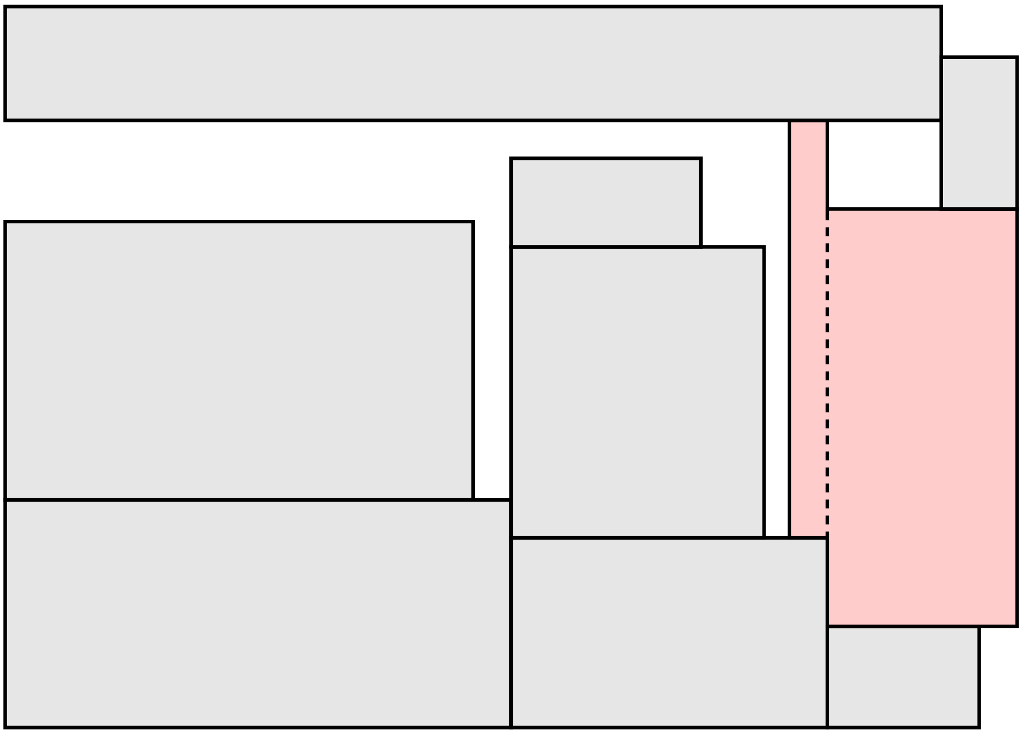


An approximation algorithm for Demand Strip Packing with moldable jobs

Master's thesis in Engineering Mathematics and Computational Science

WILLIAM BERGQUIST

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

An approximation algorithm for Demand Strip Packing with moldable jobs

WILLIAM BERGQUIST

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

An approximation algorithm for Demand Strip Packing with moldable jobs
WILLIAM BERGQUIST

Supervisor: Albert Vesterlund, Department of Mathematical Sciences
Examiner: Malin Rau, Department of Mathematical Sciences

Degree project report 2026
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: An example of a packing where a job has been vertically sliced, as is permitted in DSP.

Typeset in LaTeX
Printed by Chalmers Digital Printing
Gothenburg, Sweden 2026

An approximation algorithm for Demand Strip Packing with moldable jobs
WILLIAM BERGQUIST
Department of Mathematical Sciences
Chalmers University of Technology
University of Gothenburg

Abstract

In Strip Packing (SP), the aim is to place a set of rectangles, often called jobs, inside a strip with a bounded width D and infinite height with no overlaps whilst minimizing the height needed to fit all jobs. In Demand Strip Packing (DSP), we relax this problem by allowing jobs to be sliced vertically. DSP can more accurately model certain problems such as electricity allocation whilst also allowing for better optimal solutions than SP.

In some applications, it may be the case that a specific job could be scheduled in multiple ways. We say that a job is moldable if there are multiple options for its processing time and energy demand. The goal of the problem now becomes to find both the optimal way of scheduling the jobs but also which processing configuration each job should use.

Both SP and DSP are NP-hard problems and thus, obtaining optimal solutions to instances of these problems is not computationally feasible. Instead, research is focused on developing approximation algorithms which can find a solution that is "good enough" whilst running in polynomial time.

In this thesis, we present an approximation algorithm for DSP with moldable jobs. The algorithm runs in polynomial time with respect to the number of jobs and returns a packing whose peak demand is at most $(3/2 + \varepsilon)$ times the optimal peak demand for some $\varepsilon > 0$. This result is very close to the best possible approximation ratio, proven to be 3/2 for DSP. The algorithm as constructed, as well as the approximation ratio, is dependent on an assumption that will be a focus of further study. Key to the design of this algorithm is our approach of transforming an optimal packing in a particular way such that a strong structural result holds. The algorithm then makes heavy use of guessing and linear programming to find a feasible solution within the approximation ratio.

Keywords: Optimization, scheduling, strip packing, linear programming, moldability, NP-hard.

Acknowledgements

I would like to thank both Malin Rau and Albert Vesterlund for their support during the work on this thesis. The many discussions we have had during these past months have helped improve my research capabilities and helped prepare me for future work in this field.

To Malin in particular, I am immensely thankful for the mentorship role you have taken on during this past year, and for introducing me to this field of mathematics. Neither this thesis nor the groundwork laid during the project course before would have been possible without your expertise and help.

I would also like to thank my opponent, Marina, for her thoughtful feedback and interesting perspectives.

And of course, thanks to all my friends and family for their support and company during these past years!

William Bergquist, Kode, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

DSP	Demand Strip Packing
JALP	Job Assignment Linear Program
LP	Linear Program
NFDH	Next Fit Decreasing Height
NP	Nondeterministic Polynomial
P	Polynomial
SP	Strip Packing

Nomenclature

Below is a summary of the variables and parameters used across this thesis. Here, we only cover the variables that serve a major importance across the whole thesis. There are many more variables that are only used locally, e.g. within a specific equation or proof. Such variables are not covered in this section.

Problem fundamentals

$\mathcal{J}$	Set of jobs
j	A job, i.e. an element in $\mathcal{J}$
n	Number of jobs, i.e. $n = \lvert\mathcal{J}\rvert$
w, h	Width and height, although the exact definition varies depending on the context
C_j	Set of allowed processing configurations for job j
(w_{ij}, h_{ij})	A processing configuration denoting a possible shape/configuration that a job can be assigned, i.e. element i from the set C_j
q	Maximum number of configurations per job, i.e. $\lvert C_j\rvert \leq q \; \forall j \in \mathcal{J}$
σ	The packing, a function which determines the placement of jobs by assigning each job a starting position and a processing configuration
D	Deadline, before which all jobs must be placed
T	Maximum height / peak demand of a packing
OPT	The maximum height of the optimal packing

Algorithm

α	Approximation ratio
ε	Error term for the height needed to obtain a packing. We sometimes use scaled versions of this parameter such as ε' or ε^* in different parts of the thesis.
λ	Parameter dependent on ε which appears in the context of λ-forgiving packings
δ, μ	Parameters dependent on ε that are used to partition $\mathcal{J}$ into subsets. Used across the thesis but the definitions change. In chapter 3, the values are not fixed but the bound $\delta, \mu \geq \varepsilon^{\mathcal{O}(1/\lambda)}$ holds. In chapter 4, we set $\delta = \frac{\varepsilon}{1+\varepsilon}$, $\mu = \frac{\varepsilon'^3}{\lceil \log(1/\delta) \rceil}$.
β	A bin, which jobs are assigned to when using the JALP
Ψ	Residual item container where some jobs are placed in the λ-forgiving algorithm

Contents

1

Introduction

Scheduling and packing problems are a subset of problems within the field of mathematical optimization. In these problems, the general goal is to find the optimal way to pack some collection of jobs or items whilst satisfying relevant constraints. Depending on the specific problem being analyzed, both the objective and constraint may be different.

In this thesis, we analyze a specific problem called the Demand Strip Packing problem, abbreviated as DSP [Ebe+25; Gál+21]. To formulate this problem, we introduce the following notation. Let $\mathcal{J}$ denote the set of jobs and let $D \in \mathbb{N}$ denote a deadline. Each job $j \in \mathcal{J}$ has a processing time w_j corresponding to its width and a energy demand h_j corresponding to its height. The area of the job can be seen as the total energy needed to process the job. The goal of the problem is find the most efficient way possible to pack the jobs such that we can minimize the maximal height needed to fit all jobs. The maximum height, also referred to as the peak demand, corresponds to the amount of infrastructure or capacity that is needed to cover for the peak demand.

The way the jobs are placed into the schedule is represented by a function $\sigma : \mathcal{J} \to [0, D]$ which assigns each job a starting time. The function σ is known as a packing of the jobs $\mathcal{J}$. The goal in DSP is to find a feasible packing σ which minimizes the peak demand, the maximal height of stacked jobs somewhere in $[0, D]$. In this thesis, we let T the maximal height of the packing. Both DSP and several closely related problems have been extensively studied and have a wide range of applications. For DSP, power grids and electricity allocation is one clear application whose relevance continues to increase as the power grid transitions to a larger dependence on renewable energy and demand side management [Ala+13; Neb24].

It is proven that the DSP problem is an NP-hard problem [Ebe+25]. For both DSP and other closely related problems, research has thus been focused on developing approximation algorithms [WS11; Ebe+25; JMR19; Gál+21]. The approximation algorithms yield a solution that is "good enough", where the height of its optimal solution deviates from the true optimal height by some multiplicative factor α called an approximation ratio. Crucially, an approximation algorithm must run in polynomial time with respect to the number of jobs, meaning that the approximate solution

can be obtained within a reasonable time. As of January 2026, the most sophisticated approximation algorithm for DSP from [Ebe+25] has an approximation ratio of $3/2 + \varepsilon$. A polynomial time algorithm with an approximation ratio smaller than $3/2$ cannot exist, unless P = NP [Yaw+14].

