)\text{OPT} \leqslant 8\text{OPT}$. We get four clusters of type A, namely clusters 1 to 4, and one cluster of type B, namely cluster 5 which contains the jobs cut by the two horizontal lines at $2T_A$ and $4T_A$. Cluster 6 contains the jobs cut by the horizontal line at $6T_A$ and the jobs from cluster C_2 that remain their relative position.

processing time of at most $p_{\max} \leqslant \text{OPT}$ and can be scheduled at the same time without violating any constraint, since they are already scheduled next to each other. We schedule two of these sets of jobs in a new cluster with makespan $2p_{\max} \leqslant 2\text{OPT}$ by letting the first set start at 0 and the second start at $p_{\max}$. We say, these clusters have type B.

Case 1: $N = 3i$. In this case, the schedule on C_1 has a makespan of $T \leqslant (N + \lfloor N/3 \rfloor)\text{OPT} = 4i\text{OPT}$ and we define $T_A := T/(4i) \leqslant \text{OPT}$. We partition the given schedule as described above. Since it has a height of $4iT_A$, we get $2i$ clusters of type A each with makespan at most $2T_A \leqslant 2\text{OPT}$, see Figure 8.1. There are $4iT_A/(2T_A) - 1 = 2i - 1$ lines at multiples of $2T_A$ that have the possibility to cut jobs. Hence, we get $\left\lfloor \frac{2i-1}{2} \right\rfloor = i - 1$ clusters of type B since each cluster of type B contains two sets of jobs cut by such a horizontal line. The jobs intersecting the last line can be scheduled on one new cluster with makespan $p_{\max} \leqslant \text{OPT}$. On this last cluster after the point in time $p_{\max}$, we schedule the jobs from the cluster

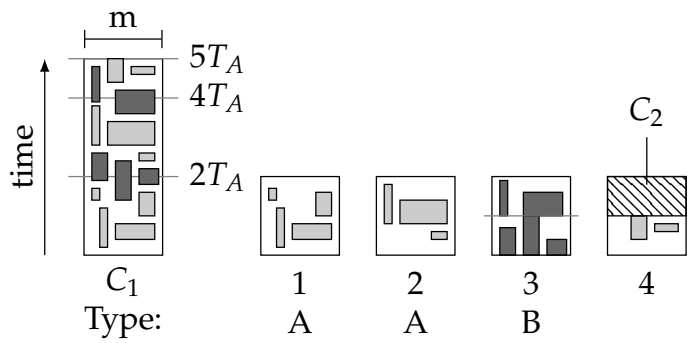


Figure 8.2. An example for $N = 3i + 1$ and $i = 1$. The makespan of the schedule on C_1 is bounded by $(N + \lfloor N/3 \rfloor)\text{OPT} = 5\text{OPT}$. Hence, we get two clusters of type A, namely cluster 1 and 2, and one cluster of type B, see cluster 3. It contains the jobs cut by the horizontal lines at $2T_A$ and $4T_A$. Cluster 4 contains the jobs which are completely scheduled between $4T_A$ and $5T_A$ as well as the jobs from cluster C_2.

C_2. Remember, the schedule on C_2 has a makespan of at most OPT and, hence, the makespan of this last cluster is bounded by 2OPT as well. In total, we have partitioned the schedule into $2i + i - 1 + 1 = 3i = N$ clusters each with makespan at most 2OPT.

Case 2: $N = 3i + 1$. In this case, the makespan of C_1 is bounded by $T \leqslant (N + \lfloor N/3 \rfloor)\text{OPT} = (4i + 1)\text{OPT}$ and we define $T_A := T/(4i + 1) \leqslant \text{OPT}$. There are $\lceil (4i + 1)/2 \rceil - 1 = 2i$ multiples of $2T_A$ smaller than $(4i + 1)T_A$, see Figure 8.2. Above the last multiple of $2T_A$ smaller than $(4i + 1)T_A$ namely $4iT_A$, the schedule has a height of at most $T_A \leqslant \text{OPT}$ left. Hence using the above-described partitioning technique, we generate $2i$ clusters of type A. The jobs intersecting the $2i$ multiples of $2T_A$ can be placed into i clusters of type B. We have left the jobs above $4iT_A$, which can be scheduled in a new cluster with makespan $T_A \leqslant \text{OPT}$. Last, we place the jobs from cluster C_2 on top of the schedule in the new cluster, such that it has a makespan of at most $T_A + \text{OPT} \leqslant 2\text{OPT}$ in total. Altogether, we have distributed the given schedule on $2i + i + 1 = 3i + 1 = N$ clusters each with makespan at most 2OPT.

Case 3: $N = 3i + 2$. In this case, the makespan on C_1 is bounded by $T \leqslant (N + \lfloor N/3 \rfloor)\text{OPT} = (4i + 2)\text{OPT}$ and we define $T_A := T/(4i + 2) \leqslant \text{OPT}$.

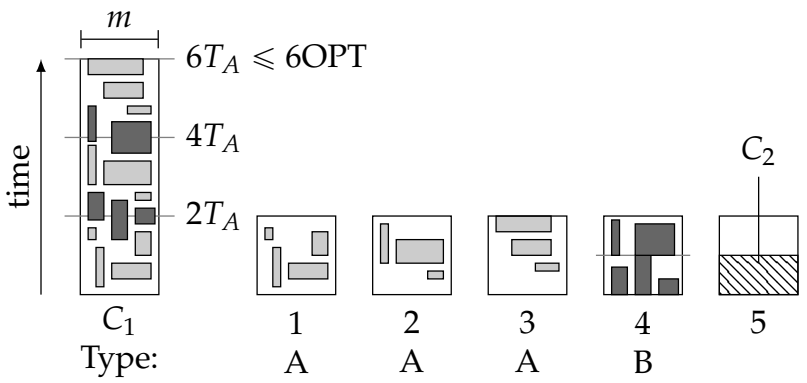


Figure 8.3. An example for $N = 3i + 2$ and $i = 1$. The total height of the schedule on C_1 is bounded by $(N + \lfloor N/3 \rfloor)\text{OPT} = 6\text{OPT}$. We get three clusters of type A namely 1, 2, and 3. Furthermore, we get one cluster of type B namely cluster number 4 which contains the jobs cut by the horizontal lines at $2T_A$ and $4T_A$. The cluster C_2 builds its own cluster, see cluster 5.

Thus, there are $(4i + 2)/2 - 1 = 2i$ vertical lines at the multiples of $2T_A$, which are strictly larger than 0 and strictly smaller than $(4i + 2)T_A$, see Figure 8.3. As a consequence, we construct $2i + 1$ clusters of type A and i clusters of type B. The cluster C_2 defines one additional cluster of this new schedule. In total, we have a schedule on $2i + 1 + i + 1 = N$ clusters with makespan bounded by 2OPT.

This distribution can be made in $\mathcal{O}(n)$ steps since we have to relocate each job at most once. This concludes the proof of Lemma 8.5. □

8.2.2 The case $N = 2$

To find a distribution for this case, we need to make a stronger assumption to the given solution. Namely, we assume that the second cluster C_2 has just $\lfloor m/6 \rfloor$ machines. As a consequence, the total work of the jobs contained on C_2 is bounded by $1/6 \cdot m\text{OPT}$.

Lemma 8.6. *Let be $N = 2$ as well as C_1 and C_2 be two clusters with m and $\lfloor m/6 \rfloor$ machines respectively. Furthermore, let* `Alg` *be an algorithm that finds for the single cluster variant a schedule or packing with height at most $2 \cdot \max\{\text{work}(\mathcal{J})/m, p_{\max}\}$ in $\text{Op}(\texttt{Alg}(\mathcal{J}, m, N))$ operations.*

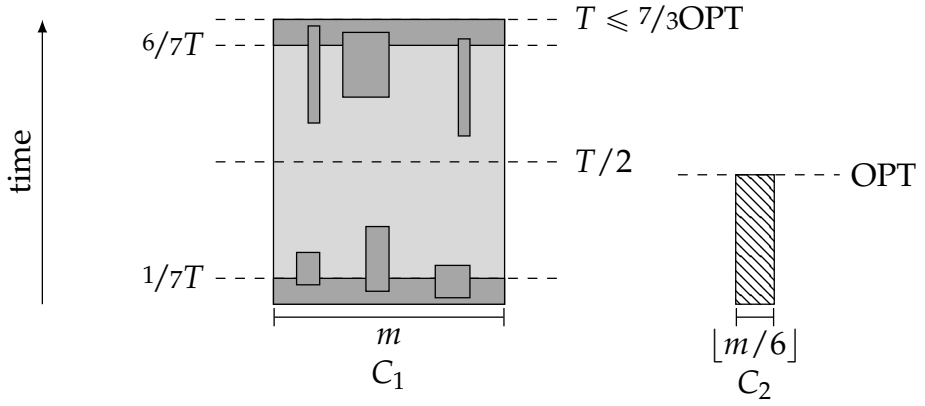


Figure 8.4. An example for a schedule on C_1 and C_2 for the case that $N = 2$. Cluster C_1 has m machines and a makespan of at most $\frac{7}{3}$OPT, while cluster C_2 has $\lfloor m/6 \rfloor$ machines and a makespan of at most OPT. The dark gray areas on the bottom represent the jobs inside the set $\mathcal{J}_1$ while the dark gray areas on the top represent the jobs contained in the set $\mathcal{J}_2$.

Given a schedule on C_1 and C_2 such that the makespan on C_1 is bounded by $T \leqslant 7/3\text{OPT}$ while bounded by OPT *on C_2, we can distribute the schedule on $N = 2$ new clusters with makespan* 2OPT *in* $\text{Op}(\texttt{Alg}(\mathcal{J}, m, N)) + \mathcal{O}(n)$ *additional operations.*

Proof. We denote by $\mathcal{J}(C_1)$ the set of jobs scheduled on C_1 and by $\mathcal{J}(C_2)$ the set of jobs scheduled on C_2. The total work of the jobs in $\mathcal{J}$ is bounded by $2\text{OPT}m$ and, hence, $\text{work}(\mathcal{J})/(2m) \leqslant \text{OPT}$. In the following, we assume that $T > 2p_{\max}$ since otherwise we have $T \leqslant 2\text{OPT}$ and do not need to reorder the schedule any further. Note that since $T \leqslant 7/3\text{OPT}$ it holds that $3/7T \leqslant \text{OPT}$ and $6/7T \leqslant 2\text{OPT}$.

In the first step, we draw horizontal lines L_1 and L_2 at $1/7T$ and at $6/7T$ respectively. We use these lines to define two sets of jobs $\mathcal{J}_1$ and $\mathcal{J}_2$. The set $\mathcal{J}_1$ contains all jobs starting before L_1 and $\mathcal{J}_2$ contains all jobs ending after L_2, see Figure 8.4. Note that the sets $\mathcal{J}_1$ and $\mathcal{J}_2$ are disjoint since $p_{\max} \leqslant T/2$ and therefore $1/7T + p_{\max} \leqslant 6/7T$ and hence a job starting before L_1 cannot end after L_2. We distinguish two cases about $\text{work}(\mathcal{J}_1)$ and $\text{work}(\mathcal{J}_2)$.

Case 1: $\text{work}(\mathcal{J}_1) \leqslant 5/6 \cdot \text{work}(\mathcal{J})/2$ *or* $\text{work}(\mathcal{J}_2) \leqslant 5/6 \cdot \text{work}(\mathcal{J})/2$.

Let w.l.o.g. $\text{work}(\mathcal{J}_2) \leqslant 5/6 \cdot \text{work}(\mathcal{J})/2 \leqslant 5/6 \cdot m\text{OPT}$. We remove all jobs in $\mathcal{J}_2$ from the cluster C_1. As a result this cluster has a makespan of $6/7T \leqslant 2\text{OPT}$. On the other hand, the total work of the jobs contained in C_2 combined with the jobs in $\mathcal{J}_2$ is bounded by $\text{work}(\mathcal{J}_2 \cup \mathcal{J}(C_2)) \leqslant m\text{OPT}$. Therefore, we can use the algorithm `Alg` to find a schedule with makespan at most $2\max\{p_{\max}, \text{work}(\mathcal{J}(C_2) \cup \mathcal{J}_2)/m\} \leqslant 2\text{OPT}$. Hence, we can find a schedule on two clusters in at most $\text{Op}(\texttt{Alg2}(\mathcal{J}, m, N)) + \mathcal{O}(n \cdot f(\varepsilon))$ for this case.

Case 2: $\text{work}(\mathcal{J}_1) > 5/6 \cdot \text{work}(\mathcal{J})/2$ *and* $\text{work}(\mathcal{J}_2) > 5/6 \cdot \text{work}(\mathcal{J})/2$. Consider the set of jobs $\mathcal{J}_3$ scheduled on C_1 but not contained in $\mathcal{J}_1$ or $\mathcal{J}_2$. Note that $\mathcal{J} = \mathcal{J}_1 \dot\cup \mathcal{J}_2 \dot\cup \mathcal{J}_3 \dot\cup \mathcal{J}(C_2)$. Since the total work of the jobs is bounded by $\text{work}(\mathcal{J}) \leqslant 2m\text{OPT}$ it holds that $\text{work}(\mathcal{J}_3 \cup \mathcal{J}(C_2)) \leqslant \text{work}(\mathcal{J}) - \text{work}(\mathcal{J}_1) - \text{work}(\mathcal{J}_2) < 1/3 \cdot m\text{OPT}$. We define $\mathcal{J}_4$ as the set of jobs ending at or before L_1 and $\mathcal{J}_5$ as the set of jobs starting at or after L_2. Both sets have a total work of at most $1/7 \cdot m \cdot T \leqslant 1/7 \cdot m \cdot 7/3 \cdot \text{OPT} = 1/3 \cdot m\text{OPT}$ each and therefore $\text{work}(\mathcal{J}_3 \cup \mathcal{J}_4 \cup \mathcal{J}_5 \cup \mathcal{J}(C_2)) \leqslant m\text{OPT}$. As a consequence, we can schedule them with the algorithm `Alg` to find a schedule on one cluster with makespan at most $2\max\{p_{\max}, \text{work}(\mathcal{J}_3 \cup \mathcal{J}_4 \cup \mathcal{J}_5 \cup \mathcal{J}(C_2))/m\} \leqslant 2\text{OPT}$.

To this point, we have scheduled all jobs except the ones cut by the line L_1 and the jobs cut by the line L_2. We schedule them in the second cluster by starting all the jobs cut by the first line at start point 0 and the second set of jobs at the start point $p_{\max} \leqslant \text{OPT}$. Note that the partition into the sets $\mathcal{J}_1, \ldots, \mathcal{J}_5$ can be done in $\mathcal{O}(n)$ and hence the partitioning step is dominated by the running time of the algorithm `Alg`. Hence, we can bound the running time of the algorithm by $\text{Op}(\texttt{Alg}(\mathcal{J}, m, N)) + \mathcal{O}(n)$ in this case.

Hence in both cases, we can find a schedule on two clusters each with makespan bounded by 2OPT which concludes the proof of Lemma 8.6. □

8.2.3 Proof of Theorem 8.1

To proof the Theorems 8.1, we need to find the schedule on C_1 and C_2 with the conditions from Lemma 8.5 and 8.6 in $\mathcal{O}(n)$. We present algorithms which can find such a schedule in Sections 8.3, 8.5, and 8.6 respectively.

More precisely, by Lemma 8.7 in Section 8.3, by Lemma 8.10 in Section 8.5 and by Lemma 8.17 in Section 8.6, we can find a schedule on two clusters C_1 and C_2 with m and $\lfloor m/c \rfloor$ machines, such that the makespan on C_2 is bounded by $p_{\max}$ and the makespan on C_1 is bounded by $(1+\varepsilon)\mathrm{OPT}_{\text{single}}$, where $\mathrm{OPT}_{\text{single}}$ is the optimum of a schedule on a single cluster with m machines. Both algorithms have a running time that is bounded by $\mathcal{O}(n) \cdot \mathcal{O}_{\varepsilon,c}(1)$. Note that since there is a schedule with makespan OPT on N clusters, there exists a schedule on one cluster with makespan at most $N \cdot \mathrm{OPT}$. Hence we can bound the optimal makespan for the single cluster version by $\mathrm{OPT}_{\text{single}} \leqslant N \cdot \mathrm{OPT}$.

For the case $N > 2$, we call the algorithm from Lemma 8.7, Lemma 8.10 or Lemma 8.17 respectively with $\varepsilon \leqslant \lfloor N/3 \rfloor / N$ and $c = 1$. As a consequence, the schedule on C_1 has a makespan bounded by

$$(1 + \lfloor N/3 \rfloor / N)\mathrm{OPT}_{\text{single}} \leqslant (N + \lfloor N/3 \rfloor)\mathrm{OPT},$$

while the makespan of C_2 is bounded by $p_{\max} \leqslant \mathrm{OPT}$. This schedule meets the requirements of Lemma 8.5. Therefore, we can use the partitioning technique to partition the given schedule onto the N clusters, such that each cluster has a makespan bounded by 2. Note that $\lfloor N/3 \rfloor / N = 1/5$ for $N = 5$ and $\lfloor N/3 \rfloor / N \in [1/4, 1/3]$ for each other $N > 2$. Since the algorithms need $1/\varepsilon$ to be integral, we call them with $\varepsilon = 1/5$ for $N = 5$, $\varepsilon = 1/3$ for each N that is dividable by 3, and with $\varepsilon = 1/4$ for all the other $N > 2$. Since the algorithms from Lemmas 8.7, 8.10 and 8.17 have a running time of the form $\mathcal{O}(n) \cdot \mathcal{O}_{\varepsilon,c}(n)$ and we call the corresponding one with a constant $\varepsilon \geqslant 1/5$ and $c = 1$, the running time is bounded by $\mathcal{O}(n)$, where the large constant corresponding to the $\mathcal{O}_{\varepsilon,c}(1)$ term is hidden in the $\mathcal{O}$-notation.

For the case of $N = 2$, we call the algorithm from Lemma 8.7 or Lemma 8.10 with $\varepsilon = 1/6$ and $c = 6$. As a consequence the schedule on C_1 is bounded by $(1+\varepsilon) \cdot 2 \cdot \mathrm{OPT} \leqslant 7/3 \cdot \mathrm{OPT}$ while it is bounded by OPT on C_2. Furthermore, C_2 has only $\lfloor m/6 \rfloor$ machines. Hence, the constructed schedule has the properties required by Lemma 8.6. Since we call the algorithms from Lemmas 8.7 and 8.10 with constants for ε and c, the running time is bounded by $\mathcal{O}(n)$. However, to apply the algorithm from Lemma 8.6, we need algorithms which can find schedules with makespan bounded by $2 \cdot \max\{p_{\max}, \mathrm{work}(\mathcal{J})/2\}$. For MCS, we use

the List-Scheduling algorithm by Garay and Graham [34], which was optimized by Turek et al. [105] to have a running time of $\mathcal{O}(n\log(n))$. On the other hand, for MSP we use Steinberg's Algorithm [100] which has a running time that is bounded by $\mathcal{O}(n\log^2(n)/\log(\log(n)))$. Both running times dominate the running time of $\mathcal{O}(n)$ which is needed to find the schedule on C_1 and C_2. This concludes the proof of the Theorems 8.1.

Indeed, the partitioning technique described in the proof of Lemma 8.5 can be used for any problem setting where there is an AEPTAS with approximation ratio $(1+\varepsilon)\mathrm{OPT} + p_{\max}$ for the single cluster version. In this context, $p_{\max}$ is the largest occurring size in the minimization dimension, e.i., it stands for $h_{\max}$ in the MSP problem.

Note that, instead of the AEPTAS, first we can try to use any heuristic or other (fast) approximation algorithm: Given a schedule by any heuristic, remove all the jobs that end after the point in time where the last job starts and place them on the cluster C_2 by starting them all at the same time. The schedule on C_2 obviously has a makespan bounded by $p_{\max} \leqslant \mathrm{OPT}$. Next, check whether the residual schedule on C_1 has a makespan of at most $(1+\lfloor N/3\rfloor/N)N\mathrm{OPT}$. For example, compare the makespan T on C_1 to the lower bound on the optimal makespan $L := \max\{p_{\max}, \mathrm{work}(\mathcal{J})/m, p(\mathcal{J}_{>m/2})\}$, where $\mathcal{J}_{>m/2}$ is the set of all jobs with machine requirement larger than $m/2$. If the makespan T is small enough, i.e., if $T \leqslant (1+\lfloor N/3\rfloor/N)L$ use the partitioning technique to find a 2-approximation. Otherwise, use the corresponding AEPTAS presented in Section 8.3 or 2.2.

8.3 An AEPTAS for Parallel Task Scheduling

In this section, we will present an *AEPTAS* for Parallel Task Scheduling (PTS) with an approximation ratio $(1+\varepsilon)\mathrm{OPT} + p_{\max}$ and running time $\mathcal{O}(n\log(1/\varepsilon)) + \mathcal{O}_\varepsilon(1)$, where OPT is the optimal makepsan for the given PTS instance. More precisely, we present an algorithm to find a schedule on the two clusters C_1 and C_2 such that the schedule on C_1 is bounded by $(1+\varepsilon)\mathrm{OPT}$ and on C_2 by $p_{\max}$. Obviously when stacking the jobs from cluster C_2 on top of C_1, we get the required AEPTAS.

Lemma 8.7. *There exists an algorithm, that finds for each instance $I = (\mathcal{J}, m)$ of the Parallel Task Scheduling problem a schedule on two clusters C_1 and C_2 with m and $\lfloor m/c \rfloor \leqslant m$ machines, such that the makespan on C_1 is bounded by $(1+\varepsilon)\mathrm{OPT}$ and on C_2 by p_{max}. This algorithm has a running time that is bounded by $\mathcal{O}(n \log(1/\varepsilon)) + 1/\varepsilon^{c \cdot 1/\varepsilon^{\mathcal{O}(1/\varepsilon)}}$*

The algorithm in this chapter is inspired by the algorithm in [50] but contains some improvements. Furthermore, note the fact that in the following algorithm the processing times of the jobs do not have to be integral. Instead, we will discretize them by rounding.

The algorithm works roughly in the following way. The set of jobs is partitioned into large, medium, and small jobs, depending on their processing times. The medium jobs have a small total work and therefore can be scheduled at the end of the schedule using a 3-approximation algorithm without doing too much harm. The large jobs are partitioned into two sets: wide jobs and narrow jobs depending on their machine requirement. There are few large wide jobs which makes it possible to guess their starting times. To place the narrow jobs, we introduce a linear program to distribute them to starting positions. We guess the non-zero components of an optimal solution to this linear program and for each of them we guess one of the constant number of values it can take. We place the narrow jobs greedily inside this solution, leaving a constant number of jobs, that cannot be placed. The total number of required machines of these non-placed jobs is bounded by $\lfloor m/c \rfloor$. We place these jobs into the extra cluster C_2. The small jobs are scheduled with a linear program. An overview of the algorithm can be found in Section 8.3.5.

We use an improved rounding strategy for large jobs compared to [50], which enables us to improve the running time. Further, we present a different linear programming approach to schedule the narrow tall jobs.

8.3.1 Simplify

Let an instance $I = (\mathcal{J}, m)$ and an $\varepsilon \in (0, 1]$ be given such that $1/\varepsilon \in \mathbb{N}$. Note that the value $T := \max\{p_{\max}, W(\mathcal{J})/m\}$ is a lower bound on the makespan of the schedule. On the other hand, we know by Turek et al. [105] that $2\max\{p_{\max}, W(\mathcal{J})/m\}$ is an upper bound on the optimal

makespan. We can find T in $\mathcal{O}(n)$.

Let δ and μ be values dependent on ε. We partition the set of jobs $\mathcal{J}$ into small $\mathcal{J}_S := \{j \in \mathcal{J} | p_j \leqslant \mu T\}$, medium $\mathcal{J}_M := \{j \in \mathcal{J} | \mu T < p_j < \delta T\}$, and large ones $\mathcal{J}_L := \{j \in \mathcal{J} | \delta T \leqslant p_j\}$. Consider the sequence $\sigma_0 = \varepsilon$, $\sigma_{i+1} = \sigma_i \varepsilon^3$. By the pigeonhole principle there exists an $i \in \{0, \ldots, 1/\varepsilon - 1\}$ such that $W(\mathcal{J}_M) \leqslant \varepsilon m \mathrm{OPT}$, when defining $\delta := \sigma_i$ and $\mu := \sigma_{i+1}$. We can find these values for δ and μ in $\mathcal{O}(n + 1/\varepsilon)$. Note that $\mu = \varepsilon^3 \delta \geqslant \varepsilon^{3/\varepsilon+3}$.

Resulting in a loss of at most εT in the approximation ratio, we can assume that the smallest processing time is at least $\varepsilon T/n$ since adding $\varepsilon T/n$ to each processing time adds at most $n \cdot \varepsilon T/n = \varepsilon T$ to the total makespan. Therefore, the largest l such that $p_j \in \{\varepsilon^l T, \varepsilon^{l-1} T\}$ is bounded by $\mathcal{O}(\log(n))$ and we know $\delta \geqslant \min\{\varepsilon/n, \varepsilon^{3/\varepsilon}\}$. We round the processing times of the large and medium jobs by using Lemma 3.2. This rounding can be done in $\mathcal{O}(n)$. Afterward, there are at most $1/\varepsilon^2$ different processing times between $\varepsilon^l T$ and $\varepsilon^{l-1} T$ for each $l \in \{1, \ldots, 3/\varepsilon + 3\}$. Therefore, the number of different processing times of large jobs is bounded by $1/\varepsilon^2 \cdot 3/\varepsilon = 3/\varepsilon^3$ since $\delta \geqslant \varepsilon^{3/\varepsilon}$. Further, the number of different processing times for medium jobs is bounded by $3/\varepsilon^2$ since the medium jobs have processing times in $(\mu = \varepsilon^3 \delta, \delta)$. Additionally, there are at most $1/\varepsilon\delta$ possible starting points for the large jobs. We denote the set of starting points for large jobs as $\mathcal{S}$ and the set of their processing times as P_L. After this step, we will only consider the rounded processing times and will denote them as p_j for each job $j \in \mathcal{J}$.

8.3.2 Large Jobs

We classify the large jobs by their width. More precisely we introduce a constant $\alpha := \varepsilon\delta/(14c)$ and say a job $j \in \mathcal{J}_L$ is wide if it uses at least αm machines. We denote the set of large wide jobs by $\mathcal{J}_{L,W}$. Note that large wide jobs have a processing time larger than δT and need at least $\lceil \alpha m \rceil$ machines while the total work of all jobs in $\mathcal{J}$ is bounded by mT. Hence, the total number of them is bounded by $1/(\delta\alpha)$. Therefore, there are at most $S^{1/(\delta\alpha)}$ possibilities to schedule the jobs in $\mathcal{J}_{L,W}$. In the algorithm, we will try each of these options.

Observation 8.8. If $\alpha m < 1$ all the large jobs are wide since each job needs

at least one machine to be processed.

In the next step, we deal with the large narrow jobs $\mathcal{J}_{L,N} := \mathcal{J}_L \backslash \mathcal{J}_{L,W}$. Note that the jobs in $\mathcal{J}_{L,N}$ have a machine requirement of at most $\lfloor \alpha m \rfloor$. Consider an optimal schedule $S = (\sigma, \rho)$, where we have rounded the processing times of the jobs as described in Lemma 3.2. For the schedule S and each starting time $s \in \mathcal{S}$, let m_s be the number of machines used by jobs in $\mathcal{J}_{L,N}$ that are processed (not only started) at that time, i.e., we define $m_s := \sum_{j \in \mathcal{J}_{L,N,s}} q_j$ where $\mathcal{J}_{L,N,s}$ is the set of jobs $j \in \mathcal{J}_{L,N}$, which have both a start point $s_j \leqslant s$ and an endpoint $e_j := s_j + p_j > s$. Note that jobs ending at s, i.e., jobs with $e_j = s_j + p_j = s$, are not part of the set $\mathcal{J}_{L,N,s}$.

For each processing time $p \in P_L$ let $q(p)$ be the total number of machines used by jobs with this processing time, i.e $q(p) := \sum_{j \in J_{L,N}, p_j = p} q_j$. Consider the following linear program LP_{large}:

$$\sum_{p \in P_L} \sum_{\substack{s \in \mathcal{S}(p)_{\leqslant s'}, \\ s+p > s'}} x_{s,p} = m_s \qquad \forall s' \in \mathcal{S} \tag{8.1}$$

$$\sum_{s \in \mathcal{S}(p)} x_{s,p} = q(p) \qquad \forall p \in P_L \tag{8.2}$$

$$x_{s,p} \geqslant 0 \qquad \forall p \in P_L, s \in \mathcal{S}(p). \tag{8.3}$$

The variable $x_{s,p}$ defines for each start point $s \in \mathcal{S}$ and each processing time $p \in P$ how many machines are used by jobs with processing time p starting at s. The first inequality ensures that the number of machines required by jobs scheduled at a start point s, i.e., jobs from the set $\mathcal{J}_{L,N,s}$, equals the number of used machines in the considered optimal schedule. The second inequality ensures that for each processing time all the jobs are scheduled. Given the considered optimal solution, we generate a solution to this linear program by counting for each starting time $s \in \mathcal{S}$ and each processing time $p \in P_L$ how many machines are used by jobs with processing time p starting at s. This linear program has $|\mathcal{S}| + |P_L|$ conditions and $|\mathcal{S}||P_L|$ variables. Since we have $|\mathcal{S}| + |P_L|$ conditions, there are at most $|\mathcal{S}| + |P_L|$ non zero components in a basic solution and for each $p \in P_L$ there has to be at least one non zero component.