DSP can be useful for modeling certain problems but in some contexts, the problem definition may be unnecessarily strict. We propose a modified problem were allow for the shapes of the jobs to potentially be altered, a property we call *moldability*. For each job $j \in \mathcal{J}$, we allow multiple possible pairs of (w_j, h_j) rather than each job having a fixed width and height. The focus of this thesis is on the development on an approximation algorithm for this modified version of the DSP problem.

1.1 Our result

We formulate an approximation algorithm for the Demand Strip Packing problem with moldable jobs. The algorithm consists of two subalgorithms which together yield a complete algorithm. The first algorithm has an approximation ratio of $3/2+\varepsilon$ and is fully complete. The second algorithm relies on an assumption regarding the structure of the packing. Whilst we believe the assumption to be true, the approximation ratio may need to be slightly increased in order to achieve it. For completeness sake, we state the assumption here without further context.

> **Assumption 1.1**
>
> For an optimal packing, the height profile of tall jobs can be shifted such that it is δ-translated.

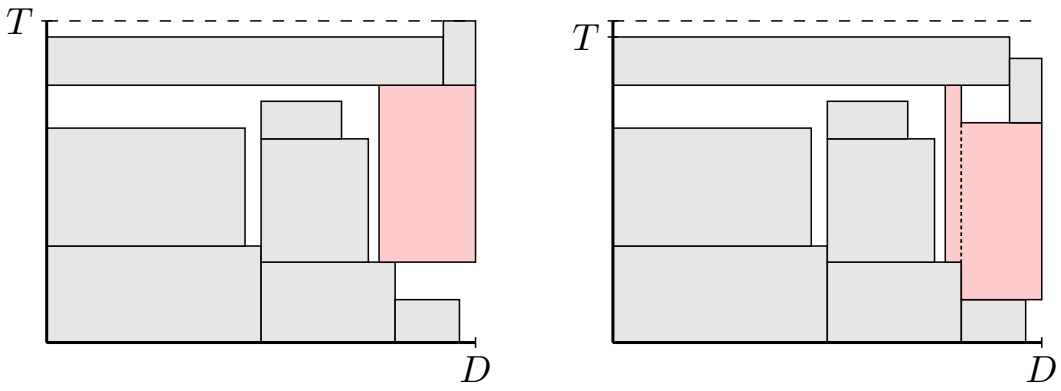
The motivation for why this assumption is needed is covered later in chapter 4. Our main result is captured by the following theorem, which we will spend the rest of this thesis proving.

> **Theorem 1.2**
>
> If assumption 1.1 holds, there exists an approximation algorithm for the Demand Strip Packing problem with moldable jobs.

1.2 Related problems

There are several closely related problems that have been extensively studied. One example is the Strip Packing problem (SP), sometimes referred to as Geometric Strip Packing [Gál+21]. There is one fundamental difference between the problems concerning the slicing and fractional placement of jobs. Strip Packing is perhaps the more intuitive problem, requiring all jobs to be placed into the schedule as contiguous rectangles without any slicing. In DSP, we allow for more flexible solutions by allowing jobs to be sliced vertically. The permitted vertical slicing means that

different segments of a job may be placed at different heights in the schedule. It is however not allowed to move the segments apart in time, or to perform horizontal slicing. A comparison of the two problems is shown in figure 1.1



(a) Strip Packing (SP) **(b)** Demand Strip Packing (DSP)

Figure 1.1: Comparison between Strip Packing (SP) and Demand Strip Packing (DSP). By allowing a job to be sliced vertically, here highlighted in red, a lower optimal value T can be attained.

As DSP is a relaxation of SP, a solution to an instance of SP will always be feasible in the corresponding DSP instance. Note that the converse will not hold in general. As of January 2026, the best approximation algorithm for Strip Packing has an approximation ratio of $5/3 + \varepsilon$ [Har+14]. Strip packing also has many applications and finding better approximation algorithms for it would be an interesting topic. However, due to the more restrictive packing rules, many of the techniques we use will not be directly applicable to this problem and analyzing strip packing is therefore beyond the scope of this project. Naturally, it is of course possible that some work done here could be of use for future work in strip packing as well.

Another family of related problems concerns machine scheduling under various constraints [JMR19; JR21]. Rather than placing the jobs directly into the schedule, each job needs to be processed by a machine. The goal is to minimize the makespan, the smallest deadline D to obtain a feasible packing, while guaranteeing that at most m machines are in use at any one time. This problem, called the Parallel Task Scheduling problem (PTS) has many similarities to DSP. The fundamental difference is that in task scheduling, we have a strict bound on the maximum height or number of machines but allow for a flexible deadline, whereas the contrary is true for DSP. This difference means that algorithms for one problem cannot be directly used for the other, although some methods and techniques may be transferable.

For PTS, the best current result is a $(3/2 + \varepsilon)$-approximation algorithm [Jan12]. While it is an interesting and relevant problem, we will not focus on it during this thesis. Some of our techniques we use may be useful if future work were to introduce

moldability to parallel task scheduling, as these have been used in previous papers in related fields [Jan12; DMT04].

1.3 Outline

In chapter 2, we start by more formally defining our problem. We then cover several common techniques and theory, as well as some other methods more specifically apply for our problem.

Our final algorithm is split into two separate parts, each of whom requires a strong result regarding the way a packing can be restructured. Chapters 3 and 4 concern these two structural results and their proofs. The results proven in these chapters are then used to construct our algorithm in chapter 5.

2

Prerequisites and concepts

In this section, we present some important theory and techniques that are needed, both for proving several structural results and for the construction of our algorithm. Before looking at these techniques, we must define our problem more rigorously.

2.1 Problem formulation

So far, we have described our modified version of DSP in a rather informal way. To proceed, we need a more rigorous definition. As before, let $\mathcal{J}$ denote the set of jobs and let D denote the deadline. To represent the moldability of jobs, multiple approaches can be considered. We select an approach where each job has an associated set of allowed processing configurations. For each job $j \in \mathcal{J}$, we let C_j denote the set of associated processing configurations where each element i in the sets C_j is a pair (w_{ij}, h_{ij}) of possible width and height combinations. We require that for all jobs $j \in \mathcal{J}$, the number of processing configurations is less than some constant $q \in \mathbb{N}$, i.e. $|C_j| \leq q,\ \forall j \in \mathcal{J}$. If one would want the allowed widths and heights to be represented by some function, this approach still works if we use the desired function to generate elements in C_j. Directly taking configurations from a continuous function however is not possible, as we need to have a bounded number of configurations.

Definition 2.1

A *D-feasible packing* is a function $\sigma : \mathcal{J} \to [0, D] \times \mathbb{N}$ that assigns each job $j \in \mathcal{J}$ a start time $t_j \in [0, D]$ and a processing configuration $(w_{ij}, h_{ij}) \in C_j$ such that $t_j + w_{ij} \leq D$.

Definition 2.2

Let t_j^σ denote the start time that job j is assigned by the packing σ and let w_{ij}^σ and h_{ij}^σ denote the particular configuration i for job j used by assignment σ. The set $\mathcal{J}^\sigma(t) = \{j \in \mathcal{J} \mid t_j^\sigma \leq t < t_j^\sigma + p_{ij}^\sigma\}$ denotes the jobs currently packed at time

t. The *peak demand* for the packing is then defined as

$$h(\sigma) = \max_{t \in [0,D]} \sum_{j \in \mathcal{J}^\sigma(t)} h_{ij}^\sigma. \tag{2.1}$$

The goal of DSP is to find a D-feasible packing which minimizes the peak demand $h(\sigma)$. The slicing that is permitted in DSP is captured by this definition, as we do not impose any restrictions on the vertical position of the jobs. Allowing for slicing allows us to interpret DSP as a relaxed version of SP, where by relaxed we mean that the feasible set of solutions to SP is a subset of the set of solutions to DSP. Any solution to SP must therefore be feasible to DSP although the contrary is not necessarily true. By introducing moldability, we allow for more flexible solutions and potentially lower optimal values. Again, moldability relaxes the problem and a feasible solution to the normal DSP problem will also be feasible to DSP with moldable jobs. As previously mentioned, our goal is to develop an approximation algorithm which can obtain a feasible packing that is optimal when considering moldable jobs.

We previously introduced the concept of an approximation algorithm rather informally. To proceed, we need a more rigorous definition.

Definition 2.3

A polynomial-time algorithm which yields a packing σ is an *α-approximation algorithm* if $h(\sigma) \leq \alpha OPT(\mathcal{J})$, where $\text{OPT}(\mathcal{J})$ is the objective value for an optimal packing of $\mathcal{J}$. The parameter α is known as the algorithm's *approximation ratio*.

When designing an algorithm, we need to both verify that it always yields a solution that is within a factor α of the true optimal solution OPT and that the algorithm will run in polynomial time with respect to the number of jobs. To handle the latter, a commonly used approach is to use various techniques that simplify the structure of packing [Ebe+25; JR19; JR21; JL18; KKR25]. For this simplified structure, the number of possible packings is constant with respect to n, i.e. it is bounded by some large constant. An algorithm is then be able to iterate over all possibilities in polynomial time. Our algorithm is based on this approach, with chapters 3 and 4 proving the existence of two such restructured packings which we make use of when constructing our algorithm in chapter 5.

As a problem is relaxed, the set of feasible solutions becomes larger. Hence, an algorithm which finds an optimal solution will often become more complex as more possible solutions need to be considered. When introducing moldability to DSP, the number of possible packings to consider grows exponentially as we need to account for all possible permutations of job configurations. Even if every job has only two possible configurations, there are 2^n configurations of jobs to consider where it is possible that only one yields an optimal solution. This is one reason why previous algorithms for DSP cannot be directly used when introducing moldable jobs. We

instead have to design a fundamentally different algorithm to account for this.

2.2 Packing algorithms

As steps in our algorithm and proofs, we make use of several standard algorithms and results. These algorithms can be seen as a "black box"; it is not of interest to us how they work, only that they offer a guaranteed way to pack a subset of jobs under the right conditions. One of the most fundamental results is the following from Steinberg.