In the algorithm, we guess, (i.e., we try out all the possibilities) which

variables are non zero variables in the basic solution. There are at most $\mathcal{O}((|\mathcal{S}||P_L|)^{|\mathcal{S}|+|P_L|})$ options. We cannot guess the exact values of the variables $x_{s,p}$ in polynomial time. Instead, we guess for each non zero variable $x_{s,p}$ the smallest multiple of αm that is larger than the value of $x_{s,p}$ in the basic solution. This can be done in $\mathcal{O}(1/\alpha^{|\mathcal{S}|+|P_L|})$. So to find a schedule for the large jobs $\mathcal{J}_L$, we use at most $\mathcal{O}(|\mathcal{S}|^{|\mathcal{J}_{L,W}|} \cdot (|\mathcal{S}||P_L|/\alpha)^{|\mathcal{S}|+|P_L|})$ guesses in total.

Note that this optimistic guessing, i.e., using the rounded up values for m_s, on the one hand ensures that all the narrow large jobs can be scheduled but on the other hand can cause violations to the machine constraints. To prevent this machine violation, the algorithm tests for each guess whether the job condition (8.2) is fulfilled for each processing time. If this is the case, each value of a non-zero component is reduced by αm. For these down-sized values, the algorithm tests the machine constraint (8.1) for each starting point $s \in S$. Note that the validation whether the constraints are fulfilled is possible in $\mathcal{O}((|P_L| + |\mathcal{S}|)^2)$ since for each of the $(|P_L| + |\mathcal{S}|)$ constraints, we have to add at most $(|P_L| + |\mathcal{S}|)$ values. If both conditions are fulfilled, the algorithm tries to schedule the small jobs, see Subsection 8.3.3. If the small jobs can be scheduled the guess was feasible.

The actual narrow large jobs from the set $\mathcal{J}_{L,N}$ are scheduled only once in the final phase of the algorithm. When scheduling the jobs in $\mathcal{J}_{L,N}$, we use the reduced guessed values. We greedily fill the jobs into the guessed starting positions $x_{s,p}$, while slicing jobs vertical if they do not fit totally at that starting position (i.e., if the total number of machines required by jobs with processing time p starting at s is larger than $x_{s,p}$ when adding the machine requirement of the currently considered job) and placing the rest of the job at the next starting position for the processing time p. We schedule the jobs which cannot be placed at the starting points defined by the values of $x_{s,p}$ (because we reduced these values) in the second cluster C_2, all staring at the beginning. The total width of these jobs placed in cluster C_2 is bounded by $\alpha m \cdot (|S| + |P_L|)$ since there are at most $|S| + |P_L|$ non zero components and before the reduction by αm all the jobs could be scheduled because the job constraint (8.2) was fulfilled.

In the described placement of the narrow large jobs, we have introduced at most one fractional job for each non zero variable and it has a width of at most $\lfloor \alpha m \rfloor$. We remove all these fractional jobs and place them next to

the jobs which did not fit. As a consequence the total machine requirement of the jobs scheduled in C_2 is bounded by $2(|S| + |P_L|)\alpha m \leqslant 2(3/\varepsilon\delta + 3/\varepsilon^3) \cdot \varepsilon\delta/(14c) \cdot m \leqslant \lfloor m/c \rfloor$ if $\alpha m \geqslant 1$. Otherwise all the large jobs are wide, and no job is scheduled on C_2. We need at most $\mathcal{O}(n + |\mathcal{S}| + |P_L|)$ operations to place the narrow large jobs inside the guessed LP-solution.

8.3.3 Small Jobs

We define a layer as the horizontal strip between two consecutive starting points in $\mathcal{S}$ and say layer l is the layer between $l\varepsilon\delta T$ and $(l+1)\varepsilon\delta T$. Note that during the processing time of a layer l the machine requirement of large jobs will not change since large jobs start and end at multiples of $\varepsilon\delta T$. Let m_l be the number of machines left for small jobs in layer l. Note that this number is fixed by the guesses for the large jobs.

We will partition the small jobs into wide and narrow jobs. A small job is wide if it requires at least εm machines and narrow otherwise. Let $\mathcal{J}_{S,W}$ be the set of small wide jobs and $\mathcal{J}_{S,N}$ be the set of small narrow jobs. We will round the machine requirements of the wide jobs using linear grouping. By Lemma 3.3, we can round the machine requirements of the wide jobs to $\mathcal{O}(1/\varepsilon^2)$ sizes in $\mathcal{O}(|\mathcal{J}_{S,N}| \log(1/\varepsilon)) \leqslant \mathcal{O}(n \log(1/\varepsilon))$ operations by extending the schedule by at most $\varepsilon \cdot \text{work}(\mathcal{J}_{S,N})/m \leqslant \varepsilon T$. We denote the resulting set of rounded jobs as $\bar{\mathcal{J}}_{S,W}$.

We say a configuration of wide jobs is a multiset of wide jobs $C := \{a_{j,C} : j | j \in \bar{\mathcal{J}}_{S,W}\}$. We say a configuration C requires at most q machines, if $\sum_{j \in \bar{\mathcal{J}}_{S,W}} a_{j,C} q_j \leqslant q$ and define $q(C) := \sum_{j \in \bar{\mathcal{J}}_{S,W}} a_{j,C} q_j$. Let $\mathcal{C}_q$ be the set of configurations with machine requirement at most q, i.e., $\mathcal{C}_q := \{C \in \mathcal{C}\} | q(C) \leqslant q\}$.

Consider the following linear program LP_{small}.

$$\sum_{C \in \mathcal{C}_m} x_{C,|S|+1} = \varepsilon T \tag{8.4}$$

$$\sum_{C \in \mathcal{C}_{m_l}} x_{C,l} = \varepsilon\delta T \qquad \forall l = 1, \ldots, |S| \tag{8.5}$$

$$\sum_{l=1}^{|S|} \sum_{C \in \mathcal{C}_{m_l}} x_{C,l} a_{j,C} = p_j \qquad \forall j \in \bar{\mathcal{J}}_{S,W} \tag{8.6}$$

$$x_{C,l} \geqslant 0 \qquad \forall l = 1, \ldots, |S|, C \in \mathcal{C}_{m_l} \tag{8.7}$$

The variable $x_{C,l}$ defines the processing time of the configuration C in layer l. The condition (8.5) ensures that we do not give a too large processing time to the configurations used in Layer l, while condition (8.6) ensures that the processing time of each job is covered. Condition (8.4) is added to place the rounded jobs inside the extra box. This linear program has $|S| + |\bar{\mathcal{J}}_{S,W}|$ conditions and at most $|S||\mathcal{C}_m|$ variables. If the values m_l are derived from an optimal solution (or are larger than in the corresponding optimal solution), the linear program above has a solution.

By Lemma 3.9, we can solve the relaxed version of LP_{small}, where we increase the right hand side of condition (8.4) and (8.5) by the factor $(1+\varepsilon)$, in at most $\mathcal{O}((|S||\bar{\mathcal{J}}_{S,W}|(\ln(|\bar{\mathcal{J}}_{S,W}|) + \varepsilon^{-4}))((|S| + |\bar{\mathcal{J}}_{S,W}|)^{1.5356} + (\log(1/\varepsilon))^3/\varepsilon^4)) \leqslant \mathcal{O}(1/\varepsilon^{12}\delta^3)$ operations. This solution uses at most $|S| + |\bar{\mathcal{J}}_{S,W}|$ non-zero components.

We will find a schedule of the jobs $\mathcal{J}_{S,W}$, by placing the configurations into the corresponding layers and greedily filling the jobs into the configurations, see Figure 8.5. To ensure that each job can be scheduled integrally, we extend each configuration by μT, which is the tallest height a small job can have. Since there are at most $|S| + |\bar{\mathcal{J}}_{S,W}|$ configurations we extend the schedule by at most $(|S| + |\bar{\mathcal{J}}_{S,W}|)\mu T \leqslant (1/\varepsilon\delta + 1/\varepsilon^2)\delta\varepsilon^3 T \leqslant 2\varepsilon^2 T$. Note that after this extension the size defining job, which might has been cut for the analysis, can be scheduled in the group where it first appears.

To schedule the small jobs, we use the NFDH-Algorithm to place them next to the configurations. We can sort the small jobs by height in $\mathcal{O}(n + \log(n)/\varepsilon^2)$ since there are at most $\mathcal{O}(\log(n)/\varepsilon^2)$ possible processing times.

Note that the total work of the small jobs has to fit next to the configurations. The reason is that the configurations have a total work which equals the total work of the wide jobs. Furthermore, after scheduling the large jobs, the total idle time of the machines was at least as large as the total work of the small jobs.

The NFDH algorithm sorts the small jobs by height and places them into shelves starting with the tallest job, see Figure 8.5. In each shelf there are at most εm machines which are completely idle since each narrow job requires at most εm machines. If there would be more idle machines,

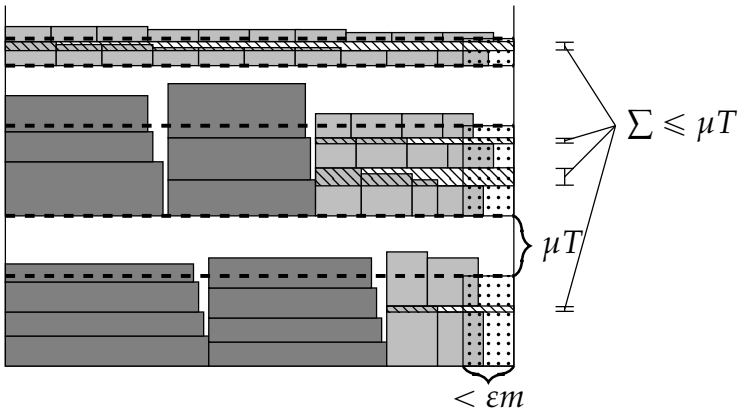


Figure 8.5. Filled configurations, containing wide (dark gray) and narrow (light gray) small jobs

another job would have fitted in this shelf.

Furthermore, there can be machines that start to idle before the starting time of the next shelf, namely in the moment when a job with a processing time smaller than the first job in this shelf has finished its processing time. Let $p_{\max,i}$ be the largest processing time in shelf i, then the idle time of the machines which start to idle in shelf i is bounded by $p_{\max,i} - p_{\max,i+1}$. Therefore, in total, the processing time of machines starting to idle over all shelves is bounded by $p_{\max} \cdot m$ while the total idle time of machine being idle during the whole shelf is bounded by $\varepsilon m \cdot T$. Hence the total work of narrow small jobs that cannot be scheduled next to the configurations is bounded by $\varepsilon m \cdot T + \mu T \cdot m$. By using NFDH again to schedule these jobs, we add at most $(\mu + \varepsilon)mT/(1-\varepsilon)m + p_{\max} \leqslant 2\varepsilon T$ to the makespan.

8.3.4 Medium Jobs

In the last step, we schedule the medium sized jobs. First, we sort them by their processing time. This can be done in $\mathcal{O}(n + 1/\varepsilon^2)$ since there are at most $3/\varepsilon^2$ different processing times between $\mu = \varepsilon^3\delta$ and δ. Afterward, we use the NFDH algorithm to place the jobs. Hence, we start with the tallest one and place the jobs one by one in shelves. By Lemma 2.2, we know that this schedule has a makespan of at most $2\text{work}(\mathcal{J}_M)/m + \max_{j\in\mathcal{J}_M} p(j)$

We know that $W(\mathcal{J}_M)$ is bounded by εmT and $p_{\max}$ is bounded by

$\delta T \leqslant \varepsilon T$. Therefore, we add at most $\text{NFDH}(\mathcal{J}_M) \leqslant 2\varepsilon T + \varepsilon T \leqslant 6\varepsilon\text{OPT}$ to the makespan, by scheduling the medium sized jobs this way.

8.3.5 Summary

Given $\mathcal{J}$, m and ε the algorithm can be summarized as follows:

1. In the first step of the algorithm, we simplify the instance. We define the lower bound $T := \max\{p_{\max}, (\sum_{j\in\mathcal{J}} p_j q_j)/m\}$ and round the processing times such that they are multiples of $\varepsilon T/n$. Next, we find the correct values for δ and μ and partition the jobs into

   $$\mathcal{J}_{L,W} \dot\cup \mathcal{J}_{L,N} \dot\cup \mathcal{J}_M \dot\cup \mathcal{J}_{S,W} \dot\cup \mathcal{J}_{S,N}$$

   accordingly. Afterward, we round the processing times of all jobs using Lemma 3.2 and generate $\bar{\mathcal{J}}_{L,N}$. Last, we generate $\bar{\mathcal{J}}_{S,W}$, i.e., we round the machine requirements of the horizontal jobs.

2. After the simplification steps are done, we start a binary search for the correct size of OPT for the rounded instance. Note that to find the correct value for OPT for the rounded instance, we are only interested in the number of layers needed to place the jobs in $\mathcal{J}_{L,W} \dot\cup \bar{\mathcal{J}}_{L,N} \dot\cup \bar{\mathcal{J}}_{S,W}$. We know that we need at least $l = 1/(\varepsilon\delta)$ layer but at most $u = 2(1+\varepsilon)(1+2\varepsilon)/(\varepsilon\delta)$ layer. We start our binary search using $L = \lfloor (l+u)/2 \rfloor$ layers.

3. Given a number of layers L, we try each possibility to schedule $\mathcal{J}_{L,W} \cup \bar{\mathcal{J}}_{L,N}$ using at most this number of layers. For each of these possibilities, we try to solve LP_{small} for $\bar{\mathcal{J}}_{S,W}$ with last allowed layer L. If LP_{small} is solvable, we save the LP-solution and the choice for $\mathcal{J}_{L,W} \cup \bar{\mathcal{J}}_{L,N}$ and set the upper bound $u = L - 1$ update L accordingly. Otherwise, we try the next choice for $\mathcal{J}_{L,W} \cup \bar{\mathcal{J}}_{L,N}$. If all the possibilities to schedule these jobs fail, we set $l = L + 1$ and update L accordingly.

4. The binary search part is finished as soon as $u < l$. When this is the case, we consider the last solution for LP_{small} and the corresponding choice for $\mathcal{J}_{L,W}$ and $\bar{\mathcal{J}}_{L,N}$. We scale back all the processing times and assign the jobs $\mathcal{J}_{L,N}$ and $\mathcal{J}_{S,W}$ to this solution and shift the fractional

placed jobs to the top or the extra cluster C_2 as described in Section 8.3.2 and Section 8.3.3. We schedule the jobs $\mathcal{J}_{S,N}$ next to the configurations for $\bar{\mathcal{J}}_{S,W}$ using the NFDH algorithm. Finally, we schedule the medium sized jobs on top of the schedule using the NFDH algorithm.

In most of the simplification steps, we have some loss in the approximation ratio of size $\mathcal{O}(\varepsilon T)$. Since $T \leqslant \text{OPT}$ it holds that the algorithm has an approximation ratio of the form $(1+\mathcal{O}(\varepsilon))\text{OPT} + p_{\max}$. To reach a $(1+\varepsilon)\text{OPT} + p_{\max}$ algorithm the value ε has to be scaled accordingly. Note that for the sake of simplicity, we did not optimize the above algorithm to guarantee the best possible running time with regard to the added $\mathcal{O}(\varepsilon)$.

The total running time of the algorithm is bounded by

$$\begin{aligned}&\mathcal{O}(\log(n)/\varepsilon^2 + n\log(1/\varepsilon)\\ &+ \log(1/\varepsilon\delta)((|S|^{1/\delta\alpha}(|S||P_L|/\alpha)^{|S|+|P_L|}))(1/\varepsilon^{10}\delta^2))\\ =&\mathcal{O}(n\log(1/\varepsilon)) + 1/\varepsilon^{1/\varepsilon^{\mathcal{O}(1/\varepsilon)}},\end{aligned}$$

which concludes the proof of Theorem 8.2.

8.4 A Faster Algorithm for a Practical Number of Jobs

Note that in the algorithm described above, we have a running time of $\mathcal{O}(n)$, but the hidden constant can be extremely large. Hence, in practical applications it can be more useful to use an algorithm with running time $\mathcal{O}(n\log(n))$ or $\mathcal{O}(n^2)$, to find an $\alpha\text{OPT} + p_{\max}$ approximation for Parallel Task Scheduling (PTS). For $N \geqslant 6$, we use $\varepsilon \in [1/4, 1/3]$ and, hence, a fast $\text{poly}(n)$ algorithm without large hidden constants and approximation ratio $(5/4)\text{OPT} + p_{\max}$ would bring an significant improvement for the vast majority of cluster numbers with 2 and 5 being the only exceptions. Even an algorithm with approximation ratio $(4/3)\text{OPT} + p_{\max}$ would speed up the algorithm for one third of all the possible instances, namely all the instances where the number of clusters is dividable by three.

To this point, we did not find either of the algorithms, and we leave this as an open question. Instead, we present a fast algorithm with ap-

proximation ratio $(3/2)\mathrm{OPT} + p_{\max}$. This algorithm for PTS leads to an algorithm for MCS with approximation ratio $9/4$ for all instances where $N \bmod 3 = 0$.

In the description of the following algorithm, we need the concept of an idle machine. A machine is *idle* at a time τ if it does not processes any job at that time. Given a point in time τ the number of idle machines at that time is given by

$$\mathrm{idle}(\tau) := m - \sum_{\substack{j \in \mathcal{J}, \\ \sigma(j) \leqslant \tau < \sigma(j)+p(j)}} m(j)$$

and the total idle time up to τ is defined by

$$\tau m - \left(\sum_{\substack{j \in \mathcal{J}, \\ \sigma(j)+p(j) \leqslant \tau}} m(j)p(j) + \sum_{\substack{j \in \mathcal{J}, \\ \sigma(j) \leqslant \tau < \sigma(j)+p(j)}} m(j)(\tau - \sigma(j)) \right).$$

Lemma 8.9. *There is an algorithm for PTS with an approximation guarantee $(3/2)\mathrm{OPT} + p_{\max}$ and running time $\mathcal{O}(n \log(n))$. This schedule can be divided into two clusters C_1 and C_2, where the schedule on C_1 has a makespan of at most $(3/2)\mathrm{OPT}$ and the makespan of C_2 is bounded by $p_{\max}$.*

Proof. In the following, we describe the steps of the algorithm. The first part of the algorithm is to find a schedule for the jobs with machine requirement larger than $m/3$. In the second part, we schedule the jobs with machine requirement at most $m/3$ in a best fit manner. This second part depends on one property from the schedule for the jobs with resource requirement larger than $m/3$, as we will see later. This algorithm uses the following optimized variant of List-Scheduling as described in Turek et al [105]: Starting at time $\tau = 0$ for every endpoint of a job, schedule the widest job that can be started at this point if there is one; otherwise, go to the next endpoint and proceed as before. The first part of the algorithm can be summarized as follows:

1. For a given set of jobs $\mathcal{J}$, first consider the jobs $j \in \mathcal{J}$ with $m(j) \in [m/3, m]$ and sort them by decreasing size of the machine requirement $m(j)$.

2. We stack all the jobs $j \in \mathcal{J}$ with $m(j) > m/2$ ordered by their machine requirement such that the largest starts at time 0, see Figure 8.6.

3. At point in time 0, we start to schedule all the jobs with machine requirement in $(m/3, m/2]$ with the optimized List-Schedule. The List-Schedule includes all the endpoints of the already scheduled jobs. We denote with τ the first point in time in the schedule where two jobs are processed simultaneously.

Let τ' be the point in time, where the last job ends, which needs more than $m/2$ machines and define τ'' to be the first point in time where both jobs scheduled at τ' have ended. Furthermore, let T' be the last point in the schedule where two jobs are processed, T'' be the last point in the schedule where any job is processed, and define $T := \max\{(T' + T'')/2, \tau'\}$. Note that at each point in the schedule after τ' and before T' there will be scheduled exactly two jobs with machine requirement in $(m/3, m/2]$, while before τ' it can happen that there is no job from this set.

We claim that $T \leqslant \text{OPT}$. If $T = \tau'$, this is obvious since we never can schedule jobs with machine requirement larger than $m/2$ at the same time. Consider the case that $T = (T' + T'')/2$. Note that when splitting the part of the last job that is scheduled after T' in half and schedule both half at the same time, both parts would end at T. Hence we can assume that between τ' and T there will be scheduled two jobs at each point in the processing time. Hence if T is larger than OPT there has to be a schedule, where the total processing time of jobs that have a width between $m/3$ and $m/2$ and are scheduled in parallel to jobs with machine requirement larger than $m/2$ is strictly larger than in the current schedule. However, since the algorithm placed all the jobs that would fit next to the jobs with machine requirement larger than $m/2$ in parallel of these jobs, such a schedule cannot exist.

In the next step, we are going to schedule the residual jobs, which have a machine requirement of at most $m/3$. In order to schedule these jobs, we might dismantle the schedule generated so far. This dismantling is necessary, if the schedule generated so far has a too large amount of idle time on the machines, because in this case we cannot guarantee a small approximation ratio, when scheduling the residual jobs. Furthermore, note that if there are no jobs with machine requirement at most $m/3$, we do not

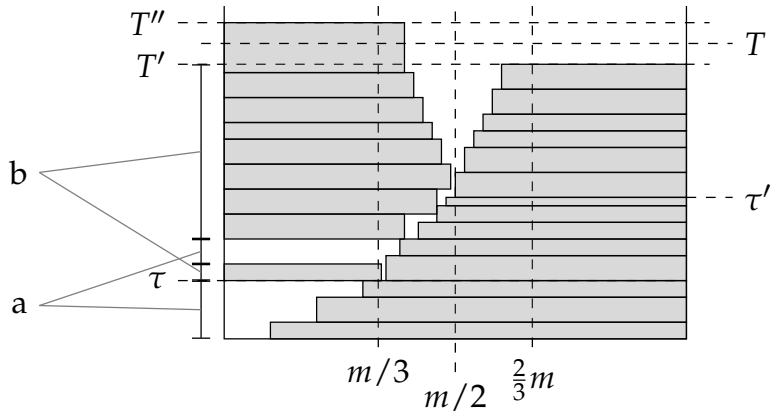


Figure 8.6. A placement of the jobs with processing time larger than $m/3$ generated bz the described algorithm. The value τ represents the first time in the schedule where two jobs are scheduled. The value τ' represents the last point in the schedule where a job with machine requirement larger than $m/2$ is scheduled. T represents the maximum of the last point in the schedule where two jobs are scheduled and τ'. The value b represents the total processing time up to T where at least two jobs are scheduled, while a represents the total processing time up to T where at most one job is scheduled. Note that after the point τ' the machine requirements are decreasing on both sides, because at each point in time the widest fitting job is scheduled.

need to add further steps and have found a schedule with approximation guarantee $\text{OPT} + p_{\max}$.

Let a be the total processing time before τ', where only one job is scheduled. This job has to be a job with machine requirement larger than $m/2$. On the other hand define $b := T - a$. Note that when splitting the part of the last job that is scheduled after T' in half and schedule both half at the same time, both parts would end at T and the total processing time where two jobs are scheduled is given by b. We will now consider two cases: $a > b$ and $a \leqslant b$. In the first case, we have to dismantle the schedule found so far, while in the second case this is not necessary.

We can summarize the second part of the algorithm, where we schedule the jobs with machine requirement at most $m/3$ as follows:

4. Find a and b.

5. If $a > b$, dismantle the schedule and stack all the jobs with machine requirement larger than $m/3$ on top of each other, sorted by machine requirement such that the widest one starts at 0. Schedule the residual jobs with the modified List-Schedule starting at 0 and using the endpoints of all jobs.

6. Else if $a \leqslant b$, determine τ'' and use the optimized List-Schedule to schedule the remaining starting at τ'' while using the endpoints of all scheduled jobs.

In the following, we will argue that the second part of the described algorithm delivers a schedule with approximation guarantee $(3/2)\mathrm{OPT} + p_{\max}$.

Case 1: $a > b$. In this case, the algorithm performs the following steps: We stack all the jobs with machine requirement larger than $m/3$ on top of each other sorted by decreasing number of required machines. This stack has a height of at most $a + 2b$. For the remaining jobs, i.e., the jobs with machine requirement at most $m/3$, we us the the improved List-Schedule algorithm as described in Turek et al. [105]. This means, we go through the schedule from the bottom to the top and look for each end point of jobs t, starting with $t = 0$, at the number of idle machines $\mathrm{idle}(t)$. We search for the widest unscheduled job with $m(j) \leqslant \mathrm{idle}(t)$ and start it at this time, if one exists, and calculate the new number of idle machines at this point in time. If no such job exists, we go to the next end point of a job since the number of idle machines only changes at these points.

We claim that this schedule has a makespan of at most $(3/2)\mathrm{OPT} + p_{\max}$. Let ρ be the last starting point of a job in this schedule. If this point is larger than $a + 2b$, the last scheduled job has a machine requirement of at most $m/3$. By construction of the schedule, this job could not be scheduled at any earlier time. Hence at each time in the schedule before ρ, we use at least $(2/3)m$ machines and therefore $\rho(2/3)m \leqslant W(\mathcal{J})$. Furthermore, we know that $\mathrm{OPT} \geqslant W(\mathcal{J})/m$. As a consequence it holds that $\rho \leqslant (3/2)\mathrm{OPT}$. Since ρ is the last starting position of all jobs, the makespan of the schedule is bounded by $(3/2)\mathrm{OPT} + p_{\max}$.

On the other hand if $\rho \leqslant a + 2b$, the last starting job can be a job with machine requirement larger than $m/3$. However, the schedule is then

bounded by $a + 2b + p_{\max}$. Since $a > b$ and $a + b \leqslant \text{OPT}$ it holds that $b \leqslant \text{OPT}/2$ and therefore $a + 2b + p_{\max} \leqslant (3/2)\text{OPT} + p_{\max}$.

Case 2: $a \leqslant b$. We now consider the case that $a \leqslant b$. In this scenario, we do not dismantle the given schedule as we do in Case 1. Instead, we use the improved List-Schedule algorithm as described in Turek et al. [105] to schedule the remaining jobs. To prove that the resulting algorithm has an approximation guarantee of $(3/2)\text{OPT} + p_{\max}$, we analyze the total idle time up to the point $\max\{T', \tau'\}$ before we schedule the residual jobs.

Let t_a be an arbitrary point in time before $\max\{T', \tau'\}$ where only one job is scheduled and let t_b be an arbitrary point in time where two jobs are scheduled. Note that $\text{idle}(t_b) < m/3$, since both jobs scheduled at this time have a machine requirement of at least $m/3$. We differentiate two cases $t_b < \tau'$ and $t_b \geqslant \tau'$ and claim that in both cases the sum of numbers of idle machines at t_a and t_b is bounded by $\frac{2}{3}m$. As a a consequence of this claim, the average number of idle machines at all of these pairs of points is bounded by $m/3$ and hence, the total idle time up tp the point T is bounded by $m/3 \cdot (a + b) \leqslant Tm/3$ because $a \leqslant b$ and at each point t_b the idle time is bounded by $m/3$.

Case 2.1: $t_b < \tau'$. In the case that $t_b < \tau'$, the number of idle machines $\text{idle}(t_b)$ is bounded by $m/6$ since there is scheduled one job with machine requirement at least $m/2$ and one job with machine requirement at least $m/3$. On the other hand, $\text{idle}(t_a)$ is bounded by $m/2$. Therefore, the sum of free machines on both points is bounded by $\frac{2}{3}m$ and hence the average is bounded by $m/3$.

Case 2.2: $t_b \geqslant \tau'$. If $t_b \geqslant \tau'$, there are two jobs with machine requirement at least $m/3$ scheduled at this point in time and hence $\text{idle}(t_b) < m/3$. Remember that $t_a < \tau'$ since at each point in time after the point τ' up to the point $\max\{T', \tau'\}$ there will be two jobs scheduled. Therefore, $t_a < t_b$ and the jobs scheduled at t_b did not fit at the time t_a since otherwise they would have been scheduled there. As a consequence, it holds that $\text{idle}(t_a) \leqslant (m - \text{idle}(t_b))/2$ because the job with the smaller machine requirement scheduled at t_b has a machine requirement of at most $(m - \text{idle}(t_b))/2$. Hence it holds that $\text{idle}(t_a) + \text{idle}(t_b) \leqslant m/2 + \text{idle}(t_b)/2$. Since $\text{idle}(t_b) \leqslant m/3$, we have $\text{idle}(t_a) + \text{idle}(t_b) \leqslant \frac{2}{3}m$.

In conclusion, we have $\text{idle}(t_a) + \text{idle}(t_b) \leqslant \frac{2}{3}m$ in both cases $t_b < \tau'$ and $t_b \geqslant \tau'$. Hence the average number of idle machines for each pair of two points t_a and t_b is bounded by $m/3$. Since $a \leqslant b$ and at each point t_b, where two jobs are scheduled, there are at most $m/3$ machines idle, the total idle time below T is bounded by $Tm/3$. The residual jobs are scheduled by the best fit algorithm in [105]. Let τ'' be the first point in time where both jobs scheduled at τ' have ended. Note that after this point in time the number of idle machines is monotonically increasing per time step. Hence, we can use the improved List-Schedule algorithm without constructing any machine conflicts.

To analyze the approximation ratio after adding the residual jobs, let ρ be the last point in the schedule where a job is started. If this job has a width of at most $m/3$ at every time before ρ and after $\max\{T', \rho'\}$ the number of idle machines is at most $m/3$ since otherwise this job would have been started earlier. If this job has a machine requirement larger than $m/3$ it has been started before $\max\{T', \rho'\}$. In both cases the total idle time up to ρ is bounded by $\rho m/3$. As a consequence, we have $\rho \leqslant 3/2 \cdot \text{OPT}$ since all jobs start before ρ and $m\text{OPT} \geqslant \sum_{j \in \mathcal{J}} p_j q_j \geqslant (2/3)\rho m$. Therefore, the schedule has a makespan of at most $(2/3)\text{OPT} + p_{\max}$.

We have proven that in both cases $a > b$ and $a \leqslant b$ the described algorithm produces a schedule with makespan at most $(2/3)\text{OPT} + p_{\max}$. This algorithm has a running time of the form $\mathcal{O}(n \log(n))$: The sorting of the items is possible in $\mathcal{O}(n \log(n))$; each of the values a, b and τ'' can be found in $\mathcal{O}(n)$; and last the optimized List-Schedule can be implemented to be in $\mathcal{O}(n \log(n))$ by organizing the relevant points in time as well as the set of items inside a search tree.

Last, we describe how to partition this schedule into the schedule on the two clusters C_1 and C_2 as needed for the algorithm in Lemma 8.5. Note that in all the described cases the additional $p_{\max}$ is added by the last started job. To partition this schedule such that it is scheduled on the two clusters C_1 and C_2, we look at the starting time ρ of the last started job. We remove this last started job and all the jobs which end strictly after ρ and place them into the second cluster C_2 and leave the rest untouched to be the schedule for C_1. As we noted before the schedule up to ρ has a height of at most $(3/2)\text{OPT}$. Furthermore, since the last job starts at ρ,

all the removed jobs have a total machine requirement of at most m, and, hence, we can start them all at the same time. The resulting schedule on C_2 has a height of at most $p_{\max}$. □

In the next step, we present the technique to divide this schedule on C_1 and C_2 to N clusters and prove Theorem 8.4 in this way. The used technique is similar to the technique in Section 8.2. However, it is no longer possible to partition the schedule into sections of height at most 2OPT.

8.4.1 Proof of Theorem 8.4

In this section, we prove Theorem 8.4. We start with the schedule given by the $(3/2)\text{OPT} + p_{\max}$ algorithm from Lemma 8.9 and its partition onto the two clusters C_1 and C_2. To partition the schedule on C_1 onto the different clusters, we differentiate the three cases $N = 3i$, $N = 3i + 1$ and $N = 3i + 2$.

Case 1: $N = 3i$ In this case, the schedule on C_1 has a height of $T \leqslant (9i/2)\text{OPT}$. We partition it into $2i$ parts of equal height $T/(2i) \leqslant 9/4\text{OPT}$. During this partition step, we cut the schedule $2i - 1$ times. The jobs intersected by this cut have to be scheduled separately using height p_{max}. Together with the jobs in C_2, we have $2i$ sets of jobs with height bounded by p_{max} and machine requirement bounded by m. We schedule these sets pairwise in i additional clusters analogously to the clusters of type B in Section 8.2. In total, we use $3i = N$ Clusters and the largest one a has height of at most $(9/4)\text{OPT} = 2.25\text{OPT}$.

Case 2: $N = 3i + 1$ In this case, the schedule on C_1 has a height of $T \leqslant (3(3i + 1)/2)\text{OPT} = ((9i + 3)/2)\text{OPT}$. We partition the schedule into $2i$ parts of equal height and one part with a smaller height. On this part, we schedule the jobs from C_2 as well. Let $T_A := (2/(9i + 3))T \leqslant \text{OPT}$. The $2i$ parts of equal height have a size of $((9i + 5)/(4i + 2))T_A$ and the last part has a height of $((5i + 3)/(4i + 2))T_A$. It is easy to verify the $2i \cdot ((9i + 5)/(4i + 2))T_A + ((5i + 3)/(4i + 2))T_A = ((9i + 3)/2)T_A = T$ and hence we have partitioned the complete schedule on C_1. By partitioning the schedule on C_1 into these parts, we have cut the schedule $2i$ times. Therefore, together with the jobs on C_2, we have to schedule $2i + 1$ parts

of height $p_{\max}$. We schedule C_2 on the cluster with current makespan $((5i+3)/(4i+2))T_A$ resulting in a schedule of height $((5i+3)/(4i+2))T_A + p_{\max} \leqslant ((9i+5)/(4i+2))\text{OPT}$, (since $p_{\max} \leqslant \text{OPT}$). We pair the other $2i$ parts and schedule them on i distinct clusters. In total, we generate $2i+1+i = 3i+1$ cluster and the largest occurring makespan is bounded by $((9i+5)/(4i+2))\text{OPT}$.

Case 3: $N = 3i+2$ In this case, the schedule on C_1 has a height of $T \leqslant (3(3i+2)/2)\text{OPT} = ((9i+6)/2)\text{OPT}$. Again, we partition this schedule into $2i+1$ parts of equal height and one part with a smaller height. On top of this part, we will schedule two parts with processing time $p_{\max}$. Let $T_A := (2/(9i+6))T \leqslant \text{OPT}$. The first $2i+1$ parts of C_1 have a height of $((9i+10)/(4i+4))T_A$ and the last part has a height of at most $((i+2)/(4i+4))T_A$. It is easy to verify that $(2i+1)((9i+10)/(4i+4))T_A + ((i+2)/(4i+4))T_A = ((9i+6)/2)T_A = T$ and, hence, we have scheduled all parts of C_1. Since $((i+2)/(4i+4))T_A + 2p_{\max} \leqslant ((9i+10)/(4i+4))\text{OPT}$, we can schedule two parts with processing time at most $p_{\max}$ on this cluster. We have cut the schedule on C_1 exactly $2i+1$ times. Together with the jobs from C_2, we have $2i+2$ parts with processing time at most $p_{\max}$ we have to schedule inside the other clusters. Since we already have scheduled two of these parts, we pair the residual $2i$ parts and generate i new clusters with makespan at most $2p_{\max} \leqslant 2\text{OPT}$. In total, we generated $2i+2+i = 3i+2$ clusters and the largest makespan occurring on the clusters is bounded by $((9i+10)/(4i+4))\text{OPT}$.

Note that this case also applies for $N = 2$. The schedule on C_1 has a height of $T \leqslant \frac{3}{2}N\text{OPT} = 3\text{OPT}$. We define $T_A := \frac{1}{3}T \leqslant \text{OPT}$. We cut the schedule once at the height $\frac{5}{2}T_A \leqslant \frac{5}{2}\text{OPT}$. The residual part has a height of $T - \frac{5}{2}T_A \leqslant \frac{1}{2}T_A \leqslant \frac{1}{2}\text{OPT}$. On this part, we place the jobs cut by the horizontal line at $\frac{5}{2}T_A$ and the jobs from the cluster C_2. Hence the makespan on this cluster is bounded by $\frac{5}{2}\text{OPT}$.

For each of the three cases $N = 3i$, $N = 3i+1$, and $N = 3i+2$, we have presented a partitioning strategy which distributes the schedule from clusters C_1 and C_2 onto N clusters such that each cluster has a makespan of at most $(9/4)\text{OPT}$, $((9i+5)/(4i+2))\text{OPT}$ or $((9i+10)/(4i+4))\text{OPT}$ respectively. Hence, we have proven Theorem 8.4.

8.5 An AEPTAS for Strip Packing

In this section, we present an $(1+\varepsilon)\mathrm{OPT} + h_{\max}$ algorithm for Strip Packing (SP) with running time $\mathcal{O}(n/\varepsilon) + \mathcal{O}_\varepsilon(1)$ proving the second part of Theorem 8.2. More precisely in this section, we will prove the following lemma.

Lemma 8.10. *There exists an algorithm, that finds for each $\varepsilon \geqslant 0$, each $c \in \mathbb{N}$, and each instance $I = (\mathcal{J}, m)$ of the Strip Packing problem a packing in two strips C_1 and C_2 with width W and $\lfloor W/c \rfloor$ respectively, such that the packing height on C_1 is bounded by $(1+\varepsilon)\mathrm{OPT}$ and on C_2 by p_{max}. This algorithm has a running time that is bounded by $\mathcal{O}(n \log(1/\varepsilon)) + 1/\varepsilon^{c \cdot 1/\varepsilon^{\mathcal{O}(1/\varepsilon)}}$.*

When we use $c = 1$ and place the jobs from cluster C_2 on top of the schedule on C_1, we get the corresponding schedule for the second part of Theorem 8.2. The algorithm is inspired by the algorithm in [67]. However, we made some improvements to guarantee an efficient running time.

8.5.1 Simplify

Similar as in Section 8.3, we start with defining an upper and a lower bound for the approximation ratio. Let $\mathrm{area}(\mathcal{I}) := \sum_{i \in \mathcal{I}} w(i)h(i)$ be the total area of all the items and let $h_{\max}$ be the largest occurring height in $\mathcal{I}$. By Steinberg [100], we now that

$$\max\{\mathrm{area}(\mathcal{I}), h_{\max}\} \leqslant \mathrm{OPT} \leqslant 2\max\{\mathrm{area}(\mathcal{I}), h_{\max}\}$$

and we define $T := \max\{\mathrm{area}(\mathcal{I}), h_{\max}\}$.

In the first step, we partition the items by their size. Other than in the algorithm for Parallel Task Scheduling (PTS), we need a gap between wide and narrow items as well. Hence, we partition the items into large $\mathcal{L} := \{i \in \mathcal{I} | h(i) \geqslant \delta T, w(i) \geqslant \delta W\}$, vertical $\mathcal{V} := \{i \in \mathcal{I} | h(i) \geqslant \delta T, w(i) \leqslant \mu W\}$, horizontal $\mathcal{H} := \{i \in \mathcal{I} | h(i) \leqslant \mu T, w(i) \geqslant \delta W\}$, small $\mathcal{S} := \{i \in \mathcal{I} | h(i) \leqslant \mu T, w(i) \leqslant \mu W\}$, and medium sized items $\mathcal{M} := \mathcal{I} \backslash (\mathcal{L} \cup \mathcal{V} \cup \mathcal{H} \cup \mathcal{S})$ for some $\delta, \mu \leqslant \varepsilon$, see Figure 8.7.

We will discard the medium sized items and place them at the end of the packing. To make this possible the total area of the medium sized

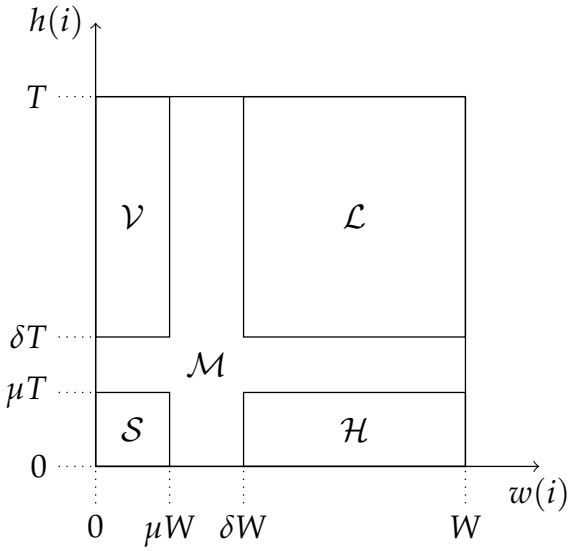


Figure 8.7. This figure shows the partition of the items. Each item $i \in \mathcal{I}$ can be represented by a point in this two-dimensional plane. The x-coordinate of the point corresponds to the width of the item, while the y-coordinate corresponds to the height of the item.

items has to be small. In the next lemma, we show that we can find values for δ and μ which guarantee this property.

Lemma 8.11. *Consider the sequence $\sigma_1 := \varepsilon^5/cx$, $\sigma_{i+1} = \sigma_{i+1}^3 \varepsilon^5/cx$ for any constant $x \in \mathbb{N}$. There exists an $j \in \{1, \ldots, 2c/\varepsilon\}$ such that when defining $\delta = \sigma_j$ and $\mu = \sigma_{j+1}$ the total area of the medium sized items $\mathcal{I}_M$ is bounded by $\varepsilon WT/c$.*

Proof. This Lemma follows by a direct application of the pigeon hole principle. Let $\mathcal{M}_j$ be the set of medium sized items when defining $\delta = \sigma_j$ and $\mu = \sigma_{j+1}$. Each item $i \in \mathcal{I}$ can appear in at most two of these sets, in the first because its width is between μW and δW and in the second, because its height is between μT and δT. Assume that all the sets $\mathcal{M}_j$ have an area of $\text{area}(\mathcal{M}_j) > \varepsilon WT/c$. As a consequence, the total area of all these sets is at least $\sum_{j=1}^{2c/\varepsilon} \text{area}(\mathcal{M}_j) > 2 \cdot W \cdot T$, a contradiction since the total area of all the items is bounded by WT. □

Furthermore, it holds that $\delta \geqslant (\varepsilon/cx)^{3^{\mathcal{O}(c/\varepsilon)}}$. We define $\delta' := \varepsilon^k$ as

the maximum number such that $\delta' \geqslant \delta \geqslant \delta'/\varepsilon$. Note that $k \in 3^{\mathcal{O}(c/\varepsilon)} \cdot \log_\varepsilon(1/cx)$. We use δ' for the partitioning of the items. As a consequence, the the area of the medium items is still at most $\varepsilon WT/c$, but the distance between δ' and μ is reduced, i.e. we have $\mu = \delta^3\varepsilon^5/cx \leqslant (\delta'/\varepsilon)^3\varepsilon^5/cx = \delta'^3\varepsilon^2/cx$. For simplicity of notation, we will write δ instead of δ' in the following and use $\mu \leqslant \delta\varepsilon^2/(cx)$ respectively.

In the second step, we round the heights of the items. By increasing the packing height by at most εT, we can round the heights of the items to multiples of $\varepsilon T/n$, because adding $\varepsilon T/n$ to each processing time lengthens the packing by at most $n \cdot \varepsilon T/n$. Hence after this rounding step, we have $T \leqslant \mathrm{OPT} \leqslant (2+\varepsilon)T$. Since each item has a height of at most T, there are at most n/ε different item sizes, and hence, sorting them by height can be done in $\mathcal{O}(n/\varepsilon)$ using Bucket-Sort. Furthermore, the largest l such that $p_j \in \{\varepsilon^l T, \varepsilon^{l-1}T\}$ is bounded by $\mathcal{O}(\log(n))$.

In the next step, we scale the instance with $n/(\varepsilon T)$. As a result all the items have a height that is one of the integral values $\{1, 2, \ldots, n/\varepsilon\}$ and the optimal packing height for this scaled instance is one of the integral values $\{n/\varepsilon, n/\varepsilon + 1, \ldots, 2n/\varepsilon + n\}$, because for the rounded instance it holds that $T \leqslant \mathrm{OPT} \leqslant 2T + \varepsilon T$ and the optimal packing height has to be integral since all the item heights are integral. We scale T accordingly such that $T = n/\varepsilon$. In the algorithm, we will do a binary search over the packing heights.

Next, we use the same geometric rounding as above to round the heights of the items to fewer different heights using Lemma 3.2 and loose a factor of at most $(1+2\varepsilon)$ in the approximation ratio with regard to the scaled instance. After this rounding the items have at most $\mathcal{O}(\min\{n/\varepsilon, \log(n)/\varepsilon^2\})$ possible different sizes and, without any further loss, we can assume that all large and vertical items start at multiples of $\varepsilon\delta' T$. We call the area between two consecutive multiples of $\varepsilon\delta' T$ a layer and number them starting at zero. To ensue the integrity of the item heights, we scale the instance with $1/(\varepsilon\delta)$ before the rounding step and scale T accordingly such that $T = n/(\varepsilon^2\delta)$. Note that $1/(\varepsilon\delta) \in \mathbb{N}$ since $1/\varepsilon \in \mathbb{N}$. To this point, we know that without all the scaling steps it holds that $T \leqslant \mathrm{OPT} \leqslant (1+2\varepsilon)(2+\varepsilon)T$. Hence the number of layers L in an optimal solution is at least $1/(\varepsilon\delta)$ and at most $(1+2\varepsilon)(2+\varepsilon)/(\varepsilon\delta) \leqslant 5/(\varepsilon\delta)$ for $\varepsilon \leqslant 1/2$.

In the next step, we remove all small and medium sized items from the optimal packing and use a lemma that is similar to Lemma 5.22, which states that we can partition any optimal packing into a constant number of sub areas such that each subarea contains just one type of item. Since the proof of this lemma works analogue, we will not prove it again.

Lemma 8.12 (Compare Lemma 5.22)**.** *We can partition the area* $W \times (1 + 2\varepsilon)\mathrm{OPT}$ *into* $\mathcal{O}(1/(\varepsilon\delta'^2))$ *rectangular areas called boxes.*

- ▻ *Each large item* $i \in \mathcal{L}$ *is contained in its personal box of height* $h(i)$ *and width* $w(i)$*.*
- ▻ *There are at most* $\mathcal{O}(1/(\varepsilon\delta^2))$ *many boxes containing horizontal items* $i \in \mathcal{H}$*. Each of them has a height of* $\varepsilon\delta' T$ *and a width larger than* $\delta' W$*.*
- ▻ *There are at most* $\mathcal{O}(1/(\varepsilon\delta^2))$ *many boxes containing vertical items* $i \in \mathcal{V}$*.*
- ▻ *No item in* $\mathcal{H}$ *is intersected vertically by any box border, but can be intersected horizontally.*
- ▻ *No item in* $\mathcal{V}$ *is intersected horizontally by any box border, but can be intersected vertically.*
- ▻ *Each boxes lower and upper borders are at multiples of* $1/(\varepsilon\delta)\mathrm{OPT}$*.*

In the algorithm, we cannot try each of these partitions since then the width of the strip W would appear linear in the running time. Instead, we are interested in the relative positioning of the large items and the boxes for horizontal items.

8.5.2 Boxes for horizontal rectangles

The last simplification step is the rounding of widths of the horizontal items. We call the set of generated rounded items $\bar{\mathcal{H}}$. We use geometric grouping to reduce the number of different widths of the items.

Lemma 8.13. *We can round the width of the horizontal items to* $\mathcal{O}(\log(1/\delta)/\varepsilon)$ *different sizes in at most* $\mathcal{O}(n \log(1/\varepsilon))$ *operations. These rounded items can be placed fractionally instead of the horizontal items and an extra box of height* εT*.*

Proof. By Lemma 3.5, we can round the width of the horizontal items to $\mathcal{O}(\log(1/\delta)/\varepsilon)$ different sizes in at most $\mathcal{O}(n \log(1/\varepsilon))$ operations, when using $\rho := \varepsilon/2$. These rounded items can be placed fractionally instead of the horizontal items and an extra box of height εT. □

In the next step, we show that it is possible to reduce the number of widths for horizontal boxes to be constant depending on δ. We do this in order to make it possible for the algorithm to guess their sizes in polynomial time.

Lemma 8.14. *Given a partition of the optimal solution into boxes, we can reduce the number of possible width for the boxes to $|\bar{\mathcal{I}}_H|^{1/\delta}$ and guarantee that at most $\mathcal{O}(1/(\varepsilon\delta))$ of these sizes are used in the partition by exactly $1/\delta$ boxes each. This rounding step adds at most εT to the packing height.*

Proof. We reduce the number of box sizes in two steps. First, we reduce the possible number of box sizes by shrinking the boxes to be a combination of widths of the rounded horizontal items. In the second step, we reduce the number of different box sizes per solution by using a linear grouping step.

Look at one box B for horizontal items. We can shift all the horizontal items in this box to the left as much as possible such that all the left borders of the horizontal items are touching either the box border or the right side of another horizontal item. If the left border of the box does not touch the leftmost item, we can move this border to the left until it does. Now the box for horizontal items has a width which is the sum of widths of rounded horizontal items, i.e. $w(B) \in \{\sum_{i=1}^{1/\delta-1} w_i | i \in \bar{\mathcal{I}}_H\}$. As a result the total number of possible box widths is bounded by $|\bar{\mathcal{I}}_H|^{1/\delta}$.

Given such a set of boxes, we can use linear grouping to reduce the total number of different box widths. Since the optimal packing has a height of at most $(1+2\varepsilon)(1+\varepsilon)2T$ and each box has a height of $\varepsilon\delta T$ and there are at most $1/\delta$ boxes for horizontal items in each layer, a sorted stack of all the boxes has a total height of at most $\varepsilon\delta T \cdot (1+2\varepsilon)(1+\varepsilon)2/\varepsilon\delta^2 \leqslant (1+2\varepsilon)(1+\varepsilon)2T/\delta$. We partition the set of boxes such that the the $1/\delta$ widest boxes are contained in the first set, the $1/\delta$ next most wide boxes are contained in the second set and so on. As a result, the total height of each set of boxes is bounded by εT and the set of boxes is partitioned

into at most $(1+2\varepsilon)(1+\varepsilon)2/(\varepsilon\delta) = \mathcal{O}(1/(\varepsilon\delta))$ groups. Note that the last group might contain less boxes than $1/\delta$. To enforce that after the rounding there are $1/\delta$ boxes of each width, we assume that the last group has additional boxes with width zero. We round the box widths to the largest box width of the corresponding set. Again the last rounded group of boxes has to be positioned at the end of the packing adding at most εT to the packing height. □

Let $\mathcal{W}_B$ be the set of rounded widths of the boxes. Note that $\mathcal{W}_B$ can contain less than $2(1+2\varepsilon)(1+\varepsilon)/(\varepsilon\delta)$ sizes if there are less than $2(1+2\varepsilon)(1+\varepsilon)/(\varepsilon\delta^2) - 1/(2\delta)$ boxes in the partition of the optimal instance. To place the horizontal items, we first guess the set $\mathcal{W}_B$. There are at most $\mathcal{O}((|\bar{\mathcal{I}}_H|^{1/\delta})^{\mathcal{O}(1/(\varepsilon\delta))}) \leqslant \mathcal{O}((\log(1/\delta)/\varepsilon)^{\mathcal{O}(1/(\varepsilon\delta^2))})$ possibilities for this set.

After we guessed the set of boxes, we check with a linear program whether all the rounded horizontal items can be placed into the boxes. Similar to the placing of small jobs in Section 8.3.3, we use configurations to place the horizontal items into the boxes. A configuration of horizontal items is a multiset $C := \{a_{i,C} : i | i \in \bar{\mathcal{I}}_H\}$. Let $\mathcal{C}$ be the set of all configurations. We say a configuration C has width $w(C) := \sum_{i\in\bar{\mathcal{I}}_H} a_{i,C} w(i)$. Let $\mathcal{C}_w$ be the set of configurations with width at most w, i.e., $\mathcal{C}_w := \{C \in \mathcal{C} | w(C) \leqslant w\}$.

Consider the following linear program LP_{small}.

$$\sum_{C\in\mathcal{C}_W} x_{C,W} = 3\varepsilon T \tag{8.8}$$

$$\sum_{C\in\mathcal{C}_w} x_{C,w} = \varepsilon\delta T \qquad \forall w \in \mathcal{W}_B \tag{8.9}$$

$$\sum_{l=1}^{|S|} \sum_{C\in\mathcal{C}_{m_l}} x_{C,l} a_{j,C} = h_j \qquad \forall j \in \bar{\mathcal{I}}_H \tag{8.10}$$

$$x_{C,l} \geqslant 0 \qquad \forall l = 1,\ldots,|S|, C \in \mathcal{C}_w \tag{8.11}$$

The variables $x_{C,w}$ represent the height of a configuration C inside the boxes of width w. The sum of these heights should equal the total height of the boxes having this width, which is ensured by the equation (8.9). Equation (8.8) is introduced to represent the extra box for the horizontal items we need due to the rounding of these items. On the other hand, each

horizontal item should be covered by the configurations, which is ensured by the equation (8.10).

Similar as for placing the small narrow jobs in Section 8.3.3, we solve a relaxed version of this linear program called $LP_{small,rel}$. In this relaxed version, we replace equation (8.9) by the equation

$$\sum_{C \in \mathcal{C}_w} x_{C,w} = (1+\varepsilon^2)\varepsilon\delta T \ \forall w \in \mathcal{W}_B$$

and, similarly, we replace the equation (8.8) by

$$\sum_{C \in \mathcal{C}_W} x_{C,W} = (1+\varepsilon^2)\varepsilon T.$$

Lemma 8.15. *If there is a solution to LP_{small}, we can find a basic solution to $LP_{small,rel}$ in $\mathcal{O}(\log(1/\delta)^{1.5356}/\varepsilon^6\delta^6)$ operations.*

Proof. Note that the described linear program and the described configurations are equivalent to the ones in Section 3.3. Hence, we can use the algorithm proposed in Lemma 3.9 to find the desired basic solution. □

We call the set of guessed boxes for horizontal items $\mathcal{B}_H$. In the end of the algorithm, we place the configurations inside the boxes and the horizontal items (fractionally) into the configurations similar to the placement of small wide jobs in Section 8.3.3. A basic solution of the above linear program has at most $|\mathcal{W}_B| + |\bar{\mathcal{I}}_H| + 1$ non zero components. When filling the configurations inside the boxes $\mathcal{B}_H$, we have to cut the configurations at the box borders of boxes with the same size. Hence, inside the boxes, we have at most $|\mathcal{B}_H| + |\bar{\mathcal{I}}_H| + 1$ configurations. At each configuration border, we generate fractionally placed horizontal items. However these items all fit next to each other since they are inside one configuration. Hence, we can remove the cut items and shift them up to the top of the packing. This step adds at most $\mu T \cdot (|\mathcal{B}_H| + |\bar{\mathcal{I}}_H| + 1) \leqslant \mu T(\log(1/\delta)/\varepsilon + \mathcal{O}(1/\varepsilon\delta^2)) = \mathcal{O}(\varepsilon T)$ to the packing height.

8.5.3 Positioning containers as well as large and vertical rectangles

In this section, we handle the positioning of the boxes for horizontal items and the placement of large and vertical items. These boxes and items are positioned by guessing the x-coordinate of the lower left corner, which has to be a multiple of $\varepsilon\delta T$. Afterward, we guess the order from left to right in which these items and boxes will appear. The technique described in this section is inspired by the techniques described in [67] Chapter 4.

In the first step, we guess the position of the lower corners of the items and boxes in $\mathcal{I}_L$ and $\mathcal{B}_H$. Note that since the boxes have an area of at least $\varepsilon\delta T \cdot \delta W$ and the large items have an area of at least $\delta T \cdot \delta W$ and the packing has an area of at most $(1+2\varepsilon)(1+\varepsilon)TW$, there are at most $\mathcal{O}(1/(\varepsilon\delta^2))$ boxes and items. Hence, the total number of possible guesses for positions of their bottom edges is bounded by $(1/\varepsilon\delta)^{\mathcal{O}(1/(\varepsilon\delta^2))}$.

Consider an optimal packing where all the items are rounded and the horizontal items are positioned in the rounded boxes. For each large item or box $i \in \mathcal{I}_L \cup \mathcal{B}_H$, we can determine the value of the y-coordinates of their left and right borders $y_{i,l}$ and $y_{i,r}$. Let $\mathcal{Y}$ be the set of all these y-coordinates $y_{i,l}$ and $y_{i,r}$. We order $\mathcal{Y}$ by value of the coordinates in the optimal packing. This gives us a permutation $\pi : \mathcal{Y} \to \{1, \ldots, |\mathcal{Y}|\}$ from the left and right corners of items and boxes to positions in the ordered list. Since the value of W is not logarithmically bounded in the input size, we cannot guess the values of the y-coordinates in polynomial time. However, it is possible to guess the correct permutation π in $|\mathcal{Y}|! \in (1/(\varepsilon\delta^2))^{\mathcal{O}(1/(\varepsilon\delta^2))}$ guesses. For a given item or box $i \in \mathcal{I}_L \cup \mathcal{B}_H$, we write $\pi(i,l)$ to refer to the position of $y_{i,l}$ and analogously $\pi(i,r)$ for the position of $y_{i,r}$ and write y_j to refer to the y-coordinate which is mapped to position j in the ordered list.

After these two guesses, the guess of the positions of lower borders and the guess of order of the items, the algorithm tests if this guess was feasible, by testing if it is possible at all to position the items as forced by this guess. This can be done in $\mathcal{O}(n)$ by starting with the left most item and position the items one by one in order of the y-coordinates as most to the left as possible by the constraints guessed. As soon as a constraint has to be violated, we stop and discard the guess. Possible violations of the constraints can be, e.g., that an item's left border has to be placed between

a left and a right border of another item but this item and the to be placed item overlap the same horizontal line that an item has to be placed such that it overlaps the right border of the strip that $\pi(i,l) > \pi(i,r)$.

Consider a feasible guess of starting positions and permutation. The next step of the algorithm is to find values for the y-coordinates of the left and right borders. It determines these values by using a linear program as described below. Indeed, since the vertical items have to be placed correctly as well, the linear program is not only concerned about determining the y-coordinates, but to place the vertical items as well. Consider two consecutive y-coordinates y_j and y_{j+1} and the segments of the layers between these. Some of them are occupied by an item or a box in $\mathcal{I}_L \cup \mathcal{B}_H$ and some are not. We will use the not occupied layers to place the vertical items. We scan the area between y_j and y_{j+1} from bottom to top and fuse each set of contiguous unoccupied layers to a box for vertical items. Let $\mathcal{B}_{V,j}$ be the set of constructed boxes for the area between the coordinates y_j and y_{j+1}. Note that there can be at most $\mathcal{O}(1/\varepsilon\delta)$ of them.

Similar as for the horizontal items, we define configurations for the vertical items. However, instead of placing these items next to each other, we will stack the items inside a configuration for vertical items on top of each other. Note that in each optimal packing a vertical line through the packing intersects at most $1/\delta$ of these items and hence configurations should contain at most this number of items. We define a new set of vertical items called $\bar{\mathcal{I}}_V$. For each appearing item height $h \in \{h(i) | i \in \mathcal{I}_V\}$, the set $\bar{\mathcal{I}}_V$ contains one job of height h and width $\sum_{i \in \mathcal{I}_V, h(i)=j} w(i)$. To reduce the running time, we will schedule the jobs in the set $\bar{\mathcal{I}}_V$ fractionally instead of the original vertical items. Note that $|\bar{\mathcal{I}}_V| \leqslant \log_\varepsilon(1/\delta)/\varepsilon^2$ due to the rounding of the vertical items.