Lemma 2.4: Steinberg's algorithm [Ste97]

There exists an algorithm that packs a set of rectangular objects $\mathcal{R}$ into a rectangular container of height a and width b in polynomial time. This algorithm works if and only if the inequalities

$$h_{\max} \leq a, \quad w_{\max} \leq b,$$
$$2\sum_{r\in\mathcal{R}} w(r)h(r) \leq ab - \max\{(2h_{\max} - a), 0\} \cdot \max\{(2w_{\max} - b), 0\},$$

hold, where $h_{\max}$ and $w_{\max}$ denotes the maximum height/width among the rectangles in $\mathcal{R}$.

In cases where we for of some subset of jobs can bound both the size of the jobs and their combined area, Steinberg's algorithm can be of great use. We especially make use of it in the following special case.

Corollary

If the area of the rectangular container is at least twice the area of all the jobs and the maximum width/height of any of the jobs is at most half the width/height of the container, all jobs can always be placed using Steinbergs algorithm.

We make us of this corollary several times. If we can bound the area of some subset of jobs, we can place them into a container with suitable dimensions using this result. Often, we consider a subset with either very narrow or very short jobs where the conditions easily can be satisfied. Another important tool is the following algorithm.

Lemma 2.5: NFDH [Cof+80]

Next-Fit-Decreasing-Height (NFDH) is an algorithm which places jobs onto a strip of width D by placing jobs onto shelves. Jobs are placed in order of decreasing height. The first job is placed at the left and subsequent jobs are placed directly to the right. When a job does not fit, a new shelf is created above at the height of the tallest job in the shelf below. The procedure is repeated until all

jobs are placed. The algorithm runs in polynomial time.

Like with Steinberg's algorithm, we can use NFDH to place jobs into a container with a specified width. We use it when placing small jobs thanks to the following result.

Lemma 2.6: Lemma 2.15 in [Chr+16]

Let B be a rectangular region with width w and height h. Consider a set of of small rectangles whose width and height both are smaller than ϵ. If we pack these into B using the NFDH algorithm, an area of $w \cdot h - (w + h)\epsilon$ can always be packed and the total wasted volume in B is at most $(w + h)\epsilon$.

2.3 Packing structure

Before we can proceed with constructing our algorithm, we need to assign some structure to the way jobs are packed. To do this, consider the two following definitions.

Definition 2.7

A D-feasible packing σ is said to be λ-*forgiving* if it has height $h(\sigma) \leq 3/2 \cdot \mathrm{OPT}$ and has an empty slot that can accommodate a container of width λD and height h_σ.

Definition 2.8

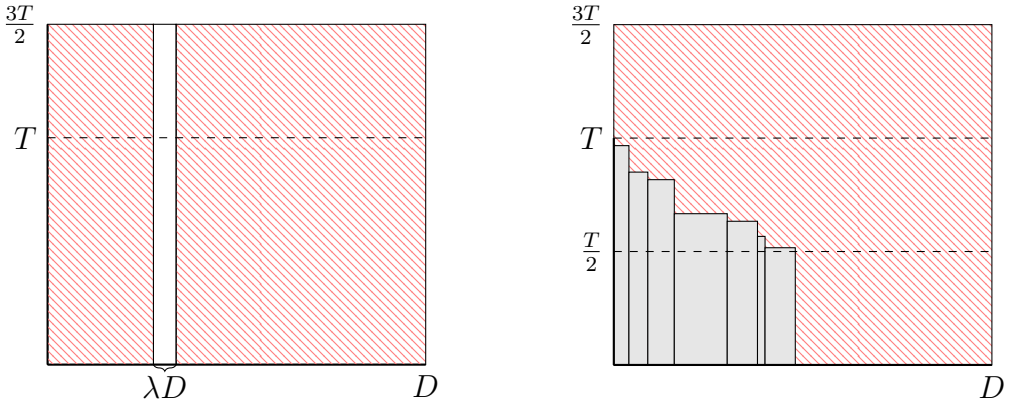
A D-feasible packing σ is said to be (ε, T)-*neat* if it has height $h(\sigma) \leq (3/2 + \varepsilon)T$ and all jobs with height greater than $T/2$ are placed next to each other in non-increasing order starting from the left.

An example comparing the two types of packings is shown in figure 2.1. These two types of packings are of great use for us thanks to the following statement proven by [Ebe+25].

Theorem 2.9: Theorem 1.2 in [Ebe+25]

Consider a set of jobs $\mathcal{J}$ and deadline D and let σ be an optimal packing for $(\mathcal{J}, D)$ with height OPT. For any $\varepsilon > 0$, $\lambda \leq \min\{\frac{\varepsilon}{3(5+4\varepsilon)}, \frac{1}{60}\}$, the optimal packing σ can then be transformed into a new packing σ' that is either λ-forgiving or $(\varepsilon, \mathrm{OPT})$-neat.

This result guarantees that a packing that is optimal to DSP can be always be transformed into a new packing that is either λ-forgiving or (ε, T)-neat. We can therefore construct algorithms that make use of the structure of these packings to obtain a feasible packing. We do not know in advance which of the two types we



(a) A λ-forgiving packing with height $T \leq 3/2 \cdot \mathrm{OPT}$. Somewhere, there exists a gap of width λD and height at least T where we can place residual jobs.

(b) An (ε, T)-neat packing. All jobs with height greater than $T/2$ are placed next to each other in non-increasing order starting from the left.

Figure 2.1: A comparison between λ-forgiving (left) and (ε, T)-neat (right) packings. In the red hatched area, other jobs can be packed with no requirements on their placement.

can transform our packing info, so we simply attempt to create both packings and consider the one which has a lower optimum. Since the packings have a maximal height of $3/2 \cdot \mathrm{OPT}$, these algorithms when combined yield a $(3/2+\varepsilon)$-approximation algorithm. We make use of the same philosophy for our algorithm.

Claim 2.10

The result from theorem 2.9 also holds for DSP with moldable jobs.

Proof. Consider a packing σ which is optimal to DSP with moldable jobs. Each job has then been assigned a specific processing configuration. We can view this as an instance of DSP, with a set of jobs $\mathcal{J}$ containing the jobs with the optimal processing configurations. The packing σ is then clearly optimal for this instance of DSP, and thus theorem 2.9 holds. □

2.4 Rounding, grouping and reduction

A key concept used several times in the thesis is the rounding of jobs. If the height and widths of jobs are used directly, it is likely that we end up with as many unique heights and widths as the number of jobs. It is at least clear that the number of unique heights and widths will depend on n. This is undesirable in many cases, as we would rather have a constant number of unique shapes to work with. To handle this, we can round either the width or height up slightly to one of a constant number of options. In this section, we present and prove several lemmas that allow us to

perform this rounding under certain conditions.

Lemma 2.11

For jobs with a height of at least δT, their heights h_j can be rounded to multiples $i\varepsilon\delta T$, $i \in \mathbb{N}$. The maximum height of the packing will increase by at most εT from this step.

Proof. Consider a specific job $j \in \mathcal{J}$ with a height h_j. Stretch the whole schedule by a factor of $(1+\varepsilon)$ and let $\widehat{h}_j$ denote the job's stretched height, i.e.

$$\widehat{h}_j := (1+\varepsilon)h_j. \tag{2.2}$$

The original height and the stretched height will differ by exactly εh_j. For jobs with a height of at least δT, we therefore know that

$$\widehat{h}_j - h_j = \varepsilon h_j \geq \varepsilon\delta T. \tag{2.3}$$

Since the endpoints of the interval differ by at least $\varepsilon\delta T$, we can be sure that there must exist some $i \in \mathbb{N}$ such that

$$i\varepsilon\delta T \in [h_j, \widehat{e}_j]\,. \tag{2.4}$$

The same principle can be applied to all jobs with a height of at least δT. Since the rounded height is at most $\widehat{h}_j = (1+\varepsilon)h_j$, stretching the whole schedule by $(1+\varepsilon)$ ensures that no jobs may overlap after the rounding step. The rounded job fits entirely within the stretched space of the original job. A $(1+\varepsilon)$ stretch of the packing will thus always allow us to round all jobs to some integer multiple of $\varepsilon\delta T$ whilst maintaining a feasible packing. The stretch corresponds to a new maximum height of $(1+\varepsilon)T$, i.e. increasing the maximum height of the packing by εT. □

Corollary

Any set of jobs rounded using lemma 2.11 has at most $1/(\varepsilon\delta)$ possible unique height levels[1].

Lemma 2.12: Geometric grouping, theorem 2 in [KK82]

For jobs $\mathcal{J}$ with a width of at least δT and a height of at most μT, their widths w_j can be grouped such that there are at most $\mathcal{O}(\log(1/\delta)/\varepsilon')$ unique widths. The maximum height of the packing will increase by at most $4\varepsilon' T$ from this step.

This rounding technique can be applied when rounding jobs that are sufficiently wide. The algorithm was first proposed by [KK82] for use in the Bin Packing

[1] With a more sophisticated rounding method using grouping, it is possible to reduce the possible height levels to $\mathcal{O}(1/\varepsilon^2 \cdot \log(1/\delta))$ [Dep+23]. For our application, $\mathcal{O}(1/(\varepsilon\delta))$ is sufficient.

Problem. It was later adapted for use in machine scheduling and strip packing problems [Rau20; Svi12], making it useful for our applications for both λ-forgiving and (ε, T)-neat packings.

Proof. To start, we partition the jobs $j \in \mathcal{J}$ logarithmically into the sets $\mathcal{J}^k$ such that $\mathcal{J}^k = \{j \in \mathcal{J} \mid \frac{D}{2^k} < w_j \leq \frac{D}{2^{k-1}}\}$. The following steps are done separately for each $k = 1, 2, \ldots, \mathcal{O}(1/\delta)$.