A configuration for vertical items is a multiset $C := \{a_{i,C} : i | i \in \bar{\mathcal{I}}_V\}$ such that $\sum_{i \in \bar{\mathcal{I}}_V} a_{i,C} \cdot h(i) \leqslant 1/\delta$ and we define its height as $h(C) := \sum_{i \in \bar{\mathcal{I}}_V} a_{i,C} \cdot h(i)$. Let $\mathcal{C}_V$ be the set of all these configurations and let $\mathcal{C}_{V,h}$ be the set of all configurations with height at most h. These configurations for vertical items are combined to hyper configurations which represent the distribution of vertical items in a vertical line through the packing. For each segment between two coordinates y_j and y_{j+1}, we define a configuration C_j as a tuple of configurations such that there is exactly one configuration for each of the boxes in $\mathcal{B}_{V,j}$, i.e., $C_j = (C \in \mathcal{C}_{V,h(b)} : b \in \mathcal{B}_{V,j})$. Let $\mathcal{C}_{V,j}$ be

the set of all configurations for the section between the coordinates y_j and y_{j+1}. We define $a_i(C)$ at the number of appearances of item $i \in \bar{\mathcal{I}}_V$ inside the configuration $C \in \mathcal{C}_{V,j}$. Note that the configurations for the boxes each have a maximum amount of vertical items they can contain and the sum of these numbers is bounded by $1/\delta$. Hence the total number of different configurations in $\mathcal{C}_{V,j}$ is bounded by $|\bar{\mathcal{I}}_V|^{1/\delta}$. To find fitting values for the y-coordinates the algorithm solves the following linear program:

$$y_0 = 0 \tag{8.12}$$

$$y_{|\mathcal{Y}|+1} = W \tag{8.13}$$

$$y_{j+1} - y_j = w_j \qquad \forall j \in \{0, \dots, |\mathcal{Y}|\} \tag{8.14}$$

$$y_{\pi(i,r)} - y_{\pi(i,l)} = w(i) \qquad \forall i \in \mathcal{I}_L \cup \mathcal{B}_H \tag{8.15}$$

$$\sum_{C \in \mathcal{C}_{V,j}} x_{C,j} = w_j \qquad \forall j \in \{0, \dots, |\mathcal{Y}|\} \tag{8.16}$$

$$\sum_{j \in \{0,\dots,|\mathcal{Y}|\}} \sum_{C \in \mathcal{C}_{V,j}} a_i(C) \cdot x_{C,j} = w(i) \qquad \forall i \in \bar{\mathcal{I}}_V \tag{8.17}$$

$$w_j \geqslant 0 \qquad \forall j \in \{0, \dots, |\mathcal{Y}| + 1\} \tag{8.18}$$

$$x_{C,j} \geqslant 0 \qquad \forall j \in \{0, \dots, |\mathcal{Y}| + 1\}, C \in \mathcal{C}_{V,j} \tag{8.19}$$

$$y_j \geqslant 0 \qquad \forall j \in \{0, \dots, |\mathcal{Y}| + 1\} \tag{8.20}$$

In this linear program there are three types of variables: x, y and w. The variables y_j for $j \in \{0, \dots, |\mathcal{Y}| + 1\}$ represent the values of the y-coordinates of the item and box borders in $\mathcal{I}_L \cup \mathcal{B}_H$, whereas y_0 represents the left border of the strip and $y_{|\mathcal{Y}|+1}$ represents the right border of the strip. The variables w_j for $j \in \{0, \dots, |\mathcal{Y}|\}$ represent the distance between the consecutive y-coordinates y_j and y_{j+1}. Last, the variables $x_{C,j}$ represent the width of the configuration C in box b which is positioned between y_j and y_{j+1}.

The first three constraints (8.12) to (8.14) ensure that the y-coordinates are positioned in the right order and that we use exactly the width of the strip. Furthermore, the variables w_j for the width between the y-coordinates are defined. The equation (8.15) ensures the y-coordinates of the items and boxes in $\mathcal{I}_L \cup \mathcal{B}_H$ are positioned such that their distance

equals the widths of the corresponding item. Equations (8.16) and (8.17) ensure that the vertical items are placed correctly. The first equation ensures that we do not use a to large width for the configurations inside the boxes while the second equation ensures that all the vertical items can be placed.

The total number of constraints is bounded by

$$\begin{aligned} &2|\mathcal{Y}| + 2 + |\mathcal{I}_L \cup \mathcal{B}_H| + |\bar{\mathcal{I}}_V| \\ &= \mathcal{O}(1/(\varepsilon\delta^2) + \log_\varepsilon(1/\delta)/\varepsilon^2) \\ &= \mathcal{O}(1/(\varepsilon\delta^2)), \end{aligned}$$

while the total number of variables is bounded by

$$\begin{aligned} &2|\mathcal{Y}| + 1 + \sum_{j=0}^{|\mathcal{Y}|+1} |\mathcal{C}_{V,j}| \\ &= \mathcal{O}((1/(\varepsilon\delta^2))((\log_\varepsilon(1/\delta)/\varepsilon^2)^{1/\delta})) \\ &= 2^{\mathcal{O}(1/\varepsilon\delta)}. \end{aligned}$$

Furthermore, all appearing values in the linear program are integer, the largest one on the left hand side is bounded by $1/\delta$ while the right hand side is bounded by W. We can solve this linear program by guessing the right set of at most $\mathcal{O}(1/(\varepsilon\delta^2))$ non-zero components and then solving the corresponding equation system using Gauß-Jordan elimination in $\mathcal{O}((2^{\mathcal{O}(1/\varepsilon\delta)})^{\mathcal{O}(1/(\varepsilon\delta^2))} \cdot (1/(\varepsilon\delta^2))^3) = 2^{\mathcal{O}(1/(\varepsilon^2\delta^3))}$.

After we have found such a solution, we fix the values for the variables y_j and w_j for each $j \in \{1, \ldots, |\mathcal{Y}| + 1\}$ and find a basic solution to the linear program consisting just of the equations (8.16), (8.17), and (8.19). Such a basic solution has at most $|\bar{\mathcal{I}}_V| + |\mathcal{Y}| \in \mathcal{O}(1/(\varepsilon\delta^2))$ non zero components and hence uses at most this number of configurations.

In the very end of the algorithm, these configurations are filled (fractionally) with the rounded vertical items analogously as small wide jobs items are filed into their configurations, see Section 8.3.3. Since each configuration contains at most $1/\delta$ items and we use at most $\mathcal{O}(1/(\varepsilon\delta^2))$ of them, there are at most $\mathcal{O}(1/(\varepsilon\delta^3))$ fractionally placed vertical items which have a total width of at most $\mathcal{O}(\mu W/(\varepsilon\delta^3))$. Since $\mu \leqslant \delta^3\varepsilon/(cx)$ for a large

enough constant x, it holds that the total width of the discarded items is smaller than $W/(2c)$. These items are placed inside the additional cluster C_2, adding at most $p_{\max}$ to its packing height.

8.5.4 Placing the Small Items

Note that the configurations for vertical and horizontal items might be smaller in height or width as the box they are placed inside, i.e., if a configuration C for vertical items is placed in side a box $b \in \mathcal{B}_{V,j}$ there is a box of free area of width $X_{C,b,j}$ and height $h(b) - h(C)$. We will use this area to place the small items. The total free area of this kind has to have the size of area$(\mathcal{I}_S)$, since the configurations contain exactly the total area of the corresponding items and the total area of all items is at most $(1+2\varepsilon)(1+\varepsilon)TW$ while the packing has a height of at least $(1+2\varepsilon)(1+\varepsilon)T$.

Since we use at most $\mathcal{O}(1/(\varepsilon\delta^3))$ configurations for vertical items and at most $|\mathcal{B}_H| + |\bar{\mathcal{I}}_H| = \mathcal{O}(1/(\varepsilon\delta^2))$ configurations for horizontal items, there are at most $\mathcal{O}(1/(\varepsilon\delta^3))$ boxes for small items. We call the set of these boxes $\mathcal{B}_S$.

Lemma 8.16. *We can place the small items inside the $\mathcal{O}(1/(\varepsilon\delta^3))$ boxes $\mathcal{B}_S$ and one additional box of width W and height $2\varepsilon T + \mu T$.*

Proof. Remember that the total area of the boxes is at least area$(\mathcal{I}_S)$. The algorithm first sorts the small items by height in $\mathcal{O}(n + \log(n)/\varepsilon^2)$ time since the small items have at most $\mathcal{O}(\log(n)/\varepsilon^2)$ different sizes. Afterward it considers the boxes for the small items $\mathcal{B}_S$ one by one and fills the small items inside them using the NFDH algorithm. If an item does not fit inside the considered box, because the item is to wide or has a to large height, the algorithm is finished with this box and considers the next. All the items that cannot be placed inside the boxes $\mathcal{B}_S$ are placed inside the newly introduced box of width W and height $2\varepsilon T + \mu T$.

Let us consider the boxes next to the configurations and the free area inside them. Let B be such a box. In B there is a free area of at most $\mu W \cdot h(B)$ on one side of B since the small items have a width of at most μW. Additionally, there can be free area of at most $\mu T \cdot w(B)$ on the top of the box since the items have a height of at least μT. Lastly there can be

free area between the items. However, as indirectly shown by Coffman et al. in [22] in the proof of Lemma 2.2, the free area provoked this way over all the boxes is bounded by $\mu T \cdot W$ since the items have a maximal height of at most μT and the boxes have a maximal width of at most W. In total the free area inside the boxes $\mathcal{B}_S$ is bounded by $\mu TW + \mu T \sum_{B \in \mathcal{B}_S} w(B) + \mu W \sum_{B \in \mathcal{B}_S} h(B) \leqslant \mu TW \cdot \mathcal{O}(1/(\varepsilon\delta^3))$. Since it holds that $\mu \leqslant \varepsilon^2\delta^3/x$ for a suitable large constant x, the total area of the non placed small items has to be bounded by εTW. Using Lemma 2.2, we can place these non placed items with a total height of at most $2\varepsilon T + \mu T$ inside the extra box. □

8.5.5 Packing medium sized items

To place the medium sized items, we partition them into two sets, $\mathcal{I}_{M,V}$ which contains all the items taller than $2\varepsilon T$ and $\mathcal{I}_{M,S} := \mathcal{I}_M \backslash \mathcal{I}_{M,V}$. Since the total area of the medium sized items is bounded by $\varepsilon TW/c$, the total width of the items in $\mathcal{I}_{M,V}$ is bounded by $W/(2c)$. Hence, we can place all these items into the cluster C_2 next to the discarded vertical items.

The jobs in $\mathcal{I}_{M,V}$ have a height of at most $2\varepsilon T$ and an area of at most εTW. Hence by Lemma 2.2, when using the NDFH algorithm to place these items, we add at most $4\varepsilon T$ to the packing height.

Note that in the case of $\lfloor W/c \rfloor \leqslant W/c \leqslant x/\varepsilon^2\delta^3$ it holds that $\delta W \leqslant 1$ and hence the set $\mathcal{V}$ and the set $\mathcal{I}_{M,V}$ will be empty. As a consequence the container C_2 will not contain any item.

8.5.6 Summary of the algorithm

In the following, we summarize the steps of the algorithm and give a short overview of the running time. An overview of the generated packing can be found in Figure 8.8.

1. In the first step of the algorithm, we perform the simplification steps. We define $T := \max\{h_{\max}, (\sum_{i \in \mathcal{I}} h(i)w(i))/W\}$, find the correct values for δ and μ as described in Lemma 8.11, and partition the set of items into $\mathcal{L}$, $\mathcal{V}$, $\mathcal{H}$, $\mathcal{S}$, and $\mathcal{M}$ accordingly. Afterward, we round the heights and the widths of the items. First, we round the height of the items to multiples of $\varepsilon T/n$ and scale the items, such that they have heights in

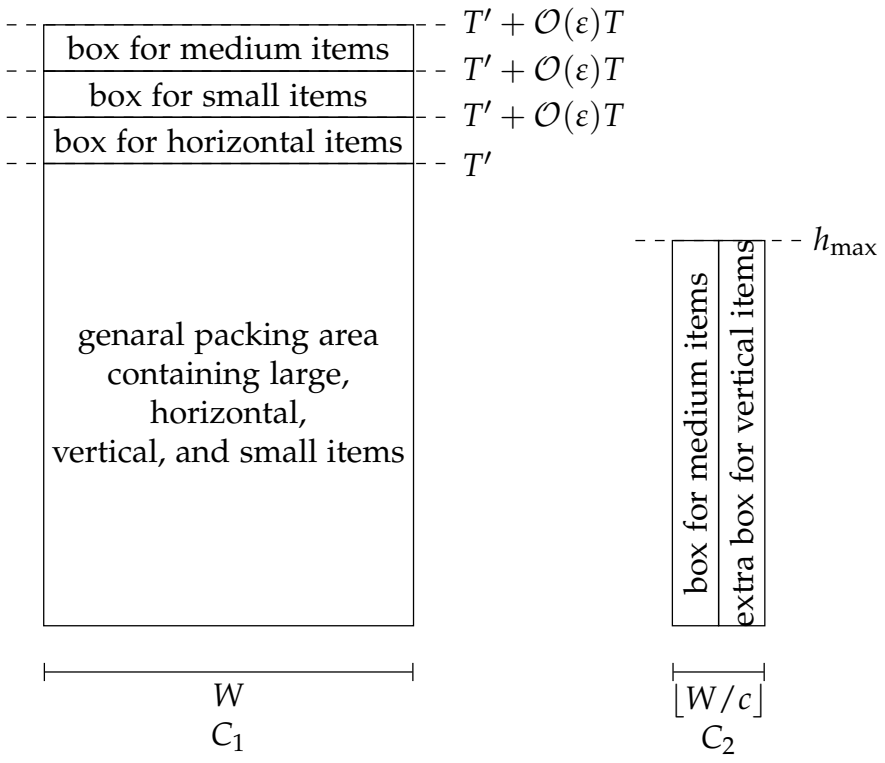


Figure 8.8. Overview of the structure of the generated packing

$\{1, \ldots, n/\varepsilon\} \subseteq \mathbb{N}$ and scale T accordingly such that $T = n/\varepsilon$. Next, we scale the instance and T again with $1/\varepsilon\delta$ and use Lemma 3.2 to round heights of the items in $\mathcal{L} \cup \mathcal{V}$, such that we can assume that they start at multiples of $1/\varepsilon\delta T$. Furthermore, introduce the set of rounded items $\bar{\mathcal{H}}$ using Lemma 8.13.

2. In the next step, we do a binary search over all the possible numbers of layers $L \in [1/(\varepsilon\delta), 5/(\varepsilon\delta)] \cap \mathbb{N}$. Let T' be the currently considered number of layers. For this number of layers, we try to find a packing by performing the following steps.

3. For each guess of the set $\mathcal{W}_B$ and each guess of y-coordinates and permutation for boxes and large items: try to solve the configuration linear program LP_{small} to place the horizontal items. If this is not possible try the next guess otherwise try to solve the LP to find the correct positions

for the boxes, large items, and vertical items. If this LP is solvable, save the guess and LP solutions and try the next smaller value for T' in binary search fashion, otherwise try the next guess. If all guesses fail try the next larger value for T' in binary search fashion.

4. Afterward, use the saved guess and LP solutions to assign the corresponding items. First, we revert the scaling of the items and scale the solution and guess accordingly. Then, we place the large, vertical, and horizontal items inside the guess as described in Section 8.5.3. Afterward, place the small items inside the resulting boxes for small items as described in Section 8.5.4. Finally, we place the medium sized items as described in Section 8.5.5.

The step 1 takes $\mathcal{O}(n \log(1/\varepsilon) + c/\varepsilon)$ operations: The set of items needs to be enumerated once to find T, i.e., its can be found in $\mathcal{O}(n)$. The correct values for δ and μ can be found in $\mathcal{O}(n + c/\varepsilon)$ and the corresponding partition can be found in $\mathcal{O}(n)$. The scaling and rounding of the item heights can be done in $\mathcal{O}(n)$. Finally the rounding of the item widths can be done in $\mathcal{O}(n \log(1/\varepsilon))$.

The binary search described in Step 2 can be done in $\mathcal{O}(\log(1/(\varepsilon\delta)))$. For each of the values T' given by the binary search framework, the number of guesses can be bounded by $\mathcal{O}((\log(1/\delta)/\varepsilon)^{\mathcal{O}(1/(\varepsilon\delta^2))})$ for $\mathcal{W}_B$, $(1/\varepsilon\delta)^{\mathcal{O}(1/(\varepsilon\delta^2))}$ for the y-coordinates, and $(1/(\varepsilon\delta^2))^{\mathcal{O}(1/(\varepsilon\delta^2))}$ for the permutation of boxes and large items. The resulting LP can be solved in $2^{\mathcal{O}(1/(\varepsilon^2\delta^3))}$. Therefore the total running time of steps 2 and 3 can be summarized as

$$\begin{aligned}\mathcal{O}(\log(1/(\varepsilon\delta))) \cdot \mathcal{O}((\log(1/\delta)/\varepsilon)^{\mathcal{O}(1/(\varepsilon\delta^2))}) \\ \cdot (1/\varepsilon\delta)^{\mathcal{O}(1/(\varepsilon\delta^2))} \cdot (1/(\varepsilon\delta^2))^{\mathcal{O}(1/(\varepsilon\delta^2))} \cdot 2^{\mathcal{O}(1/(\varepsilon^2\delta^3))} \\ \leqslant 2^{\mathcal{O}(1/\varepsilon^2\delta^3)}.\end{aligned}$$

In the final step, we place the original items inside the packing. The placement of large, vertical, and horizontal items can be done in $\mathcal{O}(n + 1/(\varepsilon\delta^3))$ since there are at most $1/(\varepsilon\delta^3)$ places for vertical and horizontal items. To place the small items, we use the NFDH algorithm and hence have a running time of at most $\mathcal{O}(1/(\varepsilon\delta^3) + n + \log(n)/\varepsilon^2)$ since the items

have at most $\log(n)/\varepsilon^2$ sizes and are placed inside at most $\mathcal{O}(1/(\varepsilon\delta^3))$ boxes. The medium sized items can be placed in at most $\mathcal{O}(n + 1/\varepsilon^2)$ since they have at most $\mathcal{O}(1/\varepsilon^2)$ (possible) different sizes. Hence the total running time of the algorithm is bounded by

$$\begin{aligned}&\mathcal{O}(n\log(1/\varepsilon) + c/\varepsilon + 2^{\mathcal{O}(1/\varepsilon^2\delta^3)} + 1/(\varepsilon\delta^3) + n + \log(n)/\varepsilon^2)\\ &\leqslant \mathcal{O}(n\log(1/\varepsilon) + \log(n)/\varepsilon^2) + 2^{(c/\varepsilon)^{3^{\mathcal{O}(c/\varepsilon)}}}.\end{aligned}$$

As a consequence, we end up with a running time of $\mathcal{O}(n\log(1/\varepsilon) + \log(n)/\varepsilon^2) + 2^{(1/\varepsilon)^{3^{\mathcal{O}(1/\varepsilon)}}}$ for the AEPTAS because we can choose $c = 1$ in this case.

8.6 A Note on Single Resource Constraint Multiple Cluster Scheduling (SRCMCS)

In this section, we shortly discuss how to generate the algorithm which produced a schedule on C_1 and C_2 as needed for Lemma 8.5. This algorithm together with the partitioning described in Lemma 8.5 makes the 2-approximation for Single Resource Constraint Multiple Cluster Scheduling (SRCMCS). More precisely, in this section, we discuss how we can generate the algorithm from the following lemma.

Lemma 8.17. *There exists an algorithm, that finds for each instance $I = (\mathcal{J}, m)$ of the Single Resource Constraint Scheduling problem a schedule on two clusters C_1 and C_2 with m machines, such that the makespan on C_1 is bounded by $(1+\varepsilon)$OPT and on C_2 by p_{max}. This algorithm has a running time that is bounded by $\mathcal{O}(n\log(1/\varepsilon)) + \mathcal{O}_\varepsilon(1)$.*

Consider the AEPTAS from Section 7.4. Note that in this algorithm the extra $p_{\max}$ is added by a set of jobs with is discarded from the schedule and added later on. Hence when removing this set again and placing it inside the extra cluster C_2, the first schedule will have a length of at most $(1+\varepsilon)$OPT and the cluster C_2 will have a makespan of at most $p_{\max} \leqslant$ OPT.

Note that for $N = 2$ we cannot use the partitioning technique from Section 8.2.2, because we are not aware of an algorithm for Single Resource

Constraint Scheduling that places jobs such that the makespan is bounded by at most two times the total area of the jobs.

8.7 Conclusion

In this paper, we presented an algorithm for each of the problems Multiple Cluster Scheduling (MCS), Multiple Strip Packing (MSP) and Single Resource Constraint Multiple Cluster Scheduling (SRCMCS) with best possible absolute approximation ratio of 2 and best possible running time $\mathcal{O}(n)$ for the case $N \geqslant 3$. Still open remains the question if for the case $N = 2$ the running time of $\mathcal{O}(n \log(n))$ or $\mathcal{O}(n \log^2(n)/(\log(\log(n))))$ for MCS and MSP respectively can be improved to $\mathcal{O}(n)$. Furthermore, to find an algorithm with approximation ratio 2 for Single Resource Constraint Multiple Cluster Scheduling (SRCMCS) and the case $N \leqslant 2$ would be interesting.

Additionally, we presented a truly fast algorithm for Multiple Cluster Scheduling (MCS) with running time $\mathcal{O}(n \log(n))$ that does not have any hidden constants. Since the running time of the $\mathcal{O}(n)$ algorithm hides large constants, it would be interesting to improve the running time of the underlying $AEPTAS$ or even to find a faster asymptotic algorithm with approximation guarantee $(5/4)\mathrm{OPT} + p_{\max}$.

Chapter 9

EPTAS Results for Scheduling with Setup Times

Integer linear programs of configurations, or configuration IPs, are a classical tool in the design of algorithms for scheduling and packing problems, where a set of items has to be placed in multiple target locations. Herein a configuration describes a possible placement on one of the target locations, and the IP is used to chose suitable configurations covering the items. We give an augmented IP formulation, which we call the module configuration IP. It can be described within the framework of n-fold integer programming and therefore be solved efficiently. As an application, we consider scheduling problems with setup times, in which a set of jobs has to be scheduled on a set of identical machines, with the objective of minimizing the makespan. For instance, we investigate the case that jobs can be split and scheduled on multiple machines. However, before a part of a job can be processed an uninterrupted setup depending on the job has to be paid. For both of the variants that jobs can be executed in parallel or not, we obtain an efficient polynomial time approximation scheme (EPTAS) of running time $f(1/\varepsilon) \times \text{poly}(|I|)$ with a single exponential term in f for the first and a double exponential one for the second case. Previously, only constant factor approximations of $5/3$ and $4/3 + \varepsilon$ respectively were known. Furthermore, we present an EPTAS for a problem where classes of (non-splittable) jobs are given, and a setup has to be paid for each class of jobs being executed on one machine.

The results described in this chapter were partially published in [52].

9.1 Introduction

In this chapter, we present an augmented formulation of the classical integer linear program of configurations (configuration IP) and demonstrate its use in the design of efficient polynomial time approximation schemes for scheduling problems with setup times. Configuration IPs are widely used in the context of scheduling or packing problems, in which items have to be distributed to multiple target locations. The configurations describe possible placements on a single location, and the integer linear program (IP) is used to choose a proper selection covering all items. Two fundamental problems, for which configuration IPs have prominently been used, are Bin Packing and Scheduling on Identical Machines. For Bin Packing, the configuration IP was introduced as early as 1961 by Gilmore and Gomory [37], and the recent results for both problems typically use configuration IPs as a core technique, see, e.g., [38, 53]. In the present work, we consider scheduling problems and therefore introduce the configuration IP in more detail using the example of Scheduling on Identical Machines.

Configuration IP for Scheduling on Identical Machines. In the problem Scheduling on Identical Machines (SIM), a set $\mathcal{J}$ of n jobs is given together with processing times $p(j)$ for each job $j \in \mathcal{J}$ and a number m of identical machines. The objective is to find a schedule $\sigma : \mathcal{J} \to [m]$ such that the makespan is minimized, that is, the latest finishing time of any job $C_{\max}(\sigma) = \max_{i \in [m]} \sum_{j \in \sigma^{-1}(i)} p(j)$. For a given makespan bound, the configurations may be defined as multiplicity vectors indexed by the occurring processing times, where the overall length of the chosen processing times does not violate the bound. The configuration IP is then given by variables x_C for each configuration C; constraints ensuring that there is a machine for each configuration, i.e., $\sum_C x_C = m$; and further constraints due to which the jobs are covered, i.e., $\sum_C C_p x_C = |\{j \in \mathcal{J} \mid p(j) = p\}|$ for each processing time p. In combination with certain simplification techniques, this type of IP is often used in the design of PTAS. In the context of Scheduling on Identical Machines, the aforementioned simplification techniques can be used to guess the target makespan T of the given instance; to upper bound the cardinality of the set of processing times P by a constant (depending in $1/\varepsilon$); and to lower bound the processing times in size such that they are

within a constant factor of the makespan T (see, e.g., [5, 53]). Hence, only a constant number of configurations is needed, yielding an integer program with a constant number of variables. Integer programs of that kind can be efficiently solved using the classical algorithm by Lenstra and Kannan [82, 72], yielding a PTAS for Scheduling on Identical Machines. Here, the error of $(1+\varepsilon)$ in the quality of the solution is due to the simplification steps, and the scheme has a running time of the form $f(1/\varepsilon) \times (|I|)^{\mathcal{O}(1)}$, where $|I|$ denotes the input size, and f some computable function. Remember, a PTAS with this property is called *efficient* (EPTAS). Note that for a regular PTAS a running time of the form $|I|^{f(1/\epsilon)}$ is allowed. It is well known, that Scheduling on Identical Machines is strongly NP-hard, and therefore there is no optimal polynomial time algorithm, unless P = NP, and also a so-called *fully polynomial* PTAS (FPTAS) – which is an EPTAS with a polynomial function f – cannot be hoped for.

Machine Scheduling with Classes. The configuration IP is used in a wide variety of approximation schemes for Scheduling on Identical Machines problems [5, 53]. However, the approach often ceases to work for scheduling problems in which the jobs have to fulfill some additional requirements, like, for instance, class dependencies. A problem emerging, in this case, is that the additional requirements have to be represented in the configurations, resulting in a super-constant number of variables in the IP. We elaborate on this using a concrete example: Consider the variant of Scheduling on Identical Machines in which the jobs are partitioned into K setup classes. For each job j a class k_j is given and for each class k a setup time $s(k)$ has to be paid on a machine, if a job belonging to that class is scheduled on it, i.e., $C_{\max}(\sigma) = \max_{i\in[m]} \left(\sum_{j\in\sigma^{-1}(i)} p(j) + \sum_{k\in\{k_j \mid j\in\sigma-1(i)\}} s(k)\right)$. With some effort, simplification steps similar to the ones for Scheduling on Identical Machines can be applied. In the course of this, the setup times as well can be bounded in number and guaranteed to be sufficiently big [55]. However, it is not hard to see that the configuration IP still cannot be trivially extended, while preserving its solvability. For instance, extending the configurations with multiplicities of setup times will not work, because then we have to make sure that a configuration is used for a fitting

subset of classes, creating the need to encode class information into the configurations or introduce other class dependent variables.

Module Configuration IP. Our approach to deal with the class dependencies of the jobs is to cover the job classes with so-called modules and cover the modules in turn with configurations in an augmented IP called the module configuration IP (MCIP). In the setup class model, for instance, the modules may be defined as combinations of setup times and configurations of processing times, and the actual configurations as multiplicity vectors of module sizes. The number of both the modules and the configurations will typically be bounded by a constant. To cover the classes by modules, each class is provided with its own set of modules, that is, there are variables for each pair of class and module. Since the number of classes is part of the input, the number of variables in the resulting MCIP is super-constant, and therefore the algorithm by Lenstra and Kannan [82, 72] is not the proper tool for the solving of the MCIP. However, the MCIP has a certain simple structure: The mentioned variables are partitioned into uniform classes each corresponding to the set of modules, and for each class, the modules have to do essentially the same – cover the jobs of the class. Utilizing these properties, we can formulate the MCIP in the framework of n-fold integer programms – a class of IPs whose variables and constraints fulfill certain uniformity requirements. In 2013 Hemmecke, Onn, and Romanchuk [46] showed that n-fold IPs can be efficiently solved, and very recently both Eisenbrand, Hunkenschröder and Klein [29] and independently Koutecký, Levin and Onn [81] developed algorithms with greatly improved running times for the problem. For a detailed description of the MCIP, the reader is referred to Section 9.3. In Figure 9.1 the basic idea of the MCIP is visualized.

Using the MCIP, we are able to formulate an EPTAS for Scheduling on Identical Machines in the setup class model described above. Before, only a regular PTAS with running time $nm^{\mathcal{O}(1/\varepsilon^5)}$ was known [55]. To the best of our knowledge, this is the first use of n-fold integer programming in the context of approximation algorithms.

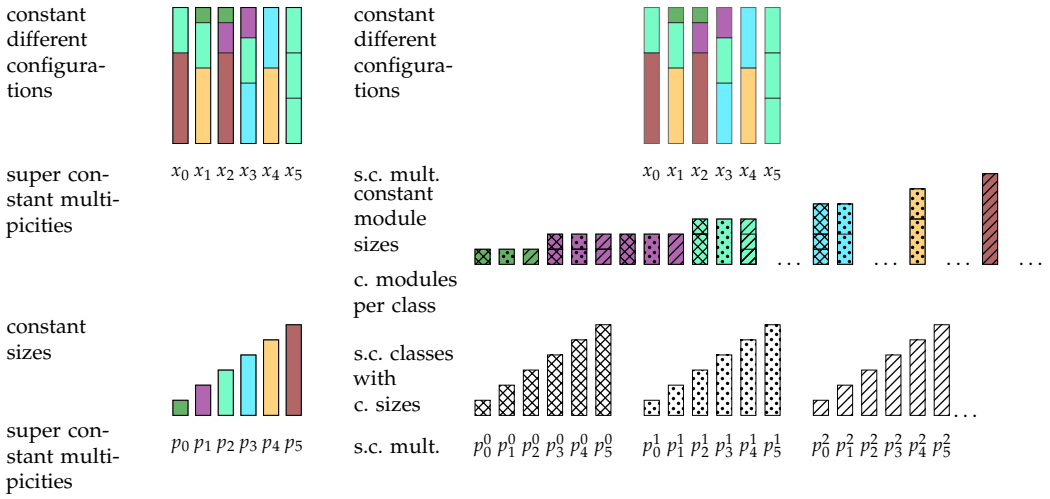


Figure 9.1. On the left, there is a schematic representation of the configuration IP. There are constant different sizes each occurring a super-constant number of times. The sizes are directly mapped to configurations. On the right, there is a schematic representation of the MCIP. There is a super-constant number of classes, each containing a constant number of sizes which have super-constant multiplicities. The elements from the class are mapped to a constant number of different modules, which have a constant number of sizes. These module sizes are mapped to configurations.

Results and Methodology. To show the conceptual power of the MCIP, we utilize it for two more problems: The *splittable* and the *preemptive* setup model of Scheduling on Identical Machines. In both variants for each job $j \in \mathcal{J}$, a setup time $s(j)$ is given. Each job may be partitioned into multiple parts that can be assigned to different machines, but before any part of the job can be processed the setup time has to be paid. In the splittable model, job parts belonging to the same job can be processed in parallel, and therefore beside the partition of the jobs, it suffices to find an assignment of the job parts to machines. This is not the case for the preemptive model, in which additionally a starting time for each job part has to be found, and two parts of the same job may not be processed in parallel. In 1999 Schuurman and Woeginger [97] presented a polynomial time algorithm for the preemptive model with approximation guarantee $4/3 + \varepsilon$, and for the splittable case a guarantee of $5/3$ was achieved by Chen, Ye and Zhang

[19]. These are the best known approximation guarantees for the problems at hand. We show that solutions arbitrarily close to the optimum can be found in polynomial time:

Theorem 9.1. *There is an efficient PTAS with running time* $2^{f(1/\varepsilon)}\text{poly}(|I|)$ *for minimum makespan scheduling on identical parallel machines in the setup-class model, as well as in the preemptive and splittable setup models.*

More precisely, we get a running time of $2^{\mathcal{O}(1/\varepsilon^3 \log^4 1/\varepsilon)} K^2 nm \log(Km)$ in the setup class model, $2^{\mathcal{O}(1/\varepsilon^2 \log^3 1/\varepsilon)} n^2 \log^3(nm)$ in the splittable, and $2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}} n^2 m \log m \log(nm)$ in the preemptive model. Note, that all three problems are strongly NP-hard, due to trivial reductions from Scheduling on Identical Machines, and our results are therefore in some sense best possible.

Summing up, the main achievement of this work is the development of the module configuration IP and its application in the development of approximation schemes. Up to now, EPTAS or even PTAS results seemed out of reach for the considered problems, and for the preemptive model we provide the first improvement in 20 years. The simplification techniques developed for the splittable and preemptive model in order to employ the MCIP are original and in the latter case quite elaborate, and therefore interesting by themselfs. Furthermore, we expect the MCIP to be applicable to other packing and scheduling problems as well, in particular for variants of Scheduling on Identical Machines and Bin Packing with additional class depended constraints. On a more conceptual level, we gave a first demonstration of the potential of n-fold integer programming in the theory of approximation algorithms, and hope to inspire further studies in this direction.

We conclude this paragraph with a more detailed overview of our results and their presentation. For all three EPTAS results we employ the classical dual approximation framework by Hochbaum and Shmoys [49] to get a guess of the makespan T. This approach is introduced in Section 9.2 together with n-fold IPs and formal definitions of the problems. In the following section, we develop the module configuration IP, in its basic form and argue that it is indeed an n-fold IP. The EPTAS results follow the same basic approach described above for Scheduling on Identical Machines: We

find a schedule for a simplified instance via the MCIP and transform it into a schedule for the original one. The simplification steps typically include rounding of the processing and setup times using standard techniques, as well as the removal of certain jobs, which later can be reinserted via carefully selected greedy procedures. For the splittable and preemptive model, we additionally have to prove that schedules with a certain simple structure exist, and in the preemptive model, the MCIP has to be extended. In Section 9.4 the basic versions of the EPTAS are presented and in Section 9.5, some improvements of the running time for the splittable and the setup class model are discussed.

Related work. For an overview on n-fold IPs and their applications, we refer to the book by Onn [93]. There have been recent applications of n-fold integer programming to scheduling problems in the context of parameterized algorithms: Knop and Koutecký [79] showed, among other things, that the problem of makespan minimization on unrelated parallel machines, where the processing times are dependent on both jobs and machines, is fixed-parameter tractable with respect to the maximum processing time and the number of distinct machine types. This was generalized to the parameters maximum processing time and rank of the processing time matrix by Chen et al. [21]. Furthermore, Knop, Koutecký and Mnich [80] provided an improved algorithm for a special type of n-fold IPs yielding improved running times for several applications of n-fold IPs including results for scheduling problems.

For work on the considered scheduling problems, we refer to section 2.5.

9.2 Preliminaries

In the following, we establish some concepts and notations, formally define the considered problems, and outline the dual approximation approach by Hochbaum and Shmoys [49], as well as n-fold integer programs.

For any integer n, we denote the set $\{1, \dots, n\}$ by $[n]$; we write $\log(\cdot)$ for the logarithm with basis 2; and we will usually assume that some instance I of the problem considered in the respective context is given

together with an accuracy parameter $\varepsilon \in (0,1)$ such that $1/\varepsilon$ is an integer. Furthermore, for any two sets X, Y we write Y^X for the set of functions $f : X \to Y$. If X is finite, we say that Y is indexed by X and sometimes denote the function value of f for the argument $x \in X$ by f_x.

Furthermore, all the following algorithms use the dual approximation framework introduced by Hochbaum and Shmoys [49]. Note that for all of the considered problems constant approximation algorithms are known, and the sum of all processing and setup times is a trivial m-approximation. Hence, as described in Section 3.1, we will always assume that a target makespan T is given. Moreover, we assume that the setup times and in the preemptive and setup class cases also the processing times are bounded by T, because otherwise we can reject T immediately.

***n*-fold Integer Programs.** We briefly define n-fold integer programs (IP) following the notation of [46] and [79] and state the main algorithmic result needed in the following. Let $n, r, s, t \in \mathbb{Z}_{>0}$ be integers and A be an integer $((r+ns) \times nt)$-matrix of the following form:

$$A = \begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}$$

The matrix A is the so-called n-fold product of the bimatrix $\binom{A_1}{A_2}$, with A_1 an $r \times t$ and A_2 an $s \times t$ matrix. Furthermore, let $w, \ell, u \in \mathbb{Z}^{nt}$ and $b \in \mathbb{Z}^{r+ns}$. Then the n-fold integer programming problem is given by

$$\min\{wx \mid Ax = b, \ell \leqslant x \leqslant u, x \in \mathbb{Z}^{nt}\}.$$

We set Δ to be the maximum absolute value occurring in A. Up to recently the best known algorithm for solving n-fold IPs was due to Hemmecke, Onn and Romanchuk [46]:

Theorem 9.2. *Let φ be the encoding length of w, b, ℓ, u and Δ. The n-fold integer programming problem can be solved in time $\mathcal{O}(\Delta^{3t(rs+st+r+s)} n^3 \varphi)$, when r, s and t are fixed.*

However, in 2018 both Eisenbrand, Hunkenschröder and Klein [81] and independently Koutecký, Levin and Onn [81] developed algorithms with improved and very similar running times. We state a variant due to Eisenbrand et al. that is adapted to our needs:

Theorem 9.3. *Let φ be the encoding length of the largest number occurring in the input, and $\Phi = \max_i(u_i - \ell_i)$. The n-fold integer programming problem can be solved in time $(rs\Delta)^{\mathcal{O}(r^2s+rs^2)}t^2n^2\varphi\log(\Phi)\log(nt\Phi)$.*

The variables x can naturally be partitioned into *bricks* $x^{(q)}$ of dimension t for each $q \in [n]$ such that $x = (x^{(1)}, \dots x^{(n)})$. Furthermore, we denote the constraints corresponding to A_1 as *globally uniform* and the ones corresponding to A_2 as *locally uniform*. Hence, r is the number of globally and s the number of locally uniform constraints (ignoring their n-fold duplication); t the *brick size* and n the *brick number*.

9.3 Module Configuration IP

In this section, we state the configuration IP for the problem Scheduling on Identical Machines; introduce a basic version of the module configuration IP (MCIP) that is already sufficiently general to work for both the splittable and setup class model; and lastly show that the configuration IP can be expressed by the MCIP in multiple ways. Before that, however, we formally introduce the concept of *configurations*.

Given a set of objects A, a configuration C of these objects is a vector of multiplicities indexed by the objects, i.e., $C \in \mathbb{Z}^A_{\geqslant 0}$. For given sizes $\Lambda(a)$ of the objects $a \in A$, the size $\Lambda(C)$ of a configuration C is defined as $\sum_{a\in A} C_a\Lambda(a)$. Moreover, for a given bound B, we define $\mathcal{C}_A(B)$ to be the set of configurations of A that are bounded in size by B, that is, $\mathcal{C}_A(B) = \{C \in \mathbb{Z}^A_{\geqslant 0} \mid \Lambda(C) \leqslant B\}$.

Configuration IP. We give a recollection of the configuration IP for the problem Scheduling on Identical Machines. Let P be the set of distinct processing times for some instance I with multiplicities n_p for each $p \in P$, meaning, I includes exactly n_p jobs with processing time p. The size $\Lambda(p)$ of a processing time p is given by itself. Furthermore, let T be a guess

of the optimal makespan. The configuration IP for I and T is given by variables $x_C \geqslant 0$ for each $C \in \mathcal{C}_P(T)$ and the following constraints:

$$\sum_{C \in \mathcal{C}_P(T)} x_C = m \tag{9.1}$$

$$\sum_{C \in \mathcal{C}_P(T)} C_p x_C = n_p \qquad \forall p \in P \tag{9.2}$$

Due to constraint (9.1), exactly one configuration is chosen for each machine, while (9.2) ensures that the correct number of jobs or job sizes is covered.

Module Configuration IP. Let $\mathcal{B}$ be a set of basic objects (e.g. jobs or setup classes) and let there be D integer values $B_1, \ldots, B_D$ for each basic object $B \in \mathcal{B}$ (e.g. processing time or numbers of different kinds of jobs). Our approach is to cover the basic objects with so-called *modules* and in turn cover the modules with configurations. Depending on the context, modules correspond to batches of jobs or job piece sizes together with a setup time and can also encompass additional information like a starting time. Let $\mathcal{M}$ be a set of such modules. In order to cover the basic objects, each module $M \in \mathcal{M}$ also has D integer values $M_1, \ldots, M_D$. Furthermore, each module M has a size $\Lambda(M)$ and a set of eligible basic objects $\mathcal{B}(M)$. The latter is needed because not all modules are compatible with all basic objects, e.g., because they do not have the right setup times. The configurations are used to cover the modules, however, it typically does not matter which module exactly is covered, but rather which size the module has. Let H be the set of distinct module sizes, i.e., $H = \{\Lambda(M) \mid M \in \mathcal{M}\}$, and for each module size $h \in H$ let $\mathcal{M}(h)$ be the set of modules with size h. We consider the set $\mathcal{C}$ of configurations of module sizes which are bounded in size by a guess of the makespan T, i.e., $\mathcal{C} = \mathcal{C}_H(T)$. In the preemptive case configurations need to additionally encompass information about starting times of modules, and therefore the definition of configurations will be slightly more complicated in that case.

Since we want to chose configurations for each machine, we have variables x_C for each $C \in \mathcal{C}$ and constraints corresponding to (9.1). Furthermore, we chose modules with variables y_M for each $M \in \mathcal{M}$ and because we want to cover the chosen modules with configurations, we have some

analogue of constraint (9.2), say $\sum_{C \in \mathcal{C}(T)} C_h x_C = \sum_{M \in \mathcal{M}(h)} y_M$ for each module size $h \in H$. It turns out, however, that to properly cover the basic objects with modules, we need the variables y_M for each basic object, and this is were n-fold IPs come into play: The variables stated so far form a brick of the variables of the n-fold IP and there is one brick for each basic object, that is, we have for each $B \in \mathcal{B}$, variables $x_C^{(B)}$ for each $C \in \mathcal{C}$, and $y_M^{(B)}$ for each $M \in \mathcal{M}$. Using the upper bounds of the n-fold model, variables $y_M^{(B)}$ are set to zero, if B is not eligible for M; and we set the lower bounds of all variables to zero. Sensible upper bounds for the remaining variables will be typically clear from context. Besides that, the module configuration integer program MCIP (for $\mathcal{B}$, $\mathcal{M}$ and $\mathcal{C}$) is given by:

$$\sum_{B \in \mathcal{B}} \sum_{C \in \mathcal{C}} x_C^{(B)} = m \tag{9.3}$$

$$\sum_{B \in \mathcal{B}} \sum_{C \in \mathcal{C}(T)} C_h x_C^{(B)} = \sum_{B \in \mathcal{B}} \sum_{M \in \mathcal{M}(h)} y_M^{(B)} \qquad \forall h \in H \tag{9.4}$$

$$\sum_{M \in \mathcal{M}} M_d y_M^{(B)} = B_d \qquad \forall B \in \mathcal{B}, d \in [D] \tag{9.5}$$

It is easy to see that the constraints (9.3) and (9.4) are globally uniform. They are the mentioned adaptations of (9.1) and (9.2). The constraint (9.5), on the other hand, is locally uniform and ensures that the basic objects are covered.

Note that, while the duplication of the configuration variables does not carry meaning, it also does not upset the model: Consider the modified MCIP that is given by not duplicating the configuration variables. A solution $(\tilde{x}, \tilde{y})$ for this IP gives a solution (x, y) for the MCIP by fixing some basic object B^*, setting $x_C^{(B^*)} = \tilde{x}_C$ for each configuration C, setting the remaining configuration variables to 0, and copying the remaining variables. Given a solution (x, y) for the MCIP, on the other hand, gives a solution for the modified version $(\tilde{x}, \tilde{y})$ by setting $\tilde{x}_C = \sum_{B \in \mathcal{B}} x_C^B$ for each configuration C. Summarizing we get:

Observation 9.4. The MCIP is an n-fold IP with brick-size $t = |\mathcal{M}| + |\mathcal{C}|$, brick number $n = |\mathcal{B}|$, $r = |H| + 1$ globally uniform and $s = D$ locally uniform constraints.

Moreover, in all the considered applications we will minimize the overall size of the configurations, i.e., $\sum_{B\in\mathcal{B}}\sum_{C\in\mathcal{C}}\Lambda(C)x_C^{(B)}$. This will be required, because in the simplification steps of our algorithms some jobs are removed and have to be reinserted later, and we therefore have to make sure that no space is wasted.

First Example. We conclude the section by pointing out several different ways to replace the classical configuration IP for scheduling on identical machines with the MCIP, thereby giving some intuition for the model. The first possibility is to consider the jobs as the basic objects and their processing times as their single value ($\mathcal{B} = \mathcal{J}$, $D = 1$); the modules are the processing times ($\mathcal{M} = P$), and a job is eligible for a module, if its processing time matches; and the configurations are all the configurations bounded in size by T. Another option is to chose the processing times as basic objects, keeping all the other definitions essentially like before. Lastly, we could consider the whole set of jobs or the whole set of processing times as a single basic object with $D = |P|$ different values. In this case, we can define the set of modules as the set of configurations of processing times bounded by T.

9.4 EPTAS results

In this section, we present approximation schemes for each of the three considered problems. Each of the results follows the same approach: The instance is carefully simplified, a schedule for the simplified instance is found using the MCIP, and this schedule is transformed into a schedule for the original instance. The presentation of the result is also similar for each problem: We first discuss how the instance can be sensibly simplified, and how a schedule for the simplified instance can be transformed into a schedule for the original one. Next, we discuss how a schedule for the simplified instance can be found using the MCIP, and lastly, we summarize and analyze the taken steps.

For the sake of clarity, we have given rather formal definitions for the problems at hand in Section 9.2. In the following, however, we will use the terms in a more intuitive fashion for the most part, and we will, for

instance, often take a geometric rather than a temporal view on schedules and talk about the *length* or the *space* taken up by jobs and setups on machines rather than time. In particular, given a schedule for an instance of any one of the three problems together with an upper bound for the makespan T, the *free space* with respect to T on a machine is defined as the summed up lengths of time intervals between 0 and T in which the machine is idle. The free space (with respect to T) is the summed up free space of all the machines. For bounds T and L for the makespan and the free space, we say that a schedule is a (T, L)-schedule if its makespan is at most T and the free space with respect to T is at least L.

When transforming the instance we will increase or decrease processing and setup times and fill in or remove extra jobs. Consider a (T', L')-schedule, where T' and L' denote some arbitrary makespan or free space bounds. If we fill in extra jobs or increase processing or setup times, but can bound the increase on each machine by some bound b, we end up with a $(T' + b, L')$-schedule for the transformed instance. In particular we have the same bound for the free space, because we properly increased the makespan bound. If, on the other hand, jobs are removed or setup times decreased, we obviously still have a (T', L')-schedule for the transformed instance. This will be used frequently in the following.

9.4.1 Setup Class Model

We start with the setup class model. In this case, we can essentially reuse the simplification steps that were developed by Jansen and Land [55] for their PTAS. The main difference between the two procedures is that we solve the simplified instance via the MCIP, while they used a dynamic program. For the sake of self-containment, we include our own simplification steps, but remark that they are strongly inspired by those from [55]. In Section 9.5 we give a more elaborate rounding procedure resulting in an improved running time.

Simplification of the Instance. In the following, we distinguish *big setup* jobs j jobs belonging to classes k with setup times $s(k) \geqslant \varepsilon^3 T$ and *small setup* jobs with $s(k) < \varepsilon^3 T$. We denote the corresponding subsets of jobs by $\mathcal{J}^{\text{bst}}$ and $\mathcal{J}^{\text{sst}}$ respectively. Furthermore, we call a job *tiny* or *small*, if

Table 9.1. Overview on the job classifications

$s(j)$ \ $p(j)$	$< \varepsilon^4 T$	$< \varepsilon T$	$\geqslant \varepsilon T$
$< \varepsilon^3 T$	$\mathcal{J}^{\text{sst}}_{\text{tiny}}$	$\mathcal{J}^{\text{sst}}_{\text{small}}$	$\mathcal{J}^{\text{sst}}_{\text{large}}$
$\geqslant \varepsilon^3 T$	$\mathcal{J}^{\text{bst}}_{\text{tiny}}$	$\mathcal{J}^{\text{bst}}_{\text{small}}$	$\mathcal{J}^{\text{bst}}_{\text{large}}$

its processing time is smaller than $\varepsilon^4 T$ or εT respectively, and *big* or *large* otherwise. For any given set of jobs J, we denote the subset of tiny jobs from J with J_{tiny} and the small, big and large jobs analogously, see Table 9.1 for an overview. We simplify the instance in four steps, aiming for an instance that exclusively includes big jobs with big setup times and additionally only a constant number of distinct processing and setup times. For technical reason we assume $\varepsilon \leqslant 1/2$.

We proceed with the first simplification step. Let I_1 be the instance given by the job set $\mathcal{J} \setminus \mathcal{J}^{\text{sst}}_{\text{small}}$ and Q the set of setup classes completely contained in $\mathcal{J}^{\text{sst}}_{\text{small}}$, i.e., $Q = \{k \mid \forall j \in \mathcal{J} : k_j = k \Rightarrow j \in \mathcal{J}^{\text{sst}}_{\text{small}}\}$. An obvious lower bound on the space taken up by the jobs from $\mathcal{J}^{\text{sst}}_{\text{small}}$ in any schedule is given by $L = \sum_{j \in \mathcal{J}^{\text{sst}}_{\text{small}}} p(j) + \sum_{k \in Q} s(k)$. Note that the instance I_1 may include a reduced number K' of setup classes.

Lemma 9.5. *A schedule for I with makespan T induces a (T, L)-schedule for I_1, that is, a schedule with makespan T and free space at least L; and any (T', L)-schedule for I_1 can be transformed into a schedule for I with makespan at most $(1+\varepsilon)T' + 2\varepsilon^3 T$.*

Proof. The first claim is obvious and we therefore assume that we have a (T', L)-schedule for I_1. We group the jobs from $\mathcal{J}^{\text{sst}}_{\text{small}}$ by setup classes and first consider the groups with summed up processing time at most $\varepsilon^2 T$. For each of these groups, we check whether the respective setup class contains a large job. If this is the case, we schedule the complete group on an arbitrary machine on which such a large job is scheduled already and reduce the free space on the machine accordingly. Since the large jobs have a length of at least εT, there are at most $T'/(\varepsilon T)$ many large jobs on each machine and therefore the schedule on the respective machine has length at most $(1+\varepsilon)T'$. Note that it is possible that on these machines

there might be free space left with respect to T', which can be used to schedule the residual jobs in $\mathcal{J}^{\text{sst}}_{\text{small}}$. If, on the other hand, the respective class does not contain a large job and is therefore fully contained in $\mathcal{J}^{\text{sst}}_{\text{small}}$, we create a container including the whole class and its setup time. Note that the overall length of the container is at most $(\varepsilon^2 + \varepsilon^3)T \leqslant \varepsilon T$ (using $\varepsilon \leqslant 1/2$).

To schedule the groups with summed up processing time larger than $\varepsilon^2 T$ and the containers created in the last step, we create a sequence containing the containers and the remaining jobs ordered by setup class. We insert the items from this sequence greedily into the remaining free space in a next-fit fashion, exceeding T' on each machine by at most one item from the sequence. This can be done because we had a free space of at least L and the inserted objects had an overall length of at most L. To make the resulting schedule feasible, we have to insert some setup times. However, because the overall length of the jobs from each class in need of a setup is at least $\varepsilon^2 T$ and the sequence was ordered by classes, there are at most $T'/(\varepsilon^2 T) + 2$ distinct classes without a setup time on each machine. Inserting the missing setup times will therefore increase the makespan by at most $(T'/(\varepsilon^2 T) + 2)\varepsilon^3 T = \varepsilon T' + 2\varepsilon^3 T$. □

After this alteration, all the classes with a small setup time contain only large jobs. Next, we deal with these remaining jobs with small setup times $j \in \mathcal{J}^{\text{sst}}_{\text{large}}$. Let I_2 be the instance we get by increasing the setup times of the classes with small setup times to $\varepsilon^3 T$. We denote the setup time of class $k \in [K']$ for I_2 by s'_k. Note that there are no small setup jobs in I_2.

Lemma 9.6. *A (T', L')-schedule for I_1 induces a $((1+\varepsilon^2)T', L')$-schedule for I_2, and a (T', L')-schedule for I_2 is also a (T', L')-schedule for I_1.*

Proof. The first claim is true because in a schedule with makespan at most T there can be at most $T'/(\varepsilon T)$ many large jobs on any machine, and the second claim is obvious. □

Let I_3 be the instance we get by replacing the jobs from $\mathcal{J}^{\text{bst}}_{\text{tiny}}$ with placeholders of size $\varepsilon^4 T$. More precisely, for each class $k \in [K]$ we introduce $\lceil (\sum_{j \in \mathcal{J}^{\text{bst}}_{\text{tiny}}, k_j = k} p(j))/(\varepsilon^4 T) \rceil$ many jobs with processing time $\varepsilon^4 T$ and class

k. We denote the job set of I_3 by $\mathcal{J}'$ and the processing time of a job $j \in \mathcal{J}'$ by p'_j. Note that I_3 exclusively contains big jobs with big setup times.

Lemma 9.7. *If there is a (T', L')-schedule for I_2, there is also a $((1+\varepsilon)T', L')$-schedule; and if there is a (T', L')-schedule for I_3, there is also a $((1+\varepsilon)T', L')$-schedule for I_2.*

Proof. Note that for any (T', L')-schedule for I_2 or I_3 there are at most $T'/(\varepsilon^3 T)$ many distinct big setup classes scheduled on any machine. As a consequence, when considering such a schedule for I_2, we can remove the tiny jobs belonging to $\mathcal{J}^{\text{bst}}_{\text{tiny}}$ from the machines and instead fill in the placeholders such that each machine for each class receives at most as much length from that class as was removed, rounded up to the next multiple of $\varepsilon^4 T$. All placeholders can be placed like this and the makespan is increased by at most $(T'/(\varepsilon^3 T))\varepsilon^4 T = \varepsilon T'$. If, on the other hand, we consider such a schedule for I_3, we can remove the placeholders and instead fill in the respective tiny jobs, again overfilling by at most one job. This yields a $((1+\varepsilon)T', L')$-schedule for I_2 with the same argument. □

Lastly, we perform both a geometric and an arithmetic rounding step for the processing and setup times. The geometric rounding is needed to suitably bound the number of distinct processing and setup times and due to the arithmetic rounding we will be able to guarantee integral coefficients in the IP. More precisely, we set $\tilde{p}_j = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} p'_j/(\varepsilon^4 T)\rceil}\varepsilon^4 T$ and $\bar{p}_j = \lceil \tilde{p}_j/\varepsilon^5 T\rceil \varepsilon^5 T$ for each $j \in \mathcal{J}'$, as well as $\tilde{s}_j = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} s'_j/(\varepsilon^3 T)\rceil}\varepsilon^3 T$ and $\bar{s}_k = \lceil \tilde{s}_j/\varepsilon^5 T\rceil \varepsilon^5 T$ for each setup class $k \in [K']$. The resulting instance is called I_4.

Lemma 9.8. *A (T', L')-schedule for I_3 induces a $((1+3\varepsilon)T', L')$-schedule for I_4, and any (T', L')-schedule for I_4 can be turned into a (T', L')-schedule for I_3.*

Proof. For the first claim, we first stretch a given schedule by $(1+\varepsilon)$. This enables us to use the processing and setup times due to the geometric rounding step. Now, using the ones due to the second step increases the schedule by at most $2\varepsilon T'$, because there where at most $T'/(\varepsilon^4 T)$ many big jobs on any machine to begin with. The second claim is obvious. □

Based on the rounding steps, we define two makespan bounds $\bar{T}$ and $\breve{T}$: Let $\bar{T}$ be the makespan bound that is obtained from T by the application of the Lemmata 9.5-9.8 in sequence, i.e., $\bar{T} = (1+\varepsilon^2)(1+\varepsilon)(1+3\varepsilon)T = (1+\mathcal{O}(\varepsilon))T$. We will find a $(\bar{T}, L)$-schedule for I_4 utilizing the MCIP and afterward apply the Lemmata 9.5-9.8 backwards, to get a schedule with makespan $\breve{T} = (1+\varepsilon)^2\bar{T} + \varepsilon^3 T = (1+\mathcal{O}(\varepsilon))T$.

Let P and S be the sets of distinct occurring processing and setup times for instance I_4. Because of the rounding, the minimum and maximum lengths of the setup and processing times, and $\varepsilon < 1$, we can bound $|P|$ and $|S|$ by $\mathcal{O}(\log_{1+\varepsilon} 1/\varepsilon) = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$.

Utilization of the MCIP. At this point, we can employ the module configuration IP. The basic objects in this context are the setup classes, i.e., $\mathcal{B} = [K']$, and the different values are the numbers of jobs with a certain processing time, i.e., $D = |P|$. We set $n_{k,p}$ to be the number of jobs from setup class $k \in [K']$ with processing time $p \in P$. The modules correspond to batches of jobs together with a setup time. Batches of jobs can be modeled as configurations of processing times, that is, multiplicity vectors indexed by the processing times. Hence, we define the set of modules $\mathcal{M}$ to be the set of pairs of configurations of processing times and setup times with a summed up size bounded by $\bar{T}$, i.e., $\mathcal{M} = \{(C,s) \mid C \in \mathcal{C}_P(\bar{T}), s \in S, s + \Lambda(C) \leqslant \bar{T}\}$, and write $M_p = C_p$ and $s(M) = s$ for each module $M = (C,s) \in \mathcal{M}$. The values of a module M are given by the numbers M_p and its size $\Lambda(M)$ by $s(M) + \sum_{p \in P} M_p p$. Remember that the configurations $\mathcal{C}$ are the configurations of module sizes H that are bounded in size by $\bar{T}$, i.e., $\mathcal{C} = \mathcal{C}_H(\bar{T})$. A setup class is eligible for a module, if the setup times fit, i.e., $\mathcal{B}_M = \{k \in [K'] \mid s(k) = s(M)\}$. Lastly, we establish $\varepsilon^5 T = 1$ by scaling.

For the sake of readability, we state the resulting constraints of the MCIP with adapted notation and without duplication of the configuration variables:

$$\min \sum_{C \in \mathcal{C}} \Lambda(C) x_C \tag{9.6}$$

$$\sum_{C \in \mathcal{C}} x_C = m \tag{9.7}$$

$$\sum_{C \in \mathcal{C}} C_h x_C = \sum_{k \in [K']} \sum_{M \in \mathcal{M}(h)} y_M^{(k)} \qquad \forall h \in H \tag{9.8}$$

$$\sum_{M \in \mathcal{M}} M_p y_M^{(k)} = n_{k,p} \qquad \forall k \in [K'], p \in P \tag{9.9}$$

Note that the coefficients are all integral and this includes those of the objective function, i.e., $\sum_C \Lambda(C) x_C$, because of the scaling step.

Lemma 9.9. *With the above definitions, there is a $(\bar{T}, L)$-schedule for I_4, iff the* MCIP *has a solution with objective value at most $m\bar{T} - L$.*

Proof. Let there be a $(\bar{T}, L)$-schedule for I_4. Then the schedule on a given machine corresponds to a distinct configuration C that can be determined by counting for each possible group size a the batches of jobs from the same class whose length together with the setup time adds up to an overall length of a. Note that the length of this configuration is equal to the used up space on that machine. We fix an arbitrary setup class k and set the variables $x_C^{(k)}$ accordingly (and $x_C^{(k')} = 0$ for $k' \neq k$ and $C \in \mathcal{C}$). By this setting we get an objective value of at most $m\bar{T} - L$ because there was L free space in the schedule. For each class k and module M, we count the number of machines on which the there are exactly M_p jobs with processing time p from class k for each $p \in P$, and set $y_M^{(k)}$ accordingly. It is easy to see that the constraints are satisfied by these definitions.