We start by ordering the jobs $j \in \mathcal{J}^k$ such that they are placed on top of each other in order of non-increasing width. Let H^k denote the total height of the stack of all jobs in $\mathcal{J}^k$. For each height $\ell \in \{0, \ldots, 1/\varepsilon'\}$, we introduce an auxiliary job j_ℓ of height $\varepsilon' H^k$ and with the same width as the job in the stack which intersects a horizontal line at height $\ell\varepsilon' H^k$. If the line intersects the border between two jobs, we assign the width of the upper job to j_ℓ. The index of the original job which was intersected by the line is denoted by i_ℓ. Let w_ℓ denote the width assigned to the auxiliary job, let $\mathcal{W}$ denote the set of all widths and let $\mathcal{J}^{\mathrm{R}}$ denote the set of auxiliary jobs.

Now, we introduce two new jobs j_ℓ^1 and j_ℓ^2 that correspond to slicing j_ℓ horizontally. The sliced jobs have width $w\ell$ and heights

$$h_{j_\ell^1} = \ell\varepsilon' H^k - \sum_{i=1}^{i_\ell - 1} h_i, \tag{2.5}$$

$$h_{j_\ell^2} = h_{j_{i_\ell}} - h_{j_\ell^1}. \tag{2.6}$$

See figure 2.2 which shown an example of the job slicing. With the jobs sliced, we can further partition $\mathcal{J}^k$ further into one set for each ℓ. We let $\mathcal{J}_\ell^k$ include the jobs with a total height of $\varepsilon' H^k$ whose width is between w_ℓ and $w_{\ell+1}$, i.e.

$$\mathcal{J}^{k,\ell} = \left\{j_\ell^2, j_{i_\ell+1}, j_{i_\ell}, \ldots, j_{i_{\ell-1}+1}\right\} \tag{2.7}$$

For all jobs in $\mathcal{J}^{k,\ell}$, we can now round their widths up to the width w_ℓ of the auxiliary item. We can therefore obtain a set of rounded jobs and their associated widths by considering all $\ell \in \{0, \ldots, 1/\varepsilon'\}$ and $k \in \{1, \ldots, \log(1/\delta)\}$. Letting $\mathcal{J}_R$ denote the set of rounded jobs, we obtain

$$\mathcal{J}_R^k = \cup_{\ell=0}^{1/\varepsilon'-1} j_\ell \tag{2.8}$$

$$\mathcal{W}_k = \cup_{\ell=0}^{1/\varepsilon'-1} w_\ell \tag{2.9}$$

$$\mathcal{J}_R = \cup_{k=1}^{1/\delta} \mathcal{J}_R^k \tag{2.10}$$

$$\mathcal{W} = \cup_{k=1}^{1/\delta} \mathcal{W}_k \tag{2.11}$$

Of course, some jobs may no longer fit integrally when the jobs have been rounded. We want to show that by increasing the packing height by at most $4\varepsilon' T$, one can always obtain a feasible fractional packing. We represent fractional packings using fractional packing vectors.

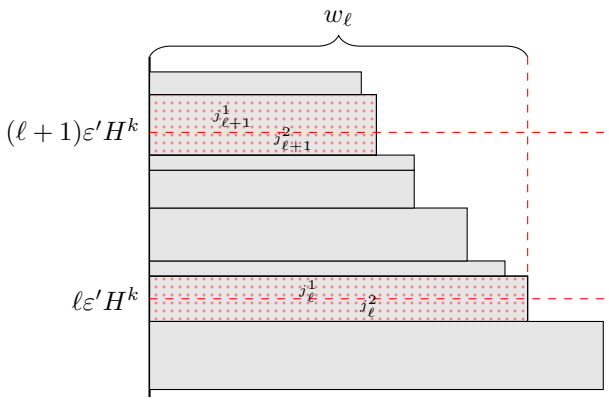


Figure 2.2: Slicing of the jobs which intersect the lines at integer multiples of $\varepsilon' H^k$.

Definition 2.13

A *fractional packing vector (FPV)* for a fractional packing ς is a vector of the form

$$(s, x, j) \in [0, D] \times (0, 1] \times \mathcal{J}, \tag{2.12}$$

denoting the height x of job j placed at starting position s. Each fractional packing ς can be represented by its associated set $\mathcal{F}^\varsigma$ of fractional packing vectors.

A fractional packing is feasible if $s \leq D - w_j$ and $\sum_{(s,x,j)\in\mathcal{F}^\varsigma} x = 1 \; \forall j \in \mathcal{J}$. To construct a feasible fractional packing ς, we consider one set $\mathcal{J}^{k,\ell}$ at a time. For each job $j \in \mathcal{J}^{k,\ell}$, we start a fraction $x = \frac{h_j}{h_{j_{\ell+1}}}$. For each set, we remove the widest item j_0.

To show that the fractional packing is feasible, we note that $w_{\ell+1} \leq w_j \forall j \in \mathcal{J}^{k,\ell}$. All jobs therefore clearly fit inside the schedule and satisfy $D - w_{j_{\ell+1}} \geq s$. For each fractionally placed job, we can see that

$$\sum_{(s,x,j_{\ell+1})\in\mathcal{F}^\varsigma} x = \sum_{j\in\mathcal{J}^{k,\ell}} \frac{h_j}{h_{j_{\ell+1}}} = 1. \tag{2.13}$$

It is thus clear that the packing where j_0 is removed from each set is fractionally feasible. All that remains is to place the removed jobs. Since each removed job has a width bounded by $D/2^{k-1}$ for its corresponding set and each removed job has a height of $\varepsilon' H^k$, we can bound the removed area by

$$\sum_{k=1}^{\log(1/\delta)} \varepsilon' H^k \frac{D}{2^{k-1}} = 2\varepsilon' \sum_{k=1}^{\log(1/\delta)} \frac{DH^k}{2^k} \leq 2\varepsilon' \sum_{k=1}^{\log(1/\delta)} \text{area}(\mathcal{J}^k) \leq 2\varepsilon' DT. \tag{2.14}$$

Since the jobs are flat and have an area bounded by $2\varepsilon' DT$, we can place them on top of the schedule inside a box of width D and height $4\varepsilon' T$ using Steinberg's algorithm.

□

Lemma 2.14

For a set of jobs with width greater than δD and height less than μT, we can reduce the number of starting points used such that there exists a set of starting positions $\mathcal{S}$ where $|\mathcal{S}| \leq \mathcal{O}(1/(\varepsilon\delta))$. Equivalently, if the set is partitioned logarithmically, each set $\mathcal{J}^\ell$ has $2^\ell/\varepsilon$ possible starting positions. Additionally, the total height of jobs stacked at any one time is a multiple of μT. This reduction increases the maximum height of the packing by at most $2\varepsilon T$, in addition to removing at most $\log(1/\delta) \cdot \frac{2\mu}{\varepsilon^2} DT$ of area which can be packed using Steinberg's algorithm.

Proof. To do this, we start by logarithmically partitioning our horizontal jobs into the sets

$$\mathcal{J}^\ell_{\text{hor}} := \left\{ j \in \mathcal{J}_{\text{hor}} \mid D/2^\ell < w'_j \leq D/2^{\ell-1} \right\},$$

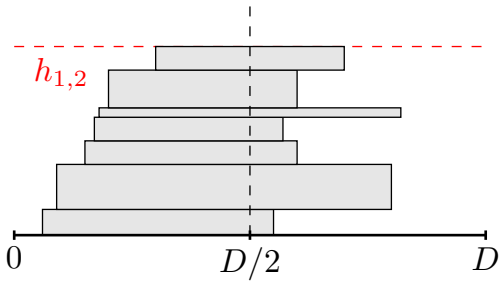
yielding at most $\lceil \log(1/\delta) \rceil$ different sets. Consider a specific set $\mathcal{J}^\ell_{\text{hor}}$. Dividing the schedule into 2^ℓ segments, each of width $D/2^\ell$, ensures that each job $j \in \mathcal{J}^\ell_{\text{hor}}$ must end and start in different segments. For each segment i, we consider the jobs $j \in \mathcal{J}^\ell_{\text{hor}}$ that end in the segment and stack them such with their order being determined by their increasing starting position (the job with the earliest starting position at the bottom). We let $h_{\ell,i}$ denote the height of these stacked jobs. Note that these stacked jobs are not the same as the *stack* we defined previously.

Now, we divide the stacked jobs into $1/\varepsilon$ layers of height $\varepsilon h_{\ell,i}$. Any jobs that intersect the boundary between layers will be sliced along their long axis and placed fractionally instead. To start our repacking, we first remove all the jobs (including fractional parts) that are located in the bottom layer. For each subsequent layer above, we shift all (fractional) jobs to the left such that their new starting position is the same as the earliest original starting position in the layer below. For each segment, each one of the $1/\varepsilon$ layers corresponds to one new starting position. An example of the procedure is shown in figure 2.3. Doing the same procedure for each of the 2^ℓ segments means that we can reduce the number of starting points for the set $\mathcal{J}^\ell_{\text{hor}}$ to $2^\ell/\varepsilon$. Applying the procedure for all widths ℓ yields to the total bound on the number of starting positions as

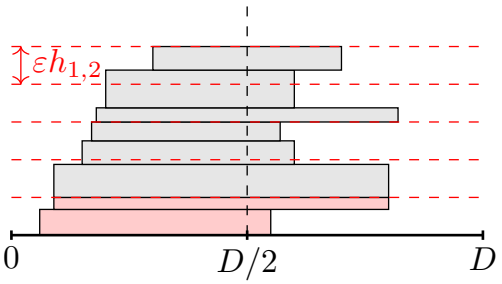
$$\sum_{\ell=1}^{\lceil \log 1/\delta \rceil} \frac{2^\ell}{\varepsilon} = \frac{2^{\lceil \log 1/\delta \rceil + 1} - 2}{\varepsilon} \in \mathcal{O}\left(\frac{1}{\varepsilon\delta}\right). \tag{2.15}$$

The problem that remains is the jobs we removed to achieve the described repacking. For each segment i, we removed one layer that is $\varepsilon h_{\ell,i}$ tall. Considering the height across all segments, we can bound the total height of the removed jobs by $\varepsilon h(\mathcal{J}^\ell_{\text{hor}})$. We now place these removed jobs on top of the schedule. As their width is bounded by $D/2^{\ell-1}$, we are able to place $2^{\ell-1}$ jobs next to each other further reducing the total height we need to add on top to $\varepsilon h(\mathcal{J}^\ell_{\text{hor}})/2^{\ell-1}$.