Given a solution (x, y) of the MCIP, we define a corresponding schedule: Because of (9.7) we can match the machines to configurations such that each machine is matched to exactly one configuration. If machine i is matched to C, for each group G we create C_G slots of length $\Lambda(G)$ on i. Next, we divide the setup classes into batches. For each class k and module M, we create $y_M^{(k)}$ batches of jobs from class k with M_p jobs with processing time p for each $p \in P$ and place the batch together with the corresponding setup time into a fitting slot on some machine. Because of (9.9) and (9.8) all jobs can be placed by this process. Note that the used space equals the overall size of the configurations and therefore we have free space of at least L. □

Result. Using the above results, we can formulate and analyze the following procedure:

Algorithm 1.

1. *Generate the modified instance I_4:*
   - ▷ *Remove the small jobs with small setup times.*
   - ▷ *Increase the setup times of the remaining classes with small setup times.*
   - ▷ *Replace the tiny jobs with big setup times.*
   - ▷ *Round up the resulting processing and setup times.*
2. *Build and solve the* MCIP *for I_4.*
3. *If the* MCIP *is infeasible or the objective value greater than $m\bar{T} - L$, report that I has no solution with makespan T.*
4. *Otherwise build the schedule with makespan $\bar{T}$ and free space at least L for I_4.*
5. *Transform the schedule into a schedule for I with makespan at most $\check{T}$:*
   - ▷ *Use the prerounding processing and setup times.*
   - ▷ *Replace the placeholders by the tiny jobs with big setup times.*
   - ▷ *Use the orignal setup times of the classes with small setup times.*
   - ▷ *Insert the small jobs with small setup times into the free space.*

The procedure is correct due to the above results. To analyze its running time, we first bound the parameters of the MCIP. We have $|\mathcal{B}| = K' \leqslant K$ and $D = |P|$ by definition, and $|\mathcal{M}| = \mathcal{O}(|S|(1/\varepsilon^3)^{|P|}) = 2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$, because $|S|, |P| \in \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$. This is true, due to the last rounding step, which also implies $|H| \in \mathcal{O}(1/\varepsilon^5)$, yielding $|\mathcal{C}| = |H|^{\mathcal{O}(1/\varepsilon^3)} = 2^{\mathcal{O}(1/\varepsilon^3 \log 1/\varepsilon)}$. According to Observation 9.4, this yields a brick size of $t = 2^{\mathcal{O}(1/\varepsilon^3 \log 1/\varepsilon)}$, a brick number of K, $r \in \mathcal{O}(1/\varepsilon^5)$ globally, and $s \in \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$ locally uniform constraints for the MCIP. We have $\Delta = \mathcal{O}(1/\varepsilon^5)$, because all occurring values in the processing time matrix are bounded in $\bar{T}$, and we have $\bar{T} = \mathcal{O}(1/\varepsilon^5)$, due to the scaling. Furthermore, the values of the objective function, the right hand side, and the upper and lower bounds on the variables are bounded by $\mathcal{O}(n/\varepsilon^5)$, yielding a bound of $\mathcal{O}(\log n/\varepsilon^5)$ for the encoding length of the biggest number in the input φ. Lastly, all

variables can be bounded by 0 from below and $\mathcal{O}(m/\varepsilon^3)$ from above, yielding $\Phi = \mathcal{O}(m/\varepsilon^3)$.

By Theorem 9.3 and some arithmetic, the MCIP can be solved in time

$$\begin{aligned}&(rs\Delta)^{\mathcal{O}(r^2s+rs^2)}t^2n^2\varphi\log(\Phi)\log(nt\Phi)\\&= 2^{\mathcal{O}(1/\varepsilon^{11}\log^2 1/\varepsilon)}K^2\log n\log m\log Km.\end{aligned}$$

When building the actual schedule, we iterate through the jobs and machines like indicated in the proof of Lemma 9.9, yielding the following:

Theorem 9.10. *The algorithm for the setup class model finds a schedule with makespan $(1+\mathcal{O}(\varepsilon))T$ or correctly determines that there is no schedule with makespan T in time $2^{\mathcal{O}(1/\varepsilon^{11}\log^2 1/\varepsilon)}K^2nm\log Km$.*

9.4.2 Splittable Model

The approximation scheme for the splittable model presented in this section is probably the easiest one discussed in this work. There is, however, one problem concerning this procedure: Its running time is polynomial in the number of machines, which might be exponential in the input size. In Section 9.5, we show how this problem can be overcome and further improve the running time.

Simplification of the Instance. In this context the set of big setup jobs $\mathcal{J}^{\text{bst}}$ is given by the jobs with setup times at least εT and the small setup jobs $\mathcal{J}^{\text{sst}}$ are all the others. Let $L = \sum_{j\in\mathcal{J}^{\text{sst}}}(s(j)+p(j))$. Because every job has to be scheduled and every setup has to be paid at least once, L is a lower bound on the summed up space due to small jobs in any schedule. Let I_1 be the instance that we get by removing all the small setup jobs from the given instance I.

Lemma 9.11. *A schedule with makespan T for I induces a (T,L)-schedule for I_1; and any (T',L)-schedule for I_1 can be transformed into a schedule for I with makespan at most $T'+\varepsilon T$.*

Proof. The first claim is obvious. Hence, consider a sequence consisting of the jobs from $\mathcal{J}^{\text{sst}}$ together with their set up times, where the setup

up time of a job is the direct predecessor of the job. We insert the setup times and jobs from this sequence greedily into the schedule in a next-fit fashion: Given a machine, we keep inserting the items from the sequence on the machine at the end of the schedule until the taken up space on the machine reaches T'. If the current item does not fit exactly, we cut it such that the used space on the machine is exactly T'. Then we continue with the next machine. We can place the whole sequence like this without exceeding the makespan T', because we have free space of at least L which is the summed up length of the items in the sequence. Next, we remove each setup time that was placed only partly on a machine together witch those that were placed at the end of the schedule, and insert a fitting setup time for the jobs that were scheduled without one, which can happen only once for each machine. This yields a feasible schedule, whose makespan is increased by at most εT. □

Next, we round up the processing and setup times of I_1 to the next multiple of $\varepsilon^2 T$, that is, for each job $j \in \mathcal{J}$ we set $\bar{p}_j = \lceil p(j)/(\varepsilon^2 T) \rceil \varepsilon^2 T$ and $\bar{s}_j = \lceil s(j)/(\varepsilon^2 T) \rceil \varepsilon^2 T$. We call the resulting instance I_2, and denote its job set by $\mathcal{J}'$.

Lemma 9.12. *If there is a (T, L')-schedule for I_1, there is also a $((1+2\varepsilon)T, L')$-schedule for I_2 in which the length of each job part is a multiple of $\varepsilon^2 T$, and any (T', L')-schedule for I_2 yields a (T', L')-schedule for I_1.*

Proof. Consider a (T, L)-schedule for I_1. There are at most $1/\varepsilon$ jobs scheduled on each machine since each setup time has a length of at least εT. On each machine, we extend each occurring setup time and the processing time of each occurring job part by at most $\varepsilon^2 T$ to round it to a multiple of $\varepsilon^2 T$. This step extends the makespan by at most $2\varepsilon T$. Since now each job part is a multiple of $\varepsilon^2 T$, the total processing time of the job is a multiple of $\varepsilon^2 T$ too.

In the last step, we check for each job $j \in \mathcal{J}^{\text{bst}}$ if the total processing time is now larger than the smallest multiple of $\varepsilon^2 T$, which is larger than its original processing time. If this is the case, we discard the spare processing time. Lastly, there is at least as much free space in the resulting schedule as in the original one, because we properly increased the makespan bound. The second claim is obvious. □

Based on the two Lemmata, we define two makespan bounds $\bar{T} = (1+2\varepsilon)T$ and $\breve{T} = \bar{T} + \varepsilon T = (1+3\varepsilon)T$. We will use the MCIP to find a $(\bar{T}, L)$-schedule for I_2 in which the length of each job part is a multiple of $\varepsilon^2 T$. Using the two Lemmata, this will yield a schedule with makespan at most $\breve{T}$ for the original instance I.

Utilization of the MCIP. The basic objects in this context are the (big setup) jobs, i.e., $\mathcal{B} = \mathcal{J}^{\text{bst}} = \mathcal{J}'$, and they have only one value ($D = 1$), namely, their processing time. Moreover, the modules are defined as the set of pairs of job piece sizes and setup times, i.e., $\mathcal{M} = \left\{(q,s) \,\middle|\, s, q \in \{x\varepsilon^2 T \mid x \in \mathbb{Z}, 0 < x \leqslant 1/\varepsilon^2\}, s \geqslant \varepsilon T\right\}$, and we write $s(M) = s$ and $q_M = q$ for each module $M = (q,s) \in \mathcal{M}$. Corresponding to the value of the basic objects the value of a module M is q_M, and its size $\Lambda(M)$ is given by $q_M + s(M)$. A job is eligible for a module, if the setup times fit, i.e., $\mathcal{B}_M = \{j \in \mathcal{J}' \mid s(j) = s(M)\}$. In order to ensure integral values, we establish $\varepsilon^2 T = 1$ via a simple scaling step. The set of configurations $\mathcal{C}$ is comprised of all configurations of module sizes H that are bounded in size by $\bar{T}$, i.e., $\mathcal{C} = \mathcal{C}_{\mathcal{M}}(\bar{T})$. We state the constraints of the MCIP for the above definitions with adapted notation and without duplication of the configuration variables:

$$\sum_{C \in \mathcal{C}} x_C = m \tag{9.10}$$

$$\sum_{C \in \mathcal{C}} C_h x_C = \sum_{j \in \mathcal{J}'} \sum_{M \in \mathcal{M}(h)} y_M^{(j)} \qquad \forall h \in H \tag{9.11}$$

$$\sum_{M \in \mathcal{M}} q_M y_M^{(j)} = p(j) \qquad \forall j \in \mathcal{J}' \tag{9.12}$$

Note that we additionally minimize the summed up size of the configurations, via the objective function $\sum_C \Lambda(C) x_C$.

Lemma 9.13. *With the above definitions, there is a $(\bar{T}, L)$-schedule for I_2 in which the length of each job piece is a multiple of $\varepsilon^2 T$, iff* MCIP *has a solution with objective value at most $m\bar{T} - L$.*

Proof. Given such a schedule for I_2, the schedule on each machine corresponds to exactly one configuration G that can be derived by counting the job pieces and setup times with the same summed up length a and

setting C_G accordingly, where G is the group of modules with length a. The size of the configuration C is equal to the used space on the respective machine. Therefore, we can fix some arbitrary job j and set the variables $x_C^{(j)}$ to the number of machines whose schedule corresponds to C (and $x_C^{(j')} = 0$ for $j' \neq j$ and $C \in \mathcal{C}$). Since there is at least a free space of L for the schedule, the objective value is bounded by $m\bar{T} - L$. Furthermore, for each job j and job part length q, we count the number of times a piece of j with length q is scheduled and set $y^{(j)}_{(q,s(j))}$ accordingly. It is easy to see that the constraints are satisfied.

Now, let (x, y) be a solution to the MCIP with objective value at most $m\bar{T} - L$. We use the solution to construct a schedule: For job j and configuration C we reserve $x_C^{(j)}$ machines. On each of these machines we create C_h slots of length h, for each module size $h \in H$. Note that because of (9.10) there is the exact right number of machines for this. Next, consider each job j and possible job part length q and create $y^{(j)}_{(q,s(j))}$ split pieces of length q and place them together with a setup of $s(j)$ into a slot of length $s(j) + q$ on any machine. Because of (9.12) the entire job is split up by this, and because of (9.11) there are enough slots for all the job pieces. Note that the used space in the created schedule is equal to the objective value of (x, y) and therefore there is at least L free space. □

Result. Summing up, we can find a schedule of length at most $(1 + 3\varepsilon)T$ or correctly determine that there is no schedule of length T with the following procedure:

Algorithm 2.

1. *Generate the modified instance I_2:*
   - ▻ *Remove the small setup jobs.*
   - ▻ *Round the setup and processing times of the remaining jobs.*
2. *Build and solve the* MCIP *for this case.*
3. *If the IP is infeasible or the objective value greater than $m\bar{T} - L$, report that I has no solution with makespan T.*

4. *Otherwise build the schedule with makespan $\bar{T}$ and free space at least L for $\bar{I}$.*
5. *Transform the schedule into a schedule for I with makespan at most $\breve{T}$:*
   - ▷ *Use the original processing and setup times.*
   - ▷ *Greedily insert the small setup jobs.*

To assess the running time of the procedure, we mainly need to bound the parameters of the MCIP, namely $|\mathcal{B}|$, $|H|$, $|\mathcal{M}|$, $|\mathcal{C}|$ and D. By definition, we have $|\mathcal{B}| = |\mathcal{J}'| \leqslant n$ and $D = 1$. Since all setup times and job piece lengths are multiples of $\varepsilon^2 T$ and bounded by T, we have $|\mathcal{M}| = \mathcal{O}(1/\varepsilon^4)$ and $|H| = \mathcal{O}(1/\varepsilon^2)$. This yields $|\mathcal{C}| \leqslant |H|^{\mathcal{O}(1/\varepsilon+2)} = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$, because the size of each module is at least εT and the size of the configurations bounded by $(1+2\varepsilon)T$.

According to Observation 9.4, we now have brick-size $t = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$, brick number $|\mathcal{B}| = n$, $r = |\Gamma| + 1 = \mathcal{O}(1/\varepsilon^2)$ globally uniform and $s = D = 1$ locally uniform constraints. Because of the scaling step, all occurring numbers in the constraint matrix of the MCIP are bounded by $1/\varepsilon^2$ and therefore $\Delta \leqslant 1/\varepsilon^2$. Furthermore, each occurring number can be bounded by $\mathcal{O}(m/\varepsilon^2)$ and this is an upper bound for each variable as well, yielding $\varphi = \mathcal{O}(\log m/\varepsilon^2)$ and $\Phi = \mathcal{O}(m/\varepsilon^2)$. Hence the MCIP, can be solved in time

$$(rs\Delta)^{\mathcal{O}(r^2 s + rs^2)} t^2 n^2 \varphi \log(\Phi) \log(nt\Phi) = 2^{\mathcal{O}(1/\varepsilon^4 \log 1/\varepsilon)} n^2 \log^2 m \log nm.$$

While the first step of the procedure is obviously dominated by the above, this is not the case for the remaining ones. In particular, building the schedule from the IP solution costs $\mathcal{O}((n+m)/\varepsilon^2)$, if the procedure described in the proof of Lemma 9.13 is realized in a straight-forward fashion. The last step of the algorithm is dominated by this, yielding the running time stated in the theorem below. Note that the number of machines m could be exponential in the number of jobs, and therefore the described procedure is a PTAS only for the special case of $m = \text{poly}(n)$. However, this limitation can be overcome with a little extra effort, as we discuss in Section 9.5.

Theorem 9.14. *The algorithm for the splittable model finds a schedule with makespan at most $(1+3\varepsilon)T$ or correctly determines that there is no schedule with makespan T in time $2^{\mathcal{O}(1/\varepsilon^4 \log 1/\varepsilon)} n^2 m \log m \log nm$.*

9.4.3 Preemptive Model

In the preemptive model we have to actually consider the time-line of the schedule on each machine instead of just the assignment of the jobs or job pieces, and this causes some difficulties. For instance, we will have to argue that it suffices to look for a schedule with few possible starting points, and we will have to introduce additional constraints in the IP in order to ensure that pieces of the same job do not overlap. Our first step in dealing with this extra difficulty is to introduce some concepts and notation: For a given schedule with a makespan bound T, we call a job piece together with its setup a *block*, and we call the schedule X-layered, for some value X, if each block starts at a multiple of X. Corresponding to this, we call the time in the schedule between two directly succeeding multiples of X a *layer* and the corresponding time on a single machine a *slot*. We number the layers bottom to top and identify them with their number, that is, the set of layers Ξ is given by $\{\ell \in \mathbb{Z}_{>0} \mid (\ell-1)X \leqslant T\}$. Note that in an X-layered schedule, there is at most one block in each slot and for each layer there can be at most one block of each job present. Furthermore, for X-layered schedules, we slightly alter the definition of free space: We solely count the space from slots that are completely free. If in such a schedule, for each job there is at most one slot occupied by this job but not fully filled, we additionally call the schedule *layer-compliant*.

Simplification of the Instance

In the preemptive model we distinguish *big*, *medium*, and *small* setup jobs, using two parameters δ and μ: The big setup jobs $\mathcal{J}^{\text{bst}}$ are those with setup time at least δT, the small $\mathcal{J}^{\text{sst}}$ have a setup time smaller than μT, and the medium $\mathcal{J}^{\text{mst}}$ are the ones in between. We set $\mu = \varepsilon^2\delta$ and we choose $\delta \in \{\varepsilon^1, \dots, \varepsilon^{2/\varepsilon^2}\}$ such that the summed up processing time together with the summed up setup time of the medium setup jobs is upper bounded by $m\varepsilon T$, i.e., $\sum_{j\in\mathcal{J}^{\text{mst}}}(s(j)+p(j)) \leqslant m\varepsilon T$. If there is a schedule with makespan T, such a choice is possible because of the pidgeon hole principle, and because the setup time of each job has to occur at least once in any schedule. Similar arguments are widely used, e.g. in the context of geometrical packing algorithms. Furthermore, we distinguish the jobs by

Table 9.2. Overview on the job classifications

$p(j)$ \ $s(j)$	$< \mu T$	$\geqslant \mu T, < \delta T$	$\geqslant \delta T$
$< \varepsilon T$	$\mathcal{J}^{\text{sst}}_{\text{small}}$	$\mathcal{J}^{\text{mst}}_{\text{small}}$	$\mathcal{J}^{\text{bst}}_{\text{small}}$
$\geqslant \varepsilon T$	$\mathcal{J}^{\text{sst}}_{\text{big}}$	$\mathcal{J}^{\text{mst}}_{\text{big}}$	$\mathcal{J}^{\text{bst}}_{\text{big}}$

processing times, calling those with processing time at least εT *big* and the others *small*. For a given set of jobs J, we call the subsets of big or small jobs J_{big} or J_{small} respectively, see Table 9.2 for an overview. We perform three simplification steps, aiming for an instance in which the small and medium setup jobs are big; small setup jobs have setup time 0; and for which an $\varepsilon\delta T$-layered, layer-compliant schedule exists.

Let I_1 be the instance we get by removing the small jobs with medium setup times $\mathcal{J}^{\text{mst}}_{\text{small}}$ from the given instance I.

Lemma 9.15. *If there is a schedule with makespan at most T for I, there is also such a schedule for I_1, and if there is a schedule with makespan at most T' for I_1 there is a schedule with makespan at most $T' + (\varepsilon + \delta)T$ for I.*

Proof. The first claim is obvious. For the second, we create a sequence containing the jobs from $\mathcal{J}^{\text{mst}}_{\text{small}}$ each directly preceded by its setup time. Recall that the overall length of the objects in this sequence is at most $m\varepsilon T$, and the length of each job is bounded by εT. We greedily insert the objects from the sequence, considering each machine in turn. On the current machine we start at time $T' + \delta T$ and keep inserting until $T' + \delta T + \varepsilon T$ is reached. If the current object is a setup time, we discard it and continue with the next machine and object. If, on the other hand, it is a job, we split it such that the remaining space on the current machine can be perfectly filled. We can place all objects like this; however, the first job part placed on a machine might be missing a setup. We can insert the missing setups because they have length at most δT and between time T' and $T' + \delta T$ there is free space. □

Next, we consider the jobs with small setup times: Let I_2 be the instance we get by removing the small jobs with small setup times $\mathcal{J}^{\text{sst}}_{\text{small}}$ and setting the setup time of the big jobs with small setup times to zero, i.e., $\bar{s}_j = 0$

for each $j \in \mathcal{J}^{\text{sst}}_{\text{big}}$. Note that in the resulting instance each small job has a big setup time. Furthermore, let $L := \sum_{j \in \mathcal{J}^{\text{sst}}_{\text{small}}} p(j) + s(j)$. Then L is an obvious lower bound for the space taken up by the jobs from $\mathcal{J}^{\text{sst}}_{\text{small}}$ in any schedule.

Lemma 9.16. *If there is a schedule with makespan at most T for I_1, there is also a (T, L)-schedule for I_2; and if there is a γT-layered (T', L)-schedule for I_2, with T' a multiple of γT, there is also a schedule with makespan at most $(1 + \gamma^{-1}\mu)T' + (\mu + \varepsilon)T$ for I_1.*

Proof. The first claim is obvious, and for the second consider a γT-layered (T', L)-schedule for I_2. We create a sequence that contains the jobs of $\mathcal{J}^{\text{sst}}_{\text{small}}$ and their setups such that each job is directly preceded by its setup. Remember that the remaining space in partly filled slots is not counted as free space. Hence, since the overall length of the objects in the sequence is L, there is is enough space in the free slots of the schedule to place them. We do so in a greedy fashion guaranteeing that each job is placed on exactly one machine: We insert the objects from the sequence into the free slots, considering each machine in turn and starting on the current machine from the beginning of the schedule and moving on towards its end. If an object cannot be fully placed into the current slot there are two cases: It could be a job or a setup. In the former case, we cut it and continue placing it in the next slot, or, if the current slot was the last one, we place the rest at the end of the schedule. In the latter case, we discard the setup and continue with the next slot and object. The resulting schedule is increased by at most εT, which is caused by the last job placed on a machine.

To get a proper schedule for I_1, we have to insert some setup times: For the large jobs with small setup times and for the jobs that were cut in the greedy procedure. We do so by inserting a time window of length μT at each multiple of γT and at the end of the original schedule on each machine. By this, the schedule is increased by at most $\gamma^{-1}\mu T' + \mu T$. Since all the job parts in need of a setup are small and did start at multiples of μT or at the end, we can insert the missing setups. Note that blocks that span over multiple layers are cut by the inserted time windows. This, however, can easily be repaired by moving the cut pieces properly down. □

We continue by rounding the medium and big setup and all the processing times. In particular, we round the processing times and the big setup times up to the next multiple of $\varepsilon\delta T$ and the medium setup times to the next multiple of $\varepsilon\mu T$, i.e., $\bar{p}_j = \lceil p(j)/(\varepsilon\delta T)\rceil \varepsilon\delta T$ for each job j, $\bar{s}_j = \lceil s(j)/(\varepsilon\delta T)\rceil \varepsilon\delta T$ for each big setup job $j \in \mathcal{J}^{\text{bst}}$, and $\bar{s}_j = \lceil s(j)/(\varepsilon\mu T)\rceil \varepsilon\mu T$ for each medium setup job $j \in \mathcal{J}^{\text{mst}}_{\text{big}}$.

Lemma 9.17. *If there is a (T, L)-schedule for I_2, there is also an $\varepsilon\delta T$-layered, layer-compliant $((1+3\varepsilon)T, L)$-schedule for I_3. On the other hand, if there is a γT-layered (T', L)-schedule for I_3, there is also such a schedule for I_2.*

While the second claim is easy to see, the proof of the first is rather elaborate and unfortunately a bit tedious. Hence, since we believe Lemma 9.17 to be fairly plausible by itself, we postpone its proof to the end of the section and proceed discussing its use.

For the big and small setup jobs both processing and setup times are multiples of $\varepsilon\delta T$. Therefore, the length of each of their blocks in an $\varepsilon\delta T$-layered, layer-compliant schedule is a multiple of $\varepsilon\delta T$. For a medium setup job, on the other hand, we know that the overall length of its blocks has the form $x\varepsilon\delta T + y\varepsilon\mu T$, with non-negative integers x and y. In particular it is a multiple of $\varepsilon\mu T$, because $\varepsilon\delta T = (1/\varepsilon^2)\varepsilon\mu T$. In a $\varepsilon\delta T$-layered, layer-compliant schedule, for each medium setup job the length of all but at most one block is a multiple of $\varepsilon\delta T$ and therefore a multiple of $\varepsilon\mu T$. If both the overall length and the lengths of all but one block are multiples of $\varepsilon\mu T$, this is also true for the one remaining block. Hence, we will use the MCIP not to find an $\varepsilon\delta T$-layered, layer-compliant schedule in particular, but an $\varepsilon\delta T$-layered one with block sizes as described above and maximum free space.

Based on the simplification steps, we define two makespan bounds $\bar{T}$ and $\breve{T}$: Let $\bar{T}$ be the makespan bound we get by the application of the Lemmata 9.15-9.17, i.e., $\bar{T} = (1+3\varepsilon)T$. We will use the MCIP to find an $\varepsilon\delta T$-layered $(\bar{T}, L)$-schedule for I_3, and apply the Lemmata 9.15-9.17 backwards to get schedule for I with makespan at most $\breve{T} = (1+(\varepsilon\delta)^{-1}\mu)\bar{T} + (\mu+\varepsilon)T + (\varepsilon+\delta)T \leqslant (1+9\varepsilon)T$, using $\varepsilon \leqslant 1/2$.

Utilization of the MCIP

Similar to the splittable case, the basic objects are the (big) jobs, i.e., $\mathcal{B} = \mathcal{J}_{\text{big}}$, and their single value is their processing time ($D = 1$). The modules, on the other hand, are more complicated, because they additionally need to encode which layers are exactly used and, in case of the medium jobs, to which degree the last layer is filled. For the latter we introduce buffers, representing the unused space in the last layer, and define modules as tuples (ℓ, q, s, b) of starting layer, job piece size, setup time and buffer size. For a module $M = (\ell, q, s, b)$, we write $\ell_M = \ell$, $q_M = q$, $s(M) = s$ and $b_M = b$, and we define the size $\Lambda(M)$ of M as $s + q + b$. The overall set of modules $\mathcal{M}$ is the union of the modules for big, medium, and small setup jobs $\mathcal{M}^{\text{bst}}$, $\mathcal{M}^{\text{mst}}$ and $\mathcal{M}^{\text{sst}}$ that are defined in the following. For this let $Q^{\text{bst}} = \{q \mid q = x\varepsilon\delta T, x \in \mathbb{Z}_{>0}, q \leqslant \bar{T}\}$ and $Q^{\text{mst}} = \{q \mid q = x\varepsilon\mu T, x \in \mathbb{Z}_{>0}, q \leqslant \bar{T}\}$ be the sets of possible job piece sizes of big and medium setup jobs; $S^{\text{bst}} = \{s \mid s = x\varepsilon\delta T, x \in \mathbb{Z}_{\geqslant 1/\varepsilon}, s \leqslant \bar{T}\}$ and $S^{\text{mst}} = \{s \mid s = x\varepsilon\mu T, x \in \mathbb{Z}_{\geqslant 1/\varepsilon}, s \leqslant \delta T\}$ be the sets of possible big and medium setup times; $B = \{b \mid b = x\varepsilon\mu T, x \in \mathbb{Z}_{\geqslant 0}, b < \varepsilon\delta T\}$ the set of possible buffer sizes; and $\Xi = \{1, \ldots, 1/(\varepsilon\delta) + 3/\delta\}$ the set of layers. We set:

$$\begin{aligned}
\mathcal{M}^{\text{bst}} =& \{(\ell, q, s, 0) \mid \ell \in \Xi, q \in Q^{\text{bst}}, s \in S^{\text{bst}}, (\ell - 1)\varepsilon\delta T + s + q \leqslant \bar{T}\} \\
\mathcal{M}^{\text{mst}} =& \{(\ell, q, s, b) \in \Xi \times Q^{\text{mst}} \times S^{\text{mst}} \times B \mid \\
& \qquad x = s + q + b \in \varepsilon\delta T\mathbb{Z}_{>0}, (\ell - 1)\varepsilon\delta T + x \leqslant \bar{T}\} \\
\mathcal{M}^{\text{sst}} =& \{(\ell, \varepsilon\delta T, 0, 0) \mid \ell \in \Xi\}
\end{aligned}$$

Concerning the small setup modules, note that the small setup jobs have a setup time of 0 and therefore may be covered slot by slot. We establish $\varepsilon\mu T = 1$ via scaling, to ensure integral values. A big, medium or small job is eligible for a module, if it is also big, medium or small respectively and the setup times fit.

We have to avoid that two modules M_1, M_2, whose corresponding time intervals overlap, are used to cover the same job or in the same configuration. Such an overlap is given, if there is some layer ℓ used by both of them, that is, $(\ell_M - 1)\varepsilon\delta T \leqslant (\ell - 1)\varepsilon\delta T < (\ell_M - 1)\varepsilon\delta T + \Lambda(M)$ for both $M \in \{M_1, M_2\}$. Hence, for each layer $\ell \in \Xi$, we set $\mathcal{M}_\ell \subseteq \mathcal{M}$ to be the set of modules that use layer ℓ. Furthermore, we partition the modules

into groups Γ by size and starting layer, i.e., $\Gamma = \{G \subseteq \mathcal{M} \mid M, M' \in G \Rightarrow \Lambda(M) = \Lambda(M') \wedge \ell_M = \ell_{M'}\}$. The size of a group $G \in \Gamma$ is the size of a module from G, i.e. $\Lambda(G) = \Lambda(M)$ for $M \in G$. Unlike before, we consider *configurations of module groups* rather than module sizes. More precisely, the set of configurations $\mathcal{C}$ is given by the configurations of groups such that for each layer at most one group using this layer is chosen, i.e., $\mathcal{C} = \{C \in \mathbb{Z}^{\Gamma}_{\geqslant 0} \mid \forall \ell \in \Xi : \sum_{G \subseteq \mathcal{M}_\ell} C_G \leqslant 1\}$. With this definition we prevent overlap conflicts on the machines. Note that unlike in the cases considered so far, the size of a configuration does not correspond to a makespan in the schedule, but to used space, and the makespan bound is realized in the definition of the modules instead of in the definition of the configurations. To also avoid conflicts for the jobs, we extend the basic MCIP with additional locally uniform constraints. In particular, the constraints of the extended MCIP for the above definitions with adapted notation and without duplication of the configuration variables are given by:

$$\sum_{C \in \mathcal{C}} x_C = m \tag{9.13}$$

$$\sum_{C \in \mathcal{C}(T)} C_G x_C = \sum_{j \in \mathcal{J}} \sum_{M \in G} y_M^{(j)} \qquad \forall G \in \Gamma \tag{9.14}$$

$$\sum_{M \in \mathcal{M}} q_M y_M^{(j)} = p(j) \qquad \forall j \in \mathcal{J} \tag{9.15}$$

$$\sum_{M \in \mathcal{M}_\ell} y_M^{(j)} \leqslant 1 \qquad \forall j \in \mathcal{J}, \ell \in \Xi \tag{9.16}$$

Like in the first two cases we minimize the summed up size of the configurations, via the objective function $\sum_C \Lambda(C) x_C$. Note that in this case the size of a configuration does not have to equal its height. It is easy to see that the last constraint is indeed locally uniform. However, since we have an inequality instead of an equality, we have to introduce $|\Xi|$ slack variables in each brick, yielding:

Observation 9.18. The MCIP extended like above is an n-fold IP with brick-size $t = |\mathcal{M}| + |\mathcal{C}| + |\Xi|$, brick number $n = |\mathcal{J}|$, $r = |\Gamma| + 1$ globally uniform and $s = D + |\Xi|$ locally uniform constraints.