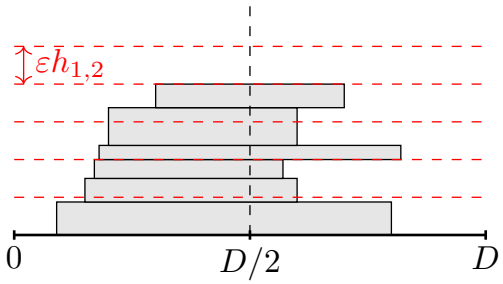
Since the total area of the packed horizontal jobs must be less than DT and the



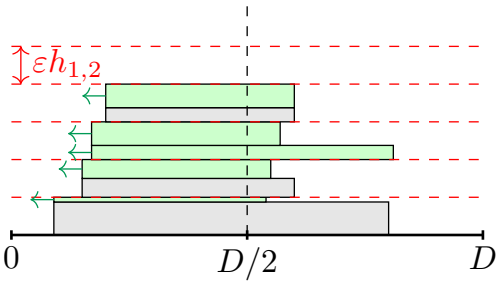
(a) The 7 jobs ending in the second segment are placed on top of each other sorted by their starting positions. The height of these jobs is denoted $h_{1,2}$.



(b) The schedule is split into layers of height $\varepsilon h_{1,2}$ and all fractional jobs in the bottom layer are removed (red in the figure).



(c) The remaining jobs are moved one layer down



(d) Starting from the top, we slide each job to the left until the latest original starting position present in the layer below. We end up with 4 starting positions instead of 7, and with some removed jobs.

Figure 2.3: An example of the reduction procedure for the set $\mathcal{J}^1_{\text{hor}}$, i.e. jobs with a width $D/2 < w \leq D$.

width of each job in $\mathcal{J}^\ell_{\text{hor}}$ is at least 2^ℓ, we can obtain the bound

$$\sum_{\ell=1}^{\lceil \log 1/\delta \rceil} \frac{D}{2^\ell} h(\mathcal{J}^\ell_{\text{hor}}) \leq DT \iff \sum_{\ell=1}^{\lceil \log 1/\delta \rceil} \frac{\varepsilon}{2^{\ell-1}} h(\mathcal{J}^\ell_{\text{hor}}) \leq 2\varepsilon T. \tag{2.16}$$

Hence, the reduction in used starting positions for horizontal jobs will add at most $2\varepsilon T$ to the height of the packing.

In order to ensure that the height profile of the horizontal jobs always is a multiple of μT, we must further remove some jobs. For each group $\mathcal{J}^\ell_{\text{hor}}$, consider the jobs stacked at each of the $2^\ell/k$ starting positions. We slice of a small amount of height to achieve a height that is a multiple of μT. Since the jobs in $\mathcal{J}^\ell_{\text{hor}}$ have a width of at most $D/2^{\ell-1}$, we must remove at most

$$\log(1/\delta) \cdot \frac{2^\ell}{\varepsilon} \cdot \frac{1}{\varepsilon} \cdot \frac{D}{2^{\ell-1}} \cdot \mu T = \log(1/\delta) \cdot \frac{2\mu}{\varepsilon^2} DT \tag{2.17}$$

area of jobs. With an appropriate choice of δ and μ, these can fit inside a container placed on top of the schedule.

□

3

λ-forgiving packings

This section concerns the case where it is possible to transform a packing σ into a λ-forgiving packing. We state and prove a structural result showing that by allowing for a slightly increased peak demand, we can transform any packing into another packing with much more structure. In addition to the transformed packing, we obtain a container of residual items with width λD. Since we are working with packings that can be transformed into λ-forgiving packings, we can place this residual item container in the λD wide slot left open.

Disclaimer

The contents of this section were originally produced for a separate project as a part of another course. The report[1] from said course is not published and thus, we include the contents of the report in their entirety here instead.

In order to define our desired structure, we must first partition the jobs $\mathcal{J}$ into 5 separate categories depending on their shapes and sizes. As our repacking procedure can result in a taller height than the optimum T, we instead consider the $T' := (1 + \mathcal{O}(\varepsilon))T$ as the upper limit for the height of the restructured packing. For an optimal packing σ, we consider the width w_{ij}^{σ} and height h_{ij}^{σ} for the particular configuration C_j used by σ. Even if a specific job has many possible configurations, we categorize the jobs based on the configuration used in the packing σ. This allows us to largely ignore the concept of moldability for the structural result. For the remainder of this section, let w_j denote w_{ij}^{σ} and let h_j denote h_{ij}^{σ}. For two values δ and μ, $\mu < \delta$, we define the following 5 categories.

- $\mathcal{J}_{\text{large}} := \{j \in \mathcal{J} | h_j \geq \delta T', w_j > \delta D\}$
- $\mathcal{J}_{\text{hor}} := \{j \in \mathcal{J} | h_j < \mu T', w_j > \delta D\}$
- $\mathcal{J}_{\text{ver}} := \{j \in \mathcal{J} | h_j \geq \delta T', w_j < \mu D\}$
- $\mathcal{J}_{\text{small}} := \{j \in \mathcal{J} | h_j < \mu T', w_j < \mu D\}$

[1] *A new structural result for the Demand Strip Packing problem with moldable jobs*, [BR26]

- $\mathcal{J}_{\text{medium}} := \mathcal{J} \setminus (\mathcal{J}_{\text{large}} \cup \mathcal{J}_{\text{hor}} \cup \mathcal{J}_{\text{ver}} \cup \mathcal{J}_{\text{small}})$

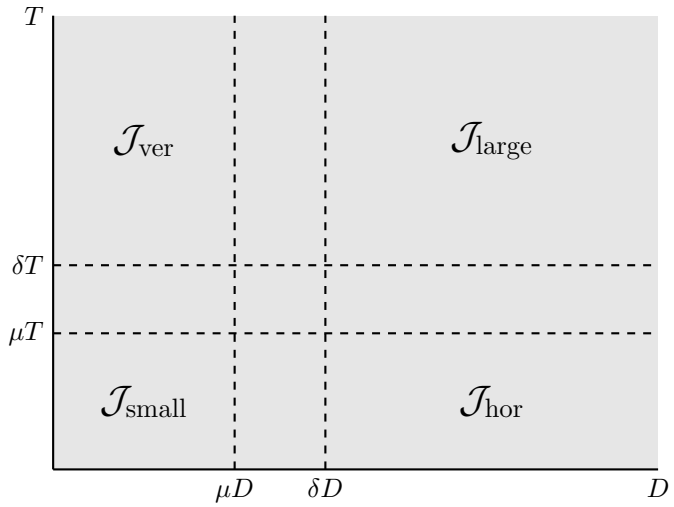


Figure 3.1: The partitioning scheme for jobs. If a job j starts in the bottom left corner, it will belong to whichever set its top right corner is located in according to this partitioning. If the top right corner does not appear in any of the 4 regions, it is instead a medium job.

A visualization of the partitioning is shown in figure 3.1. We will now introduce several definitions that will later be used both for proving the existence of a solution and in the repacking algorithm.

Definition 3.1

Consider a packing σ of jobs $j \in \mathcal{J}$ with assigned heights h_j. A *band* is an inclusion maximal set of jobs of the same height that are placed next to each other under σ.

Definition 3.2

Consider a packing σ of jobs $j \in \mathcal{J}$ with assigned widths w_j. A *stack* is an inclusion maximal set of jobs of the same width that are placed on top of each other under σ.

Definition 3.3

Consider a packing σ of jobs $j \in \mathcal{J}$. We define the *profile* as the function $P_{\sigma,\mathcal{J}}(t) : [0, D] \to \mathbb{R}$ such that $P_{\sigma,\mathcal{J}}(t) := \sum_{j \in \mathcal{J}^\sigma(t)} h_{ij}$. On a given interval $[a, b]$, we say that a profile varies more than y if $\max_{a \le x \le b} P_{\sigma,\mathcal{J}}(x) - \min_{a \le x \le b} P_{\sigma,\mathcal{J}}(x) > y$.

In most cases, these definitions will be applied to a specific subset of the jobs $\mathcal{J}$. Bands are used for vertical jobs, stacks are used for horizontal jobs and we often consider the profile of several different subsets depending on the context. The struc-

tural result we prove is stated in theorem 3.5, based on the following definition.

Definition 3.4

Consider a packing σ of jobs $j \in \mathcal{J}$. We say that the packing σ is *compliant* if

1. There are no medium jobs
2. All vertical jobs are placed into bands and the number of bands, the number of possible starting positions of bands and the number of possible widths and height of bands is in $\mathcal{O}_n(1)$
3. All horizontal jobs are placed into stacks and the number of stacks, the number of possible starting positions of stacks and the number of possible widths and height of stacks is in $\mathcal{O}_n(1)$
4. Horizontal and large jobs have starting points $s \in \mathcal{S}, |\mathcal{S}| \leq \mathcal{O}_n(1)$
5. Small jobs are placed either into the gaps left over inside the schedule or on top whilst adding at most εT to the optimal height.

Using definition 3.4, we state the following theorem.

Theorem 3.5

Let $\varepsilon > 0$, $\lambda \in \mathcal{O}(\varepsilon)$ and let $\mathcal{S}$ denote the set of starting points for horizontal and large jobs. Given an optimal packing σ which admits a λ-forgiving packing, there exists a restructuring of its jobs such that almost all jobs can be placed into a new packing σ' which is compliant. The few remaining jobs can be placed on top of the schedule or into a container Ψ of height T and width λD. The peak demand of this restructured packing together with Ψ is at most $(1 + \mathcal{O}(\varepsilon))T$.