Lemma 9.19. *With the above definitions, there is an $\varepsilon\delta T$-layered $(\bar{T}, L)$-schedule*

for I_3 in which the length of a block is a multiple of $\varepsilon\delta T$, if it belongs to a small or big setup job, or a multiple of $\varepsilon\mu T$ otherwise, iff the extended MCIP *has a solution with objective value at most $m\bar{T} - L$.*

Proof. We first consider such a schedule for I_3. For each machine, we can derive a configuration that is given by the starting layers of the blocks together with the summed up length of the slots the respective block is scheduled in. The size of the configuration C is equal to the used space on the respective machine. Hence, we can fix some arbitrary job j and set $x_C^{(j)}$ to the number of machines corresponding to j (and $x_C^{(j')} = 0$ for $j' \neq j$). Keeping in mind that in an $\varepsilon\delta T$-layered schedule the free space is given by the free slots, the above definition yields an objective value bounded by $m\bar{T} - L$, because there was free space of at least L. Next, we consider the module variables for each job j in turn: If j is a small setup job, we set $y^{(j)}_{(\ell,\varepsilon\delta T,0,0)}$ to 1, if the j occurs in ℓ, and to 0 otherwise. Now, let j be a big setup job. For each of its blocks, we set $y^{(j)}_{(\ell,z-s(j),s(j),0)} = 1$, where ℓ is the starting layer and z the length of the block. The remaining variables are set to 0. Lastly, let j be a medium setup job. For each of its blocks, we set $y^{(j)}_{(\ell,z-s(j),s(j),b)} = 1$, where ℓ is the starting layer of the block, z its length and $b = \lceil z/(\varepsilon\delta T)\rceil\varepsilon\delta T - z$. Again, the remaining variables are set to 0. It is easy to verify that all constraints are satisfied by this solution.

If, on the other hand, we have a solution (x, y) to the MCIP with objective value at most $m\bar{T} - L$, we reserve $\sum_j x_C^{(j)}$ machines for each configuration C. There are enough machines to do this, because of (9.13). On each of these machines we reserve space: For each $G \in \Gamma$, we create an allocated space of length $\Lambda(G)$ starting from the starting layer of G, if $C_G = 1$. Let j be a job and ℓ be a layer. If j has a small setup time, we create $y^{(j)}_{(\ell,\varepsilon\delta T,0,0)}$ pieces of length $\varepsilon\delta T$ and place these pieces into allocated spaces of length $\varepsilon\delta T$ in layer ℓ. If, on the other hand, j is a big or medium setup job, we consider each possible job part length $q \in Q^{\text{bst}}$ or $q \in Q^{\text{mst}}$, create $y^{(j)}_{(\ell,q,s(j),0)}$ or $y^{(j)}_{(\ell,q,s(j),b)}$, with $b = \lceil q/(\varepsilon\delta T)\rceil\varepsilon\delta T - \varepsilon\delta T$, pieces of length q, and place them together with their setup time into allocated spaces of length q in layer ℓ. Because of (9.15) the entire job is split up by this, and because of (9.14) there are enough allocated spaces for all the job pieces.

The makespan bound is ensured by the definition of the modules, and overlaps are avoided, due to the definition of the configurations and (9.16). Furthermore, the used slots have an overall length equal to the objective value of (x, y) and therefore there is at least L free space. □

Result

Summing up the above considerations, we get:

Algorithm 3.

1. *If there is no suitable class of medium setup jobs, report that there is no schedule with makespan T and terminate the procedure.*
2. *Generate the modified instance I_3:*
   - ▻ *Remove the small jobs with medium setup times.*
   - ▻ *Remove the small jobs with small setup times, and decrease the setup time of big jobs with small setup time to* 0.
   - ▻ *Round the big processing times, as well as the medium, and the big setup times.*
3. *Build and solve the* MCIP *for I_3.*
4. *If the* MCIP *is infeasible, or the objective value greater than $m\bar{T} - L$, report that I has no solution with makespan T.*
5. *Otherwise build the $\varepsilon\delta T$-layered schedule with makespan $\bar{T}$ and free space at least L for I_3.*
6. *Transform the schedule into a schedule for I with makespan at most $\breve{T}$:*
   - ▻ *Use the prerounding processing and setup times.*
   - ▻ *Insert the small jobs with small setup times into the free slots and insert the setup times of the big jobs with small setup times.*
   - ▻ *Insert the small jobs with medium setup times.*

We analyze the running time of the procedure, and start by bounding the parameters of the extended MCIP. We have $|\mathcal{B}| = n$ and $D = 1$ by definition, and the number of layers $|\Xi|$ is obviously $\mathcal{O}(1/(\varepsilon\delta)) = \mathcal{O}(1/\varepsilon^{2/\varepsilon+1}) = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$. Furthermore, it is easy to see that

$$\begin{aligned} |Q^{\text{bst}}| &= \mathcal{O}(1/(\varepsilon\delta)), \\ |Q^{\text{mst}}| &= \mathcal{O}(1/(\varepsilon^3\delta)), \\ |S^{\text{bst}}| &= \mathcal{O}(1/(\varepsilon\delta)), \\ |S^{\text{mst}}| &= \mathcal{O}(1/(\varepsilon^3)), \text{ and} \\ |B| &= \mathcal{O}(1/\varepsilon^2). \end{aligned}$$

This gives us

$$\begin{aligned} \mathcal{M}^{\text{bst}} &\leqslant |\Xi||Q^{\text{bst}}||S^{\text{bst}}| = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)} \\ \mathcal{M}^{\text{mst}} &\leqslant |\Xi||Q^{\text{mst}}||S^{\text{mst}}||B| = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}, \text{ and} \\ \mathcal{M}^{\text{sst}} &= |\Xi| = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}. \end{aligned}$$

Therefore, we have $|\mathcal{M}| = |\mathcal{M}^{\text{bst}}| + |\mathcal{M}^{\text{mst}}| + |\mathcal{M}^{\text{sst}}| = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$. Since there are $\mathcal{O}(1/(\delta\varepsilon))$ distinct module sizes, the number of groups $|\Gamma|$ can be bounded by $\mathcal{O}(|\Xi|/(\varepsilon\delta)) = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$. Hence, for the number of configurations we get $|\mathcal{C}| = \mathcal{O}((1/(\varepsilon\delta))^{|\Gamma|}) = 2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}}$. By Observation 9.18, the modified MCIP has $r = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ many globally and $s = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ many locally uniform constraints; its brick number is n, and its brick size is $t = 2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}}$. All occurring values in the matrix are bounded by $\bar{T}$, yielding $\Delta \leqslant \bar{T} = 1/(\varepsilon\mu) + 1/\mu = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$, due to the scaling step. Furthermore, the numbers in the input can be bounded by $m2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ and all variables can be upper bounded by $\mathcal{O}(m)$. Hence, we have $\varphi = \mathcal{O}(\log m + 1/\varepsilon \log 1/\varepsilon)$ and $\Phi = \mathcal{O}(m)$, and due to Theorem 9.3 we can solve the MCIP in time

$$(rs\Delta)^{\mathcal{O}(r^2s+rs^2)} t^2 n^2 \varphi \log(\Phi) \log(nt\Phi) = 2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}} n^2 \log^2 m \log nm.$$

A straight-forward realization of the procedure for the creation of the $\varepsilon\delta T$-layered $(\bar{T}, L)$-schedule for I_3 (the fifth step), which is described in the proof of Lemma 9.19, will take $nm2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ time, yielding:

Theorem 9.20. *The algorithm for the preemptive model finds a schedule with makespan at most* $(1+9\varepsilon)T$ *or correctly determines that there is no schedule with makespan* T *in time* $2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}} n^2 m \log m \log nm$.

Proof of Lemma 9.17

We divide the proof into three steps, which can be summarized as follows:

1. We transform a (T, L)-schedule for I_2 into a $((1+3\varepsilon)T, L)$-schedule for I_3 in which the big setup jobs are already properly placed inside the layers.
2. We construct a flow network with integer capacities and a maximum flow, based on the placement of the remaining jobs in the layers.
3. Using flow integrality and careful repacking, we transform the schedule into a $\varepsilon\delta T$-layered, layer-compliant schedule.

More precisely the above transformation steps will produce a $\varepsilon\delta T$-layered, layer-compliant $((1+3\varepsilon)T, L)$-schedule with following additional properties. It might happen that too much processing time is inserted for some jobs, or setup times are produced that are not followed by the corresponding job pieces. Note that this does not cause any problems: We can simply remove the extra setups and processing time pieces. For the medium jobs this results in a placement with at most one used slot that is not fully filled, as required in a layer-compliant schedule.

Step 1. Remember that a block is a job piece together with its setup time placed in a given schedule. Consider a (T, L)-schedule for I_2 and suppose that for each block in the schedule there is a container perfectly encompassing it. Now, we stretch the entire schedule by a factor of $(1+3\varepsilon)$ and in this process we stretch and move the containers correspondingly. The blocks are not stretched but moved in order to stay in their container, and we assume that they are positioned at the bottom, that is, at the beginning of the container. Note that we could move each block inside its respective container without creating conflicts with other blocks belonging to the same job. In the following, we use the extra space to modify the

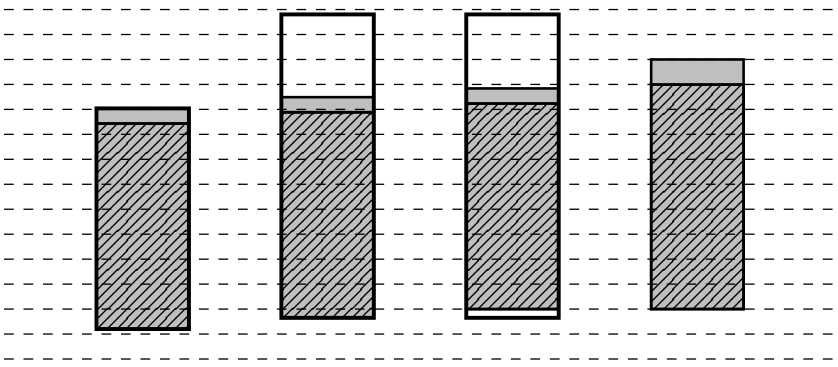


Figure 9.2. The stretching and rounding steps, for a small job part with big setup time starting in the first layer of the schedule, depicted from left to right: The schedule and the containers are stretched; the block is moved up; and the processing and the setup time are increased. The hatched part represents the setup time, the thick rectangle the container, and the dashed lines the layers, with $\varepsilon = \delta = 1/8$.

schedule. Similar techniques are widely used in the context of geometric packing algorithms.

Let j be a big setup job. In each container containing a block belonging to j, there is a free space of at least $3\varepsilon\delta T$, because the setup time of j is at least δT and therefore the container had at least that length before the stretching. Hence, we have enough space to perform the following two steps. We move the block up by at most $\varepsilon\delta T$ such that it starts at a multiple of $\varepsilon\delta T$. Next, we enlarge the setup time and the processing time by at most $\varepsilon\delta T$ such that both are multiples of $\varepsilon\delta T$. Now the setup time is equal to the rounded setup time, while the processing time might be bigger, because we performed this step for each piece of the job. We outline the procedure in Figure 9.2.

We continue with the small setup jobs. These jobs are big and therefore for each of them there is a summed up free space of at least $3\varepsilon^2 T$ in the containers belonging to the respective job – more than enough to enlarge some of the pieces such that their overall length matches the rounded processing time.

Lastly, we consider the medium setup jobs. These jobs are big as well and we could apply the same argument as above, however, we need to be a little bit more careful in order to additionally realize the rounding

of the setup times and an additional technical step, that we need in the following. Fix a medium setup job j and a container filled with a block belonging to j. Since the setup time has a length of at least μT, the part of the container filled with it was increased by at least $3\varepsilon\mu T$. Hence, we can enlarge the setup time to the rounded setup time without using up space in the container that was created due to the processing time part. We do this for all blocks belonging to medium setup jobs. The extra space in the containers of a medium setup job due to the processing time parts is still at least $3\varepsilon^2 T \geqslant 3\varepsilon\delta T$. For each medium setup job j we spend at most $\varepsilon\delta T$ of this space to enlarge its processing time to its rounded size and again at most $\varepsilon\delta T$ to create a little bit of extra processing time in the containers belonging to j. The size of this extra processing time is bounded by $\varepsilon\delta T$ and chosen in such a way that the overall length of all blocks belonging to j in the schedule is also a multiple of $\varepsilon\delta T$. Because of the rounding, the length of the added extra processing time for each j is a multiple of $\varepsilon\mu T$. The purpose of the extra processing time is to ensure integrality in the flow network, which is constructed in the next step.

Note that the free space that was available in the original schedule was not used in the above steps, in fact it was even increased by the stretching. Hence, we have created a $((1+3\varepsilon)T, L)$-schedule for I_3 – or a slightly modified version thereof – and the big setup jobs are already well behaved with respect to the $\varepsilon\delta T$-layers, that is, they start at multiples of $\varepsilon\delta T$, and fully fill the slots they are scheduled in.

Step 2. Note that for each job j and layer $\ell \in \Xi$, the overall length $q_{j,\ell}$ of job and setup pieces belonging to j and placed in ℓ is bounded by $\varepsilon\delta T$. We say that j is *fully*, or *partially*, or *not* scheduled in layer ℓ, if $q_{j,\ell} = 1$, or $q_{j,\ell} \in (0,1)$, or $q_{j,\ell} = 0$ respectively. Let X_j be the set of layers in which j is scheduled partially and Y_ℓ the set of (medium or small setup) jobs partially scheduled in ℓ. Then $a_j = \sum_{\ell \in X_j} q_{j,\ell}$ is a multiple of $\varepsilon\delta T$ and we set $n_j = a_j/(\varepsilon\delta T)$. Furthermore, let $b_\ell = \sum_{j \in Y_\ell} q_{j,\ell}$ and $k_\ell = \lceil b_\ell/(\varepsilon\delta T) \rceil$.

Our flow network has the following structure: There is a node v_j for each medium or small setup job, and a node u_ℓ for each layer ℓ, as well as a source α and a sink ω. The source node is connected to the job nodes via edges (α, v_j) with capacity n_j; and the layer nodes are connected to

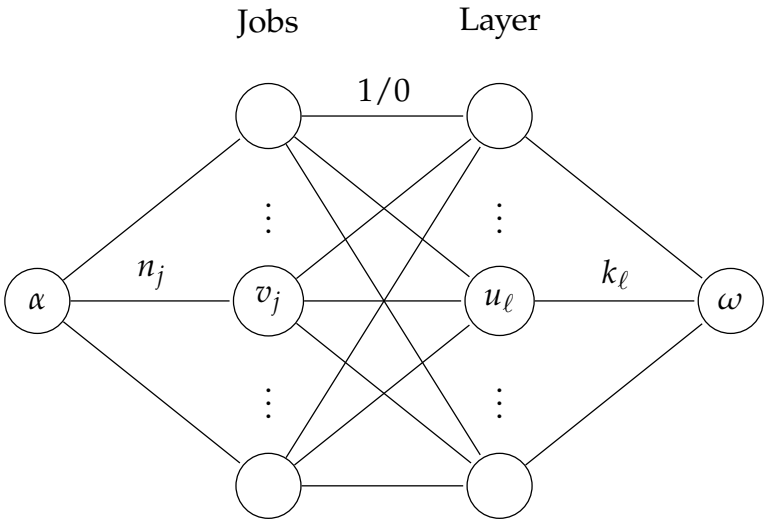


Figure 9.3. Flow network for layers and partially scheduled jobs.

the sink via edges (u_ℓ, ω) with capacity k_ℓ. Lastly, there are edges (v_j, u_ℓ) between job and layer nodes with capacity 1, if j is partially scheduled in layer ℓ, or 0 otherwise. In Figure 9.3 a sketch of the network is given.

The schedule can be used to define a flow f with value $\sum_j n_j$ in the network, by setting $f(\alpha, v_j) = n_j$, $f(u_\ell, \omega) = b_\ell/(\varepsilon\delta T)$, and $f(v_j, u_\ell) = q_{j,\ell}/(\varepsilon\delta T)$. It is easy to verify that f is a maximum flow, and because all capacities in the flow network are integral, we can find another maximum flow f' with integral values.

Step 3. We start by introducing some notation and a basic operation for the transformation of the schedule: Given two machines i and i' and a time t, a *machine swap* between i and i' at moment t produces a schedule, in which everything that was scheduled on i from t on is now scheduled on i' and vice versa. If on both machines there is either nothing scheduled at t, or blocks are starting or ending at t, the resulting schedule is still feasible. Moreover, if there is a block starting at t on one of the machines and another one belonging to the same job ending on the other we can merge the two blocks and transform the setup time of the first into processing time. We assume in the following that we always merge if this is possible, when performing a machine swap. Remember that by definition blocks

belonging to the same job cannot overlap. However, if there was overlap, it could be eliminated using machine swaps [97].

If a given slot only contains pieces of jobs that are partially scheduled in the layer, we call the slot *usable*. Furthermore, we say that a job j is *flow assigned* to layer ℓ, if $f'(v_j, u_\ell) = 1$. In the following, we will iterate through the layers, and create as many usable slots as possible, reserve them for flow assigned jobs, and fill them with processing and setup time of the corresponding slot later on. To do so, we have to distinguish different types of blocks belonging to jobs that are partially placed in a given layer: Inner blocks, which lie completely inside the layer and touch at most one of its borders; upper cross-over blocks, which start inside the layer and end above it; and lower cross-over blocks, which start below the layer and end inside it. When manipulating the schedule layer by layer, the cross-over jobs obviously can cause problems. To deal with this, we will need additional concepts: A *repair piece* for a given block is a piece of setup time of length less than $\varepsilon\delta T$, with the property that the block and the repair piece together make up exactly one setup of the respective job. Hence, if a repair-piece is given for a block, the block is comprised completely of setup time. Moreover, we say that a slot reserved for a job j has a *dedicated setup* if there is a block of j including a full setup *starting or ending* inside the slot.

In the following, we give a detailed description of the transformation procedure followed by a high-level overview of the procedure. The procedure runs through two phases. In the first phase the layers are transformed one after another from bottom to top. After a layer is transformed the following invariants will always hold:

1. A scheduled block either includes a full setup, or has a repair piece, and in the latter case it was an upper cross-over block in a previous iteration.

2. Reserved slots that are not full have a dedicated setup.

Note that the invariants are trivially fulfilled in the beginning. During the first phase, we remove some job and setup parts from the schedule that are reinserted into the reserved slots in the second phase. Let $\ell \in \Xi$ denote the current layer.

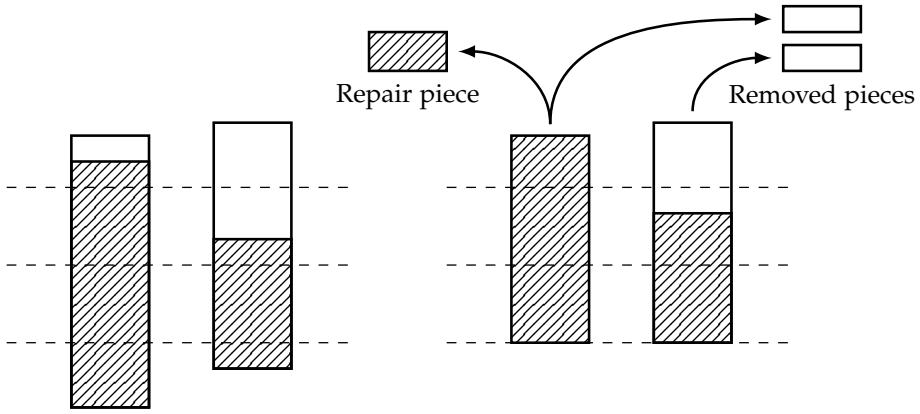


Figure 9.4. The rectangles represent blocks, the hatched parts the setup times, and the dashed lines layer borders. The push and cut step is performed on two blocks. For one of the two a repair piece is created.

In the first step, our goal is to ensure that jobs that are fully scheduled in ℓ occupy exactly one slot, thereby creating as many usable slots as possible. Let j be a job that is fully scheduled in layer ℓ. If there is a block belonging to j and ending inside the layer at time t, there is another block belonging to j and starting at t, because j is fully scheduled in ℓ and there are no overlaps. Hence, we can perform a machine swap at time t between the two machines the blocks are scheduled on. We perform this swap for each job fully scheduled in the layer and each corresponding pair of blocks. After this step, there are at least k_ℓ usable slots and at most k_ℓ flow assigned jobs in layer ℓ.

Next, we consider upper cross-over blocks of jobs that are partially scheduled in the layer ℓ but are not flow assigned to it. These are the blocks that cause the most problems, and we perform a so-called *push and cut step* (see Figure 9.4) for each of them: If q is the length of the part of the block lying in ℓ, we cut away the upper part of the block of length q and move the remainder up by q. If the piece we cut away does contain some setup time, we create a repair piece for the block out of this setup time. The processing time part of the piece, on the other hand, is removed. Note that this step preserves the first invariant. The repair piece is needed in the case that the job corresponding to the respective block is flow assigned to the layer in which the block ends.

We now remove all inner blocks from the layer, as well as the parts of the upper and lower cross-over blocks that lie in the layer. After this, all usable slots are completely free. Furthermore, note that the the first invariant might be breached by this.

Next, we arbitrarily reserve usable slots for jobs flow assigned to the layer. For this, note that due to the definition of the flow network, there are at most k_ℓ jobs flow assigned to the layer and there are at least as many usable slots, as noted above.

Using machine swaps at the upper and lower border of the layer, we then ensure that the upper and lower cross-over blocks of the jobs flow assigned to the layer lie on the same machine as the reserved slot. This step might breach the second invariant as well.

However, for each job j flow assigned to the layer, we perform the repair steps in order to restore the invariants: If there is an upper cross-over block for j, we reinsert the removed part of the block at the end of the slot, thereby providing a dedicated setup for the remaining free space in the slot. If there is a lower, but no upper cross-over block for j, there are two cases: Either there was a repair piece for the block or not. In both cases we reinsert the removed part of the block in the beginning of the slot and in the first we additionally insert as much setup of the repair piece as possible. The possible remainder of the repair piece is removed. Now the slot is either full, or a full setup is provided. If there is neither an upper nor a lower block for j, there is an in inner block belonging to j. This has to be the case, because otherwise the capacity in the flow network between j and ℓ is 0 and j could not have been flow assigned to ℓ. Moreover, this inner block contains a full setup and we can place it in the beginning of the slot, thus providing the dedicated setup. The invariants are both restored.

After the first phase is finished, we have to deal with the removed pieces in the second one. The overall length of the reserved slots for a job j equals the overall length a_j of its setup and job pieces from layers in which j was partially scheduled. Since we did not create or destroy any job piece, we can place the removed pieces corresponding to job j perfectly into the remaining free space of the slots reserved for j, and we do so after transforming them completely into processing time. Because of the second invariant, there is a dedicated setup in each slot, however, it may be positioned directly above the newly inserted processing time.

This can be fixed by switching the processing time with the top part of the respective setup time.

Lastly, all remaining usable slots are completely free at the end of this procedure, and since the others are full they have an overall size of at least L. We conclude the proof of Lemma 9.17 with an overview of the transformation procedure.

Algorithm 4.
Phase 1: *For each layer $\ell \in \Xi$, considered bottom to top, perform the following steps:*

1. *Use machine swaps to ensure that jobs fully scheduled in ℓ occupy exactly one slot.*
2. *For each upper cross-over block of a job partially scheduled but not flow assigned to ℓ perform a push and cut step.*
3. *Remove inner blocks and parts of cross-over blocks that lie in ℓ.*
4. *Reserve usable slots for jobs flow assigned to the layer.*
5. *Use machine swaps to ensure that cross-over blocks of flow assigned jobs lie on the same machine as the reserved slot.*
6. *For each job j flow assigned to the layer, perform exactly one of the repair steps.*

Phase 2:

1. *Transform all removed pieces into processing time and insert the removed pieces into the reserved slots.*
2. *If processing time has been inserted ahead of the dedicated setup of the slot, reschedule properly.*

9.5 Improvements of the running time

In this section, we revisit the splittable and the setup time model. For the former, we address the problem of the running time dependence in the number of machines m, and for both we present an improved rounding procedure, yielding a better running time.

9.5.1 Splittable Model – Machine Dependence

In the splittable model, the number of machines m might be super-polynomial in the input size, because it is not bounded by the number of jobs n. Hence, we need to be careful already when defining the schedule in order to get a polynomially bounded output. We say a machine is *composite* if it contains more than one job, and we say it is *plain* if it contains at most one job. For a schedule with makespan T, we call each machine *trivial* if it is plain and has load T or if it is empty, and *nontrivial* otherwise. We say a schedule with makespan T is *simple*, if the number of nontrivial machines is bounded by $\binom{n}{2}$.

Lemma 9.21. *If there is a schedule with makespan T for I there is also a simple schedule with makespan T.*

Proof. Let there be a schedule S with makespan T for I. For the first step, let us assume there are more than $\binom{n}{2}$ composite machines. In this case, there exist two machines M_1 and M_2 and two jobs $a, b \in \mathcal{J}, a \neq b$ such that both machines contain parts of both jobs since there are at most $\binom{n}{2}$ different pairs of jobs. Let $t_{M_x}(y)$ be the processing time combined with the setup time of job $y \in \{a, b\}$ on machine M_x, $x \in \{1, 2\}$. W.l.o.g. let $t_{M_1}(a)$ be the smallest value of the four. We swap this job part and its setup time with some of the processing time of the job b on machine M_2. If the processing time of b on M_2 is smaller than $t_{M_1}(a)$, there is no processing time of b on M_2 left and we can discard the setup time from b on this machine. We can repeat this step iteratively until there are at most $\binom{n}{2}$ machines containing more than one job.

In the second step, we shift processing time from the composite machines to the plain ones. We do this for each job until it is either not contained on a composite machine or each plain machine containing this job has load T. If the job is no longer contained on a composite machine, we shift the processing time of the job such that all except one machine containing this job has load T. Since this job does not appear on any composite machine, their number can be bounded by $\binom{n-1}{2}$ by repeating the first step. Therefore, the number of nontrivial machines is bounded by $\binom{n-i}{2} + i \leqslant \binom{n}{2}$ for some $i \in \{0, \ldots, n\}$. □

For a simple schedule a polynomial representation of the solution is

possible: For each job, we give the number of trivial machines containing this job, or fix a first and last trivial machine belonging to this job. This enables a polynomial encoding length of the output, given that the remaining parts of the jobs are not fragmented into too many parts, which can be guaranteed using the results of Section 9.4.

To guarantee that the MCIP finds a simple solution, we need to modify it a little. We have to ensure that nontrivial configurations are not used to often. We can do this by summing up the number of those configurations and bound them by $\binom{n}{2}$. Let $\mathcal{C}' \subseteq \mathcal{C}$ be the set of nontrivial configurations, i.e., the set of configurations containing more than one module or one module with size smaller than T. We add the following globally uniform constraint to the MCIP:

$$\sum_{C \in \mathcal{C}'} x_C \leqslant \binom{|\mathcal{J}^{\text{bst}}|}{2} \tag{9.17}$$

Since this is an inequality, we have to introduce a slack variable increasing the brick size by one. Furthermore, the bound on the biggest number occurring in the input as well as the range of the variables has to be increased by a factor of $\mathcal{O}(n^2)$, yielding a slightly altered running time for the MCIP of

$$2^{\mathcal{O}(1/\varepsilon^4 \log 1/\varepsilon)} n^2 \log^3(nm).$$

The number of modules with maximum size denotes for each job in $\mathcal{J}^{\text{bst}}$ how many trivial machines it uses. The other modules can be mapped to the nontrivial configurations and the jobs can be mapped to the modules.

We still have to schedule the jobs in $\mathcal{J}^{\text{sst}}$. We do this as described in the proof of Lemma 9.11. We fill the nontrivial machines greedily step by step starting with the jobs having the smallest processing time. When these machines are filled, there are some completely empty machines left. Now, we estimate how many machines can be completely filled with the current job j. This can be done by dividing the remaining processing time by $T - s(i)$ in $\mathcal{O}(1)$. The remaining part is scheduled on the next free machine. This machine is filled up with the next job and again the number of machines which can be filled completely with the rest of this new job is determined. These steps are iterated until all jobs in $\mathcal{J}^{\text{sst}}$ are scheduled. This greedy procedure needs at most $\mathcal{O}(|\mathcal{J}^{\text{bst}}|(|\mathcal{J}^{\text{bst}}| - 1) + |\mathcal{J}^{\text{sst}}|) =$

$\mathcal{O}(n^2)$ operations. Therefore, we can avoid the dependence in the number of machines and the overall running time is dominated by the time it takes to solve the MCIP.