The rest of this chapter is spent on proving theorem 3.5. To prove the theorem, we will verify that we can obtain a compliant packing by considering each type of job separately. As a complaint packing cannot contain any medium jobs, we know that these jobs must be placed inside the residual job container Ψ. To make sure that they will fit, we start with a lemma that allows us to bound the amount of area/work that will be categorized in $\mathcal{J}_{\text{medium}}$.

Lemma 3.6

For a packing σ and any $c, \varepsilon > 0$, it is possible to pick constants $\mu, \delta \geq \varepsilon^{\mathcal{O}(1/\lambda)}$ as subsequent pairs from a sequence $\{\rho_i\}_{i=0}^{\lceil 16/\lambda \rceil}$ such that $\text{area}(\mathcal{J}_{\text{medium}}) \leq \lambda DT/8$ and $\mu \leq c\varepsilon^5\delta$.

Proof. Consider a sequence $\rho_0 := \varepsilon^5/4,\ \rho_{i+1} := c\rho_i\varepsilon^5$. Take an index $i \in \{0, \dots, \lceil 16/\lambda \rceil\}$ and let $\delta = \rho_i$ and $\mu = \rho_{i+1}$. The smallest possible choice for μ and δ occurs for $i = \lceil 16/\lambda \rceil$ with $\rho_{\lceil 16/\lambda \rceil} = \varepsilon^5/4 \cdot (c\varepsilon^5)^{\lceil 16/\lambda \rceil} \geq \varepsilon^{\mathcal{O}(1/\lambda)}$. It is therefore clear δ and μ that both must be greater than $\varepsilon^{\mathcal{O}(1/\lambda)}$ and that $\mu \leq c\varepsilon^5\delta$ as $\mu = \rho_{i+1} = c\rho_i\varepsilon^5 = c\delta\varepsilon^5$. We want to show that for at least one of these i, the area of the medium jobs we get from using the values of δ and μ associated with that i is less than $\lambda DT/8$. Let $\mathcal{J}^i_{\text{medium}}$ denote all jobs classified as medium if we set $\delta = \rho_i$ and $\mu = \rho_{i+1}$. For a job of a specific size, there are only 2 possible values of i that could ever result in the job being classified as medium. Hence, counting the area of all jobs classified as medium would result in at most twice the area of all jobs. Since we know that the area of all jobs is bounded by DT, we can ensure that

$$\sum_{i=0}^{\lceil 16/\lambda \rceil} \text{area}(\mathcal{J}^i_{\text{medium}}) \leq 2DT. \tag{3.1}$$

This maximum area of $2DT$ must be dispersed between the $16/\lambda$ possibilities. By the pigeonhole principle, it is clear that at least one of the choices of i must have a job area less than

$$\frac{2DT}{16/\lambda} = \lambda DT/8, \tag{3.2}$$

or else every i would have an above average area. This shows that there will always exist a choice of i, and by extension δ and μ such that we can obtain this small bound on the amount of medium jobs. □

3.1 Vertical jobs

We will begin the proof of theorem 3.5 by analyzing the placement of vertical jobs. The statement in the theorem can then be equivalently stated as lemma 3.7 below.

Lemma 3.7

Let σ be a packing with height T. By removing a small amount of vertical jobs, the packing can be transformed to a new packing σ' such that all remaining vertical jobs are placed into bands where the number of bands is bounded by $\mathcal{O}(1/(\varepsilon\delta^2))$. The removed jobs fit inside one half of the residual item container Ψ.

To prove this lemma, we will reuse many methods from section 5.2 and 5.3 in [Dep+23]. In this paper, these methods were used to create an algorithm for placing the vertical jobs. We instead use largely the same methods to prove a structural result. In this paper, vertical jobs are placed into the schedule using a configuration LP, a special type of linear program. Whilst the structure of the packing attained is not discussed in detail, the configuration LP technique actually placed the jobs into bands. Hence to show the desired band placement, we will reuse the steps made in this paper and formalize them.

We start by rounding all vertical jobs using lemma 2.11. Since they have a height

of at least δT, we can round the heights of all vertical jobs to values $i\varepsilon\delta T$. From corollary 2.4, we get at most $\mathcal{O}(1/(\varepsilon\delta))$ possible heights. Next we partition the schedule into $1/\gamma$ segments of width γD for some $\gamma \in \mathcal{O}_\varepsilon(1)$. Inside each γD wide segment, we consider the height profile of all large and horizontal jobs present. If the height profile jumps inside the segment such that the difference between the maximum and minimum height is at least εT, we fractionally remove any vertical and small jobs inside the segment. The two following claims concern these removed vertical jobs.

Claim 3.8

The area of the removed fractional jobs is bounded by $\mathcal{O}(1/(\varepsilon\delta))\gamma DT$

Proof. Jobs are only removed when the height profile changes, which can only occur when a horizontal or large job either starts or ends. Since large and horizontal jobs have a width of at least δD and their total area must be bounded by DT, we know that their total height is at most $DT/(\delta D) = T/\delta$. As the height difference needs to be at least εT for the jobs to be removed, the number of segments where jobs are removed is bounded by

$$2\frac{T/\delta}{\varepsilon T} = \mathcal{O}(1/(\varepsilon\delta)), \tag{3.3}$$

where the factor of 2 comes from considering both start and end points. As the maximum area of a segment is γDT, we can bound the area of the removed jobs by

$$\mathcal{O}(1/(\varepsilon\delta))\gamma DT. \tag{3.4}$$

□

Claim 3.9

It is possible to choose a value of $\gamma \in \mathcal{O}(\varepsilon\delta\lambda)$ such that the fractionally removed vertical jobs can be placed inside the first vertical quarter of the residual job container Ψ, i.e. a container with width $w(\Psi)/4 = \lambda D/4$ and height $h(\Psi) = T$.

Proof. To start, consider all removed vertical jobs with height $h_j > T/2$ and place them next to each other. Using the bound on the area of removed jobs from claim 3.8, we can bound the width of these jobs by $2\cdot\mathcal{O}(1/(\varepsilon\delta))\gamma DT/(T/2) = 4\cdot\mathcal{O}(1/(\varepsilon\delta))\gamma D$. Next, we consider the remaining removed jobs, i.e. the ones with height $h_j \leq T/2$. To place these jobs, we first slice them into slices with a width of 1 and stack several of them on top of each other until the height $T/2$ is attained. It will always be possible to fractionally place these slices on top of each other without exceeding height T. Placing this slice of width 1 next to the other jobs, we can bound the total width by

$$4 \cdot \mathcal{O}\left(\frac{1}{\varepsilon\delta}\right)\gamma D + 1 \leq \mathcal{O}\left(\frac{1}{\varepsilon\delta}\right)\gamma D. \tag{3.5}$$

From this bound on the width it is clear that with a choice of $\gamma \in \mathcal{O}(\varepsilon\delta\lambda)$, we obtain a maximal width of

$$\mathcal{O}\left(\frac{1}{\varepsilon\delta}\right) D \cdot \mathcal{O}(\varepsilon\delta\lambda) = \mathcal{O}(\lambda)D. \tag{3.6}$$

With an appropriate choice of constant, it is clear that the removed vertical jobs will have a maximal width of $\lambda D/4$ and thus will fit inside of the residual job container Ψ. □

The jobs that are removed cannot be placed into the schedule. Instead, we make use of the residual job container Ψ to place them. With the above claims, we have shown that with an appropriate choice of γ, the jobs we need to remove are few enough that they will fit inside this container. The placement of all removed jobs is covered in more detail later in section 3.4. For now, we will proceed with the remaining vertical jobs. Inside each segment, we can make a guess for the height reserved for the small and vertical jobs. There will be $\mathcal{O}(1/\varepsilon^{1/\gamma})$ possible guesses in total thanks to the rounding step previously, whilst adding at most $2\varepsilon T$ to the max height. We must also allow for an additional extra height of εT to account for the jump in the horizontal and large jobs, which is at most εT. As mentioned previously, the aim is now to use a configuration LP to place the jobs. First, we must define what a configuration is.

Definition 3.10

A *configuration* C is a multiset of jobs and their associated heights of the form

$$C = \{a_\eta : \eta \mid \eta \in \{h_j \mid j \in \mathcal{J}\}\},$$

where jobs η are placed with multiplicity a_η.

Note that the term *configuration* here is not related to the *processing configurations* that represent the moldable shapes of the jobs. The configuration LP is a linear program which places the configurations into the different segments. It has decision variables $x_{C,i\varepsilon T}$ which represents the width of configuration C inside all segments that have a reserved height of $i\varepsilon T$. To be able to state the constraints for the linear program, we introduce some new parameters and notation. Let

- S_{ver} be the set of all segments,
- $h_{s,\text{ver}} \in \{i\varepsilon T \mid i = 0, \ldots, 1/\varepsilon\}$ be the guessed height reserved for small and vertical jobs inside each segment $s \in S_{\text{ver}}$,
- $S_{\text{ver},h}$ be the set of all segments of height exactly h,
- $\mathcal{C}$ be the set of all configurations,
- $a_\eta(C)$ denote the number of jobs present in C of height exactly η,

- $\mathcal{C}_h$ denote the set of configurations of height at most h.

With these parameters, we now define the configuration LP as the follow equations.

$$\sum_{C \in \mathcal{C}_{i\varepsilon T}} x_{C,i\varepsilon T} = w(S_{\text{ver},i\varepsilon T}), \qquad \forall i \in \{1, 2, \ldots, 1/\varepsilon\} \tag{3.7}$$

$$\sum_{s \in S} \sum_{C \in \mathcal{C}_{h_{s,\text{ver}}}} a_\eta(C) x_{C,i\varepsilon T} = \sum_{\substack{j \in \mathcal{J}_{\text{ver}} \\ h(j)=\eta}} w(j), \qquad \forall \eta \in \{h(j) | j \in \mathcal{J}_{\text{ver}}\} \tag{3.8}$$

$$x_{C,i\varepsilon T} \geq 0, \qquad \forall C \in \mathcal{C}, i \in \{1, 2, \ldots, 1/\varepsilon\} \tag{3.9}$$

The function $w(x)$ denotes the width of object x. Note that $w(j) = w_j$, we have chosen to use the w-notation to be coherent with the expression in equation (3.7). To place our jobs $j \in \mathcal{J}_{\text{ver}}$, we simply place the jobs into the configurations, slicing when something does not fit. In the demand strip packing problem, jobs may be sliced but must still be placed contiguously across the time axis. The fractional placement of jobs caused by this slicing step can therefore result in a schedule that is not valid for this problem. These fractional jobs that we slice are therefore removed. The details on how these removed jobs are handled will be covered later.

Equation (3.7) ensures that for each possible height $i\varepsilon T$, the width of all configurations of at least that height exactly matches the total width of all segments that have the same height reserved. In equation (3.8) we get one constraint for each rounded height η that is attained by some vertical job $j \in \mathcal{J}_{\text{ver}}$. Looking over all segments, we consider the configurations that fit inside each segment (configurations with a maximum height equal to the reserved height inside each segment). For these configurations, we ensure that the width of all jobs of height η contained inside them exactly matches the width of all the jobs with height η. By considering all heights attained by vertical jobs, we ensure that the configurations can contain all the vertical jobs. The last constraint is trivial, giving a configuration a negative width would be rather hard to interpret. However, the LP is of course only of use to us if we can ensure that it has a solution.

Claim 3.11

For a sufficiently large height T, The configuration LP is always solvable.

Proof. Let σ be an optimal packing to the problem. Consider specifically the vertical jobs $\mathcal{J}_{\text{ver}}$. The packing assigns each vertical job $j \in \mathcal{J}_{\text{ver}}$ a start time t_j and a processing time p_j. Let

$$\mathcal{T} = \{t_1, \ldots, t_n, t_1 + p_1, \ldots, t_n + p_n\}, \; n = |\mathcal{J}_{\text{ver}}| \tag{3.10}$$

be the set of all start and end points of vertical jobs. Between each $t \in \mathcal{T}$, we introduce a segment I_k, i.e. $T_k = [t_k, t_{k+1})$. For each I_k, we also introduce a configuration C_k for the jobs contained inside I_k. In the context of the configuration LP, we let S_{ver} be the set of intervals I_k and let $\mathcal{C}$ be the set of configurations C_k.

Let $h_{s,\text{ver}} = h_{I_k,\text{ver}}$ be the height reserved for the vertical jobs contained in segment $s = I_k$ (rounded up to the next $i\varepsilon T$). The decision variables $x_{C,i\varepsilon T}$ can be defined as

$$x_{C,i\varepsilon T} = \begin{cases} t_{k+1} - t_k, \mid C = C_k \wedge i\varepsilon T = h_{I_k,\text{ver}} \\ 0 \text{ otherwise} \end{cases} . \tag{3.11}$$

We now verify that the constraints of the LP hold. In equation (3.7), the LHS represents the total width of all configurations C inside segments with height $i\varepsilon T$. The RHS is the width of the segments of that height. Since we have constructed the configurations to match the segments, this constraint clearly holds.

In equation (3.8), the LHS determines the total width assigned for a specific height η. As the number of active jobs does not change inside a segment thanks to our segment definition, we can simplify the expression to simply counting the width of jobs of height η, precisely what the RHS is counting. Using our definition of $x_{C,i\varepsilon T}$, the constraint becomes

$$\sum_{s \in S} \sum_{C \in \mathcal{C}_{h_{s,\text{ver}}}} a_\eta(C) x_{C,i\varepsilon T} = \sum_{k=1}^{|S|} a_\eta(C_s)(t_{k+1} - t_k), \tag{3.12}$$

which represents the width of jobs of height η. The constraint is therefore satisfied.

Finally, equation (3.9) is obviously satisfied as the width of segments (and therefore configurations) are positive as we define them to run between distinct increasing times. We have thus shown that for an optimal packing σ, the configuration LP must have a solution. □

The placement of jobs from the configuration LP ends up placing jobs into bands as each occurrence of a height in a configuration C becomes a contiguous run of jobs of that same height, i.e. a band. Each configuration will contain one or more bands inside itself, so when the configurations are placed into the schedule when solving the LP, we end up with a schedule were vertical jobs are scheduled into bands.

In order to bound the number of bands, we must also consider the slicing that occurs at the borders of the segments γ. Each nonzero component in the LP can give rise to multiple bands, as the width of placed jobs may be split across several segments with the same reserved height. As each vertical job has a height of at least δT, we know that we can at most fit $\mathcal{O}(1/\delta)$ different vertical jobs on top of each other inside a configuration. Each of these jobs will give rise to a band for each nonzero $x_{C,i\varepsilon T}$ or at a segment border. In total, we can therefore bound the number of bands by

$$\left(\mathcal{O}\left(\frac{1}{\varepsilon\delta}\right) + \frac{1}{\gamma}\right) \mathcal{O}\left(\frac{1}{\delta}\right) = \mathcal{O}\left(\frac{1}{\varepsilon\delta^2\lambda}\right), \tag{3.13}$$

as $\gamma \in \mathcal{O}(\varepsilon\delta\lambda)$. To obtain a compliant packing, we also want to number of unique widths and starting positions of bands to not depend on n.

Claim 3.12

The widths of bands can be rounded to values $i\mu D,\ i \in \mathbb{N}$ and the number of distinct starting positions of bands can be reduced to $\mathcal{O}_n(1)$ options. All jobs that need to be removed to account for this rounding fit inside a strip of width $\lambda D/16$.

Proof. For each band, round its' width up to the next integer multiple of μD. As some bands may now overlap, some jobs may need to be removed. Rather than considering each band, we can look at the up to $1/\delta$ bands stacked on top of each other. There are $\mathcal{O}(1/(\varepsilon\delta\lambda))$ of these since the number of bands is bounded by $\mathcal{O}(1/(\varepsilon\delta^2\lambda))$. The total area of removed jobs can be bounded by

$$\mu DT \cdot \mathcal{O}\left(\frac{1}{\varepsilon\delta\lambda}\right). \tag{3.14}$$

Since $\mu \leq c\varepsilon^5\delta$ as per lemma 3.6 and since $\lambda \in \mathcal{O}(\varepsilon)$, it is clear that the area of the removed jobs must be less than $\lambda DT/32$. Hence, we can use Steinberg's algorithm to place the jobs inside a container of width $\lambda D/16$ and height T inside Ψ.

For reducing the number of starting positions, we can use a left-shifting technique. Inside each of the $1/\gamma$ segments, shift all bands as far left as possible, either until they hit the segment border or until they hit the right edge of another band. This shifting ensures that all bands start either at a segment γ or at the end of another band. As each band now has a width of at least μD, we know that at most $1/\mu$ bands can be placed next to each other. There are also $1/\mu$ distinct possible widths to choose from. Hence, the total number of starting postions of bands can be bounded by

$$\frac{1}{\gamma} \cdot \left(\frac{1}{\mu}\right)^{\frac{1}{\mu}} \leq (1/\varepsilon)^{(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}} \tag{3.15}$$

Since $\mu \geq \varepsilon^{\mathcal{O}(1/\lambda)}$. Although a very large constant, not having a dependency on n is sufficient. □

Finally, we need the following result for later when placing the removed jobs.

Claim 3.13

The fractional vertical jobs that are removed when sliced during the placement with the configuration LP can be placed inside a vertical quarter of the residual job container Ψ, i.e. a container with width $w(\Psi)/4 = \lambda D/4$ and height $h(\Psi) = T$.

Proof. We start by obtaining a bound on the total area of fractional jobs. Slicing may occur either at the end of a configuration or at a segment border. Vertical jobs have a maximum width of μD and may be stacked up to T in height inside each

configuration. Thus, we can bound the area of fractional jobs by

$$\mu DT\left(\mathcal{O}(1/(\varepsilon\delta)) + 1/\gamma\right) \tag{3.16}$$

The $1/\gamma$ term will dominate the expression. From claim 3.9, we choose $\gamma \in \mathcal{O}(\varepsilon\delta\lambda)$ and thus the area can further be bounded by $\mathcal{O}(\mu/(\varepsilon\delta\lambda))DT$. From lemma 3.6, we choose μ such that $\mu \leq c\varepsilon^5\delta$ for any constant c. Hence, we can further bound the area by

$$\mathcal{O}(\mu/(\varepsilon\delta\lambda))DT \leq cDT \cdot \mathcal{O}(\varepsilon^4/\lambda) \leq c\lambda DT, \tag{3.17}$$

where the final inequality holds because $\lambda \in \mathcal{O}(\varepsilon)$. Since this holds for any c, it in particular holds for $c = 1/8$, bounding the maximum area of fractional jobs by $\lambda DT/8$. We can now use Steinberg's algorithm to place these jobs into one vertical quarter of Ψ as the jobs have at most half the area of the container and the width of any one job is at most $\mu D \leq \lambda D$. □

3.2 Horizontal and large jobs

Next, we consider the horizontal and large jobs. In order for the restructured packing σ to be compliant, we require that the horizontal jobs are placed into stacks and that the total number of these stacks is bounded by some constant m, in this case $m = \mathcal{O}(1/(\varepsilon\delta))$. We also require that these stacks of jobs only start at starting points $s \in \mathcal{S}$ and that the number of used starting points is bounded. There are no specific requirements on the placement of large jobs other than that they should start at positions $s \in \mathcal{S}$. This is formalized in the following lemma.

Lemma 3.14

Let σ be a packing with height T. By adding at most $4\varepsilon T$ to the packing height, it is possible to transform the packing to a new packing σ' where there are at most $\mathcal{O}(1/(\varepsilon\delta))$ stacks of horizontal jobs and each of these stacks starts at one of $\mathcal{O}(1/(\varepsilon\delta))$ starting positions.