9.5.2 Improved Rounding Procedures

To improve the running time in the splittable and setup class model, we reduce the number of module sizes via a geometric and an arithmetic rounding step. In both cases, the additional steps are performed following all the other simplification steps. The basic idea is to include setup times together with their corresponding job pieces or batches of jobs respectively into containers with suitably rounded sizes and to model these containers using the modules. The containers have to be bigger in size than the objects they contain and the load on a machine is given by the summed up sizes of the containers on the machine. Let H^* be a set of container sizes. Then a H^*-structured schedule is a schedule in which each setup time together with its corresponding job piece or batch of jobs is packed in a container with the smallest size $h \in H^*$ such that the summed up size of the setup time and the job piece or batch of jobs is upper bounded by h.

Splittable Model. Consider the instance I_2 for the splittable model described in Section 9.4.2. In this instance, each setup and processing time is a multiple of $\varepsilon^2 T$ and we are interested in a schedule of length $(1+2\varepsilon)T$. For each multiple h of $\varepsilon^2 T$, let $\tilde{h} = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} h/(\varepsilon^2 T)\rceil}\varepsilon^2 T$ and $\bar{h} = \lceil \tilde{h}/\varepsilon^2 T\rceil \varepsilon^2 T$, and $\bar{H} = \{\bar{h} \mid h \in \varepsilon^2 T\mathbb{Z}_{\geqslant 1}, h \leqslant (1+2\varepsilon)^2 T\}$. Note that $|\bar{H}| \in \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$

Lemma 9.22. *If there is a $((1+2\varepsilon)T, L')$-schedule for I_2 in which the length of each job part is a multiple of $\varepsilon^2 T$, there is also a $\bar{H}$-structured $((1+2\varepsilon)^2 T, L')$-schedule for I_2 with the same property.*

Proof. Consider such a schedule for I_2 and a pair of setup time s and job piece q scheduled on some machine. Let $h = s + q$. Stretching the schedule by $(1+2\varepsilon)$ creates enough space to place the pair into a container of size $\bar{h}$, because $(1+\varepsilon)h \leqslant \tilde{h}$, and $\varepsilon h \leqslant \varepsilon^2 T$ since $s \geqslant \varepsilon T$. □

To implement this lemma into the procedure, the processing time bounds $\bar{T}$ and $\check{T}$ both have to be properly increased. Modeling a $\bar{H}$-structured schedule can be done quite naturally: We simply redefine the size $\Lambda(M)$ of a module $M = (s, q) \in \mathcal{M}$ to be $s \dotplus q$. With this definition, we have $|H| = |\bar{H}| = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$, yielding an improved running time for solving the MCIP of

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

Combining this with the results above and the considerations in Section 9.4.2 yields the running time claimed below Theorem 9.1.

Setup Class Model. In the setup class model, an analogue approach also yields a reduced set of module sizes, that is, $|H| = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$. Therefore, the MCIP can be solved in time

$$2^{\mathcal{O}(1/\varepsilon^3 \log^4 1/\varepsilon)} K^2 \log(n) \log(m) \log(Km).$$

Hence, we get the running time claimed beneath Theorem 9.1.

9.6 Conclusion

We presented a more advanced version of the classical configuration IP, showed that it can be solved efficiently using algorithms for n-fold IPs, and developed techniques to employ the new IP for the formulation of efficient polynomial time approximation schemes for three scheduling problems with setup times, for which no such algorithms were known before.

For further research the immediate questions are whether improved running times for the considered problems, in particular for the preemptive model, can be achieved; whether the MCIP can be solved more efficiently; and to which other problems it can be reasonably employed. From a broader perspective, it would be interesting to further study the potential of new algorithmic approaches in integer programming for approximation, and, on the other hand, further study the respective techniques.

Bibliography

[1] Anna Adamaszek, Tomasz Kociumaka, Marcin Pilipczuk, and Michal Pilipczuk. "Hardness of approximation for strip packing". In: *TOCT* 9.3 (2017), 14:1–14:7. DOI: 10.1145/3092026.

[2] Ali Allahverdi, Jatinder ND Gupta, and Tariq Aldowaisan. "A review of scheduling research involving setup considerations". In: *Omega* 27.2 (1999), pp. 219–239.

[3] Ali Allahverdi, CT Ng, TC Edwin Cheng, and Mikhail Y Kovalyov. "A survey of scheduling problems with setup times or costs". In: *European journal of operational research* 187.3 (2008), pp. 985–1032.

[4] Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. "Approximation schemes for scheduling". In: *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 5-7 January 1997, New Orleans, Louisiana, USA.* 1997, pp. 493–500. URL: http://dl.acm.org/citation.cfm?id=314161.314371.

[5] Noga Alon, Yossi Azar, Gerhard J Woeginger, and Tal Yadid. "Approximation schemes for scheduling on parallel machines". In: *Journal of Scheduling* 1.1 (1998), pp. 55–66.

[6] Abdel Krim Amoura, Evripidis Bampis, Claire Kenyon, and Yannis Manoussakis. "Scheduling independent multiprocessor tasks". In: *Algorithmica* 32.2 (2002), pp. 247–261. DOI: 10.1007/s00453-001-0076-9.

[7] Brenda S. Baker, Donna J. Brown, and Howard P. Katseff. "A 5/4 algorithm for two-dimensional packing". In: *Journal of Algorithms* 2.4 (1981), pp. 348–368. DOI: 10.1016/0196-6774(81)90034-1.

[8] Brenda S. Baker, Edward G. Coffman Jr., and Ronald L. Rivest. "Orthogonal packings in two dimensions". In: *SIAM Journal on Computing* 9.4 (1980), pp. 846–855. DOI: 10.1137/0209064.

[9] Peter A. Beling and Nimrod Megiddo. "Using fast matrix multiplication to find basic solutions". In: *Theor. Comput. Sci.* 205.1-2 (1998), pp. 307–316. DOI: 10.1016/S0304-3975(98)00003-6.

[10] Iwo Bladek, Maciej Drozdowski, Frédéric Guinand, and Xavier Schepler. "On contiguous and non-contiguous parallel task scheduling". In: *J. Scheduling* 18.5 (2015), pp. 487–495. DOI: 10.1007/s10951-015-0427-z.

[11] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. "Time bounds for selection". In: *J. Comput. Syst. Sci.* 7.4 (1973), pp. 448–461. DOI: 10.1016/S0022-0000(73)80033-9. URL: https://doi.org/10.1016/S0022-0000(73)80033-9.

[12] Marin Bougeret, Pierre-François Dutot, Klaus Jansen, Christina Otte, and Denis Trystram. "A fast 5/2-approximation algorithm for hierarchical scheduling". In: *Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy, August 31 - September 3, 2010, Proceedings, Part I*. 2010, pp. 157–167. DOI: 10.1007/978-3-642-15277-1_16.

[13] Marin Bougeret, Pierre-François Dutot, Klaus Jansen, Christina Otte, and Denis Trystram. "Approximating the non-contiguous multiple organization packing problem". In: *Theoretical Computer Science - 6th IFIP TC 1/WG 2.2 International Conference, TCS 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings*. 2010, pp. 316–327. DOI: 10.1007/978-3-642-15240-5_23.

[14] Marin Bougeret, Pierre-François Dutot, Klaus Jansen, Christina Otte, and Denis Trystram. "Approximation algorithms for multiple strip packing". In: *Approximation and Online Algorithms, 7th International Workshop, WAOA 2009, Copenhagen, Denmark, September 10-11, 2009. Revised Papers*. 2009, pp. 37–48. DOI: 10.1007/978-3-642-12450-1_4.

[15] Marin Bougeret, Pierre-François Dutot, Klaus Jansen, Christina Robenek, and Denis Trystram. "Approximation algorithms for multiple strip packing and scheduling parallel jobs in platforms". In: *Discrete Mathematics, Algorithms and Applications* 3.4 (2011), pp. 553–586. DOI: 10.1142/S1793830911001413.

[16] Marin Bougeret, Pierre-François Dutot, Denis Trystram, Klaus Jansen, and Christina Robenek. "Improved approximation algorithms for scheduling parallel jobs on identical clusters". In: *Theor. Comput. Sci.* 600 (2015), pp. 70–85. DOI: 10.1016/j.tcs.2015.07.003.

[17] Alberto Caprara, Hans Kellerer, Ulrich Pferschy, and David Pisinger. "Approximation algorithms for knapsack problems with cardinality constraints". In: *European Journal of Operational Research* 123.2 (2000), pp. 333–345. DOI: 10.1016/S0377-2217(99)00261-1. URL: https://doi.org/10.1016/S0377-2217(99)00261-1.

[18] Bo Chen. "A better heuristic for preemptive parallel machine scheduling with batch setup times". In: *SIAM Journal on Computing* 22.6 (1993), pp. 1303–1318.

[19] Bo Chen, Yinyu Ye, and Jiawei Zhang. "Lot-sizing scheduling with batch setup times". In: *Journal of Scheduling* 9.3 (2006), pp. 299–310.

[20] Lin Chen, Klaus Jansen, and Guochuan Zhang. "On the optimality of approximation schemes for the classical scheduling problem". In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*. 2014, pp. 657–668. DOI: 10.1137/1.9781611973402.50. URL: https://doi.org/10.1137/1.9781611973402.50.

[21] Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. "Parameterized and approximation results for scheduling with a low rank processing time matrix". In: *LIPIcs-Leibniz International Proceedings in Informatics*. Vol. 66. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017.

[22] Edward G. Coffman Jr., Michael R. Garey, David S. Johnson, and Robert Endre Tarjan. "Performance bounds for level-oriented two-dimensional packing algorithms". In: *SIAM Journal on Computing* 9.4 (1980), pp. 808–826. DOI: 10.1137/0209062.

[23] José Correa, Alberto Marchetti-Spaccamela, Jannik Matuschke, Leen Stougie, Ola Svensson, Víctor Verdugo, and José Verschae. "Strong lp formulations for scheduling splittable jobs on unrelated machines". In: *Mathematical Programming* 154.1-2 (2015), pp. 305–328.

[24] José Correa, Victor Verdugo, and José Verschae. "Splitting versus setup trade-offs for scheduling to minimize weighted completion time". In: *Operations Research Letters* 44.4 (2016), pp. 469–473.

[25] Max A. Deppert and Klaus Jansen. "Near-linear approximation algorithms for scheduling problems with batch setup times". In: *CoRR* abs/1810.01223 (2018). arXiv: 1810.01223. URL: http://arxiv.org/abs/1810.01223.

[26] Maciej Drozdowski. "On the complexity of multiprocessor task scheduling". In: *Bulletin of the Polish Academy of Sciences Technical Sciences* 43.3 (1995).

[27] Jianzhong Du and Joseph Y.-T. Leung. "Complexity of scheduling parallel task systems". In: *SIAM Journal on Discrete Mathematics* 2.4 (1989), pp. 473–487. DOI: 10.1137/0402042.

[28] Pierre-François Dutot, Klaus Jansen, Christina Robenek, and Denis Trystram. "A $(2+\varepsilon)$-approximation for scheduling parallel jobs in platforms". In: *Euro-Par 2013 Parallel Processing - 19th International Conference, Aachen, Germany, August 26-30, 2013. Proceedings*. 2013, pp. 78–89. DOI: 10.1007/978-3-642-40047-6_11.

[29] Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. "Faster algorithms for integer programs with block structure". In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. 2018, 49:1–49:13.

[30] Leah Epstein and Asaf Levin. "AFPTAS results for common variants of bin packing: a new method for handling the small items". In: *SIAM Journal on Optimization* 20.6 (2010), pp. 3121–3145. DOI: 10.1137/090767613.

[31] Leah Epstein and Rob van Stee. "This side up!" In: *ACM Trans. Algorithms* 2.2 (2006), pp. 228–243. DOI: 10.1145/1150334.1150339.

[32] Anja Feldmann, Jirí Sgall, and Shang-Hua Teng. "Dynamic scheduling on parallel machines". In: *Theoretical Computer Science* 130.1 (1994), pp. 49–72. DOI: 10.1016/0304-3975(94)90152-X.

[33] Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, and Arindam Khan. "Improved pseudo-polynomial-time approximation for strip packing". In: *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. 2016, 9:1–9:14. DOI: 10.4230/LIPIcs.FSTTCS.2016.9.

[34] Michael R. Garey and Ronald L. Graham. "Bounds for multiprocessor scheduling with resource constraints". In: *SIAM Journal on Computing* 4.2 (1975), pp. 187–200. DOI: 10.1137/0204015.

[35] Michael R. Garey and David S. Johnson. "Complexity results for multiprocessor scheduling under resource constraints". In: *SIAM Journal on Computing* 4.4 (1975), pp. 397–411.

[36] Michael R. Garey and David S. Johnson. *Computers and intractability: a guide to the theory of np-completeness*. 1979. ISBN: 0-7167-1044-7.

[37] Paul C Gilmore and Ralph E Gomory. "A linear programming approach to the cutting-stock problem". In: *Operations research* 9.6 (1961), pp. 849–859.

[38] Michel X Goemans and Thomas Rothvoß. "Polynomiality for bin packing with a constant number of item types". In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pp. 830–839.

[39] Igal Golan. "Performance bounds for orthogonal oriented two-dimensional packing algorithms". In: *SIAM Journal on Computing* 10.3 (1981), pp. 571–582. DOI: 10.1137/0210042.

[40] Ronald L Graham. "Bounds for certain multiprocessing anomalies". In: *Bell System Technical Journal* 45.9 (1966), pp. 1563–1581.

[41] Michael D. Grigoriadis, Leonid G. Khachiyan, Lorant Porkolab, and J. Villavicencio. "Approximate max-min resource sharing for structured concave optimization". In: *SIAM Journal on Optimization* 11.4 (2001), pp. 1081–1091. DOI: 10.1137/S1052623499358689.

[42] Alexander Grigoriev, Maxim Sviridenko, and Marc Uetz. "Machine scheduling with resource dependent processing times". In: *Mathematical programming* 110.1 (2007), pp. 209–228.

[43] Alexander Grigoriev and Marc Uetz. "Scheduling jobs with time-resource tradeoff via nonlinear programming". In: *Discrete optimization* 6.4 (2009), pp. 414–419.

[44] Rolf Harren, Klaus Jansen, Lars Prädel, and Rob van Stee. "A $(5/3 + \epsilon)$-approximation for strip packing". In: *Computational Geometry* 47.2 (2014), pp. 248–267. DOI: 10.1016/j.comgeo.2013.08.008.

[45] Rolf Harren and Rob van Stee. "Improved absolute approximation ratios for two-dimensional packing problems". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, vol. 5687. Lecture Notes in Computer Science. Springer, 2009, pp. 177–189. DOI: 10.1007/978-3-642-03685-9_14.

[46] Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. "N-fold integer programming in cubic time". In: *Mathematical Programming* (2013), pp. 1–17.

[47] Sören Henning, Klaus Jansen, Malin Rau, and Lars Schmarje. "Complexity and inapproximability results for parallel task scheduling and strip packing". In: *Computer Science - Theory and Applications - 13th International Computer Science Symposium in Russia, CSR 2018, Moscow, Russia, June 6-10, 2018, Proceedings*. 2018, pp. 169–180. DOI: 10.1007/978-3-319-90530-3_15.

[48] Sören Henning, Klaus Jansen, Malin Rau, and Lars Schmarje. "Complexity and inapproximability results for parallel task scheduling and strip packing". In: *Theory of Computing Systems* (2019). ISSN: 1433-0490. DOI: 10.1007/s00224-019-09910-6. URL: https://doi.org/10.1007/s00224-019-09910-6.

[49] Dorit S. Hochbaum and David B. Shmoys. "Using dual approximation algorithms for scheduling problems theoretical and practical results". In: *J. ACM* 34.1 (1987), pp. 144–162. DOI: 10.1145/7531.7535.

[50] Klaus Jansen. "A $(3/2+\varepsilon)$ approximation algorithm for scheduling moldable and non-moldable parallel tasks". In: *24th ACM Symposium on Parallelism in Algorithms and Architectures, (SPAA)*. 2012, pp. 224–235. DOI: 10.1145/2312005.2312048.

[51] Klaus Jansen. "An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables". In: *SIAM J. Discrete Math.* 24.2 (2010), pp. 457–485. DOI: 10.1137/090749451. URL: https://doi.org/10.1137/090749451.

[52] Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. "Empowering the configuration-ip - new PTAS results for scheduling with setups times". In: *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*. 2019, 44:1–44:19. DOI: 10.4230/LIPIcs.ITCS.2019.44. URL: https://doi.org/10.4230/LIPIcs.ITCS.2019.44.

[53] Klaus Jansen, Kim-Manuel Klein, and José Verschae. "Closing the gap for makespan scheduling via sparsification techniques". In: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*. 2016, 72:1–72:13. DOI: 10.4230/LIPIcs.ICALP.2016.72.

[54] Klaus Jansen and Stefan Erich Julius Kraft. "A faster FPTAS for the unbounded knapsack problem". In: *Eur. J. Comb.* 68 (2018), pp. 148–174. DOI: 10.1016/j.ejc.2017.07.016.

[55] Klaus Jansen and Felix Land. "Non-preemptive scheduling with setup times: a ptas". In: *European Conference on Parallel Processing*. Springer. 2016, pp. 159–170.

[56] Klaus Jansen and Felix Land. "Scheduling monotone moldable jobs in linear time". In: *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*. 2018, pp. 172–181. DOI: 10.1109/IPDPS.2018.00027.

[57] Klaus Jansen, Marten Maack, and Malin Rau. "Approximation schemes for machine scheduling with resource (in-)dependent processing times". In: *ACM Trans. Algorithms* 15.3 (2019), 31:1–31:28. DOI: 10.1145/3302250.

[58] Klaus Jansen, Marten Maack, and Malin Rau. "Approximation schemes for machine scheduling with resource (in-)dependent processing times". In: *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*. 2016, pp. 1526–1542. DOI: 10.1137/1.9781611974331.ch104.

[59] Klaus Jansen and Lorant Porkolab. "Linear-time approximation schemes for scheduling malleable parallel tasks". In: *Algorithmica* 32.3 (2002), pp. 507–520. DOI: 10.1007/s00453-001-0085-8.

[60] Klaus Jansen and Malin Rau. "Closing the gap for pseudo-polynomial strip packing". In: *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany.* 2019, 62:1–62:14. DOI: 10.4230/LIPIcs.ESA.2019.62.

[61] Klaus Jansen and Malin Rau. "Closing the gap for pseudo-polynomial strip packing". In: *CoRR* abs/1705.04587 (2017). arXiv: 1712.04922. URL: https://arxiv.org/abs/1712.04922.

[62] Klaus Jansen and Malin Rau. "Improved approximation for two dimensional strip packing with polynomial bounded width". In: *Theoretical Computer Science* 789 (2019), pp. 34–49. DOI: 10.1016/j.tcs.2019.04.002.

[63] Klaus Jansen and Malin Rau. "Improved approximation for two dimensional strip packing with polynomial bounded width". In: *WALCOM: Algorithms and Computation*. Vol. 10167 of LNCS. 2017, pp. 409–420. DOI: 10.1007/978-3-319-53925-6_32.

[64] Klaus Jansen and Malin Rau. "Linear time algorithms for multiple cluster scheduling and multiple strip packing". In: *Euro-Par 2019: Parallel Processing - 25th International Conference on Parallel and Distributed Computing, Göttingen, Germany, August 26-30, 2019, Proceedings*. 2019, pp. 103–116. DOI: 10.1007/978-3-030-29400-7_8.

[65] Klaus Jansen and Malin Rau. "Linear time algorithms for multiple cluster scheduling and multiple strip packing". In: *CoRR* abs/1902.03428 (2019). arXiv: 1902.03428. URL: https://arxiv.org/abs/1902.03428.

[66] Klaus Jansen and Lars Rohwedder. "On integer programming and convolution". In: *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*. 2019, 43:1–43:17. DOI: 10.4230/LIPIcs.ITCS.2019.43.

[67] Klaus Jansen and Roberto Solis-Oba. "Rectangle packing with one-dimensional resource augmentation". In: *Discrete Optimization* 6.3 (2009), pp. 310–323. DOI: 10.1016/j.disopt.2009.04.001.

[68] Klaus Jansen and Rob van Stee. "On strip packing with rotations". In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*. 2005, pp. 755–761. DOI: 10.1145/1060590.1060702.

[69] Klaus Jansen and Ralf Thöle. "Approximation algorithms for scheduling parallel jobs". In: *SIAM Journal on Computing* 39.8 (2010), pp. 3571–3615. DOI: 10.1137/080736491.

[70] Klaus Jansen and Denis Trystram. "Scheduling parallel jobs on heterogeneous platforms". In: *Electronic Notes in Discrete Mathematics* 55 (2016), pp. 9–12. DOI: 10.1016/j.endm.2016.10.003.

[71] Berit Johannes. "Scheduling parallel jobs to minimize the makespan". In: *Journal of Scheduling* 9.5 (2006), pp. 433–452. DOI: 10.1007/s10951-006-8497-6.

[72] Ravi Kannan. "Minkowski's convex body theorem and integer programming". In: *Mathematics of operations research* 12.3 (1987), pp. 415–440.

[73] Mohammad M. Karbasioun, Gennady Shaikhet, Evangelos Kranakis, and Ioannis Lambadaris. "Power strip packing of malleable demands in smart grid". In: *Proceedings of IEEE International Conference on Communications, ICC 2013, Budapest, Hungary, June 9-13, 2013*, pp. 4261–4265. DOI: 10.1109/ICC.2013.6655233.

[74] Narendra Karmarkar and Richard M. Karp. "An efficient approximation scheme for the one-dimensional bin-packing problem". In: *23rd Annual Symposium on Foundations of Computer Science (FOCS), Chicago, Illinois, USA, 3-5 November 1982*. 1982, pp. 312–320.

[75] ShanXue Ke, BenSheng Zeng, WenBao Han, and Victor Y Pan. "Fast rectangular matrix multiplication and some applications". In: *Science in China Series A: Mathematics* 51.3 (2008), pp. 389–406.

[76] Hans Kellerer. "An approximation algorithm for identical parallel machine scheduling with resource dependent processing times". In: *Operations Research Letters* 36.2 (2008), pp. 157–159.

[77] Claire Kenyon and Eric Rémila. "A near-optimal solution to a two-dimensional cutting stock problem". In: *Mathematics of Operations Research* 25.4 (2000), pp. 645–656. DOI: 10.1287/moor.25.4.645.12118.

[78] Peter Kling, Alexander Mäcker, Sören Riechers, and Alexander Skopalik. "Sharing is caring: multiprocessor scheduling with a sharable resource". In: *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*. 2017, pp. 123–132.

[79] Dusan Knop and Martin Koutecký. "Scheduling meets n-fold integer programming". In: *Journal of Scheduling* (2017), pp. 1–11.

[80] Dusan Knop, Martin Koutecký, and Matthias Mnich. "Combinatorial n-fold integer programming and applications". In: *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*. 2017, 54:1–54:14.

[81] Martin Koutecký, Asaf Levin, and Shmuel Onn. "A parameterized strongly polynomial algorithm for block structured integer programs". In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. 2018, 85:1–85:14.

[82] Hendrik W Lenstra Jr. "Integer programming with a fixed number of variables". In: *Mathematics of operations research* 8.4 (1983), pp. 538–548.

[83] Walter Ludwig and Prasoon Tiwari. "Scheduling malleable and nonmalleable parallel tasks". In: *5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1994, pp. 167–176.

[84] Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. "Non-preemptive scheduling on machines with setup times". In: *Workshop on Algorithms and Data Structures*. Springer. 2015, pp. 542–553.

[85] Michael R. Marty and Mark D. Hill. "Virtual hierarchies to support server consolidation". In: *34th International Symposium on Computer Architecture (ISCA 2007), June 9-13, 2007, San Diego, California, USA*. 2007, pp. 46–56. DOI: 10.1145/1250662.1250670.

[86] Monaldo Mastrolilli and Marcus Hutter. "Hybrid rounding techniques for knapsack problems". In: *Discrete Applied Mathematics* 154.4 (2006). Efficient Algorithms Efficient Algorithms, pp. 640–649. ISSN: 0166-218X. DOI: http://dx.doi.org/10.1016/j.dam.2005.08.004.

[87] Nimrod Megiddo and Arie Tamir. "Linear time algorithms for some separable quadratic programming problems". In: *Operations Research Letters* 13.4 (1993), pp. 203–211. ISSN: 0167-6377. DOI: https://doi.org/10.1016/0167-6377(93)90041-E. URL: http://www.sciencedirect.com/science/article/pii/016763779390041E.

[88] Flávio Keidi Miyazawa and Yoshiko Wakabayashi. "Packing problems with orthogonal rotations". In: *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings*. 2004, pp. 359–368. DOI: 10.1007/978-3-540-24698-5_40.

[89] Clyde L Monma and Chris N Potts. "Analysis of heuristics for preemptive parallel machine scheduling with batch setup times". In: *Operations Research* 41.5 (1993), pp. 981–993.

[90] Gregory Mounie, Christophe Rapine, and Denis Trystram. "A 3/2-approximation algorithm for scheduling independent monotonic malleable tasks". In: *SIAM J. Comput.* 37.2 (2007), pp. 401–412. DOI: 10.1137/S0097539701385995.

[91] Giorgi Nadiradze and Andreas Wiese. "On approximating strip packing with a better ratio than 3/2". In: *27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2016, pp. 1491–1510. DOI: 10.1137/1.9781611974331.ch102.

[92] Martin Niemeier and Andreas Wiese. "Scheduling with an orthogonal resource constraint". In: *Algorithmica* 71.4 (2015), pp. 837–858. DOI: 10.1007/s00453-013-9829-5. URL: https://doi.org/10.1007/s00453-013-9829-5.

[93] Shmuel Onn. "Nonlinear discrete optimization". In: *Zurich Lectures in Advanced Mathematics, European Mathematical Society* (2010).

[94] Anshu Ranjan, Pramod P. Khargonekar, and Sartaj Sahni. "Offline first-fit decreasing height scheduling of power loads". In: *J. Scheduling* 20.5 (2017), pp. 527–542. DOI: 10.1007/s10951-017-0528-y.

[95] Frans Schalekamp, René Sitters, Suzanne Van Der Ster, Leen Stougie, Víctor Verdugo, and Anke Van Zuylen. "Split scheduling with uniform setup times". In: *Journal of scheduling* 18.2 (2015), pp. 119–129.

[96] Ingo Schiermeyer. "Reverse-fit: a 2-optimal algorithm for packing rectangles". In: *2nd Annual European Symposium on Algorithms (ESA) - Algorithms*. 1994, pp. 290–299. DOI: 10.1007/BFb0049416.

[97] Petra Schuurman and Gerhard J Woeginger. "Preemptive scheduling with job-dependent setup times". In: *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 1999, pp. 759–767.

[98] Uwe Schwiegelshohn, Andrei Tchernykh, and Ramin Yahyapour. "Online scheduling in grids". In: *22nd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, Miami, Florida USA, April 14-18, 2008*. 2008, pp. 1–10. DOI: 10.1109/IPDPS.2008.4536273.

[99] Daniel Dominic Sleator. "A 2.5 times optimal algorithm for packing in two dimensions". In: *Information Processing Letters* 10.1 (1980), pp. 37–40. DOI: 10.1016/0020-0190(80)90121-0.

[100] A. Steinberg. "A strip-packing algorithm with absolute performance bound 2". In: *SIAM Journal on Computing* 26.2 (1997), pp. 401–409. DOI: 10.1137/S0097539793255801.

[101] Maxim Sviridenko. "A note on the kenyon-remila strip-packing algorithm". In: *Inf. Process. Lett.* 112.1-2 (2012), pp. 10–12. DOI: 10.1016/j.ipl.2011.10.003.

[102] Shaojie Tang, Qiuyuan Huang, Xiang-Yang Li, and Dapeng Wu. "Smoothing the energy consumption: peak demand reduction in smart grid". In: *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*. 2013, pp. 1133–1141. DOI: 10.1109/INFCOM.2013.6566904.

[103] Andrei Tchernykh, Juan Manuel Ramírez-Alcaraz, Arutyun Avetisyan, Nikolai Kuzjurin, Dmitry Grushin, and Sergey Zhuk. "Two level job-scheduling strategies for a computational grid". In: *Parallel Processing and Applied Mathematics, 6th International Conference, PPAM 2005, Poznan, Poland, September 11-14, 2005, Revised Selected Papers*. 2005, pp. 774–781. DOI: 10.1007/11752578_93.

[104] Andrei Tchernykh, Uwe Schwiegelshohn, Ramin Yahyapour, and Nikolai Kuzjurin. "On-line hierarchical job scheduling on grids with admissible allocation". In: *J. Scheduling* 13.5 (2010), pp. 545–552. DOI: 10.1007/s10951-010-0169-x.

[105] John Turek, Joel L. Wolf, and Philip S. Yu. "Approximate algorithms scheduling parallelizable tasks". In: *4th annual ACM symposium on Parallel algorithms and architectures (SPAA)*. 1992, pp. 323–332. DOI: 10.1145/140901.141909.

[106] Wenceslas Fernandez de la Vega and George S. Lueker. "Bin packing can be solved within 1+epsilon in linear time". In: *Combinatorica* 1.4 (1981), pp. 349–355. DOI: 10.1007/BF02579456.

[107] Deshi Ye, Xin Han, and Guochuan Zhang. "Online multiple-strip packing". In: *Theor. Comput. Sci.* 412.3 (2011), pp. 233–239. DOI: 10.1016/j.tcs.2009.09.029.

[108] Minyi Yue. "On the exact upper bound for the multifit processor scheduling algorithm". In: *Annals of Operations Research* 24.1 (1990), pp. 233–259.

[109] SN Zhuk. "Approximate algorithms to pack rectangles into several strips". In: *Discrete Mathematics and Applications dma* 16.1 (2006), pp. 73–85. DOI: 10.1515/156939206776241264.