Yet again, we will reuse many methods from [Dep+23] for this proof. Analogously to the rounding of heights for vertical jobs, we first want to reduce the number of widths of horizontal jobs present. With geometric grouping, we can round the widths of horizontal jobs such that the total number of widths are bounded by $\mathcal{O}(\log(1/\delta)/\varepsilon)$. This rounding will increase the total height of the packing by at most $2\varepsilon T$. For the rest of this section, it is important to be clear whether we are talking about the original or rounded widths of jobs. Therefore, let w_j denote the original width of job j and let w'_j denote the rounded width of job j, $j \in \mathcal{J}_{\text{hor}}$. One major consequence of using rounded jobs is that some jobs may need to be sliced (along their long axis) and they will therefore need to be fractionally placed in our restructured packing instead. It is this rounding and fractional placement that necessitates an additional $2\varepsilon T$ height on top to guarantee that the packing still will be feasible. The details on where the need for this additional height comes from is covered in the coming claims.

Claim 3.15

We can reduce the number of starting points for (rounded) horizontal and large jobs such that $|\mathcal{S}| \leq (1/\varepsilon)^{(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}}$, where $\mathcal{S}$ denotes the set of starting points. This reduction has no impact on the maximum height of the packing.

Proof. Consider all large and horizontal jobs starting in the first segment. Since a large or horizontal job is wider than a segment, no job may end in the first segment. Hence, shifting all these jobs to the left such that they have starting time 0 will not have an impact on the maximum height of stacked jobs in the first segment. We can repeat this argument for the second segment. Consider a job j starting in the second segment. If there is not a job ending before the start of j, we can simply shift the start of j to γD whilst not affecting the height of the packing. A problem may arise however if there is a previous wide job in the first segment, as we cannot shift j over without the jobs overlapping. If this occurs, we can instead shift j over as far left as possible until the start of j is the same as the endpoint for the previous jobs, ensuring that no jobs overlap. We repeat this procedure for all segments, ensuring that all large and horizontal jobs either start at a multiple of γD or start at the endpoint of some job $j \in \mathcal{J}_{\text{hor}} \cup \mathcal{J}_{\text{large}}$. Up to $1/\delta$ horizontal and large jobs may be placed directly next to each other. Thus, we need to consider all possible subsets of wide jobs containing at most $1/\delta$ jobs to account for all possible starting positions. The set $\mathcal{S}$ of starting points can therefore be reduced to

$$\mathcal{S} = \left\{ i\gamma D + \sum_{j \in \mathcal{J}_{\text{hor}} \cup \mathcal{J}_{\text{large}}} w'_j k_j \;\middle|\; i = 0, 1, \ldots, 1/\gamma,\ k_j \in \{0,1\}\ \forall j,\ \sum_{j \in \mathcal{J}_{\text{hor}} \cup \mathcal{J}_{\text{large}}} k_j \leq 1/\delta \right\}. \tag{3.18}$$

We know that there are $1/\gamma$ segments and at most $\mathcal{O}(\log(1/\delta)/\varepsilon)^{1/\delta}$ possible combinations of rounded widths to add for the different possible endpoints and hence we get $|\mathcal{S}| \leq 1/\gamma \cdot (\log(1/\delta)/\varepsilon)^{1/\delta} = (1/\varepsilon)^{(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}}$, where the last equality can be derived using bounds on δ from lemma 3.6. □

For the horizontal jobs in particular, we can further reduce their starting positions. Using lemma 2.14, the number of unique starting positions can be reduced to $\mathcal{O}(1/(\varepsilon\delta))$. This increases the packing height by $2\varepsilon T$, as well as removing some additional jobs. This reduction also allows us to round the height of the stacked horizontal jobs to an integer multiple of μT. As $\mu \leq c\varepsilon^5\delta$, the additional removed jobs fit inside a container of height $\mathcal{O}(\varepsilon)T$ which we place on top of the schedule.

With the starting points reduced, we return to proving the rest of lemma 3.14 by showing how the new set of starting points will result in the horizontal and large jobs are placed into stacks. Similarly to the vertical jobs, we will reorganize the packing using a linear program to show this structure. First, we deal with the large jobs by guessing their starting positions from our set $\mathcal{S}$. With our bound on the size of $\mathcal{S}$ obtained earlier in claim 3.15, we have a total of $|\mathcal{S}|^{|\mathcal{J}_{\text{large}}|} = (1/\varepsilon)^{(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}}$ possible guesses. For the horizontal jobs, we can also guess which starting positions in $\mathcal{S}$ to

use following claim 2.14. Let $\bar{\mathcal{S}}$ denote the used starting points for horizontal jobs. For each starting point $s \in \bar{\mathcal{S}}$, we let $h_{\text{res},s}$ denote residual height left at position s after the large jobs have been placed using the previous guess. Let Ξ denote the set of rounded widths of horizontal jobs, i.e. $\Xi = \{w'_j \mid j \in \mathcal{J}_{\text{hor}}\}$. We can now introduce the linear program

$$\sum_{\xi \in \Xi} \sum_{\substack{s' \in \bar{\mathcal{S}} \\ s' \leq s \leq s' + \xi}} x_{\xi,s'} \leq h_{\text{res},s}, \qquad \forall s \in \bar{\mathcal{S}}, \tag{3.19}$$

$$\sum_{s \in S} x_{\xi,s} = \sum_{\substack{j \in \mathcal{J}_{\text{hor}} \\ w'_j = \xi}} h_j, \qquad \forall \xi \in \Xi, \tag{3.20}$$

$$x_{\xi,s} \geq 0 \qquad \forall s \in \bar{\mathcal{S}}, \forall \xi \in \Xi, \tag{3.21}$$

where the decision variable $x_{\xi,s}$ denotes the height of all the jobs with rounded width ξ starting at position s. The first equation gives one constraint for each of the reduced starting points in $\bar{\mathcal{S}}$. For each starting point s, we consider all possible rounded widths in Ξ and all other starting points s' at most ξ to the left of s. The constraint ensures that the amount of reserved height $h_{\text{res},s}$ at s is not exceeded by any other jobs starting previously and overlapping s. The second equation gives a constraint for each rounded width ξ present and ensures that for each width ξ, the total height of all these jobs (RHS of the equation) is placed with our decision variables (LHS). As we ensure that all height is covered, we ensure that all jobs are placed into the schedule. The final equation is trivial, as we do not allow for negative amounts of height to be assigned.

We have previously defined a stack as a set of jobs of the same (rounded) width that are placed on top of each other in a packing. If we view the fractional jobs that are placed using the LP as normal jobs, we see that each $x_{\xi,s}$ will result in one stack of width ξ. The fact that the jobs are placed fractionally does not matter, we will still end up with all jobs of a certain (rounded) width placed into stacks containing only said width. If we can show that the above LP is solvable, we will show that the packing can be transformed such that almost all horizontal jobs are placed into stacks. Just like the proof of the configuration LP in the previous section, we will consider the number of nonzero components in the solution by checking the number of constraints present. We can see that we have

$$|\bar{\mathcal{S}}| + |\Xi| = \mathcal{O}\left(\frac{1}{\varepsilon\delta}\right) + \mathcal{O}\left(\frac{\log 1/\delta}{\varepsilon}\right) = \mathcal{O}\left(\frac{1}{\varepsilon\delta}\right)$$

constraints and therefore $\mathcal{O}(1/(\varepsilon\delta))$ nonzero components in a basic feasible solution to the linear program. As each nonzero $x_{\xi,s}$ corresponds to one stack, it is clear that the number of stacks is bounded by $\mathcal{O}(1/(\varepsilon\delta))$.

3.3 Small jobs

The small jobs only represent a small fraction of the total area that is to be packed but we still need to show that they can fit into the packing. Packing the vertical,

horizontal and large jobs is the most problematic, hence why more complicated techniques are used to show that they can be placed. For the small jobs, the idea is to fit them into the gaps left over, which we formalize with the height profile of the other jobs in the following lemma.

Lemma 3.16

Let σ be a packing of large, vertical, and horizontal jobs with height T. If the height profile of the vertical jobs has at most $\mathcal{O}(1/(\varepsilon^4\delta))$ jumps, it is possible to either fit all jobs inside using height at most T, or to find a packing that has a height of at most $(1+\varepsilon)T$.

Proof. For this proof, we will make use of the NFDH algorithm and the associated lemma 2.6 stated previously. If we can show that there is enough space left over, lemma 2.6 guarantees that the small jobs can be placed using the NFDH algorithm. When placing the vertical jobs using the configuration LP, we introduced the variable $h_{s,\text{ver}}$ which represented the height reserved for vertical and small jobs inside segment $s \in S_{\text{ver}}$. As each segment is not fully filled with vertical jobs, there will be some left over space in these segments where we can squeeze in some small jobs. Reserving this height for small and medium jobs also allows us to ignore the presence of horizontal and large jobs.

With the profile of vertical jobs having at most $\kappa = \mathcal{O}(1/(\varepsilon^4\delta))$ jumps, we split the schedule into κ boxes $B_i, i = 1, \ldots, \kappa$. Since we round the height profile of vertical and small jobs up to the the next multiple of εT, each jump will give rise to a box with a minimum height of εT. Let $\mathcal{B}$ denote the set of these boxes. The total area of all boxes $B_i \in \mathcal{B}$ will be at least as large as the size of all small jobs, i.e.

$$\sum_{i=1}^{\kappa} \text{area}(B_i) \geq \text{area}(\mathcal{J}_{\text{small}}). \tag{3.22}$$

There is however no guarantee that the jobs actually can be placed just because the area is sufficient. Let w_i denote the width of box B_i and let h_i denote the width of box B_i. We will use lemma 2.6 over all boxes to place as many small jobs into these boxes as possible. The small job can be bounded inside a square with sides $c\mu$, where $c = \max(D, T)$. From lemma 2.6, a total area of

$$\sum_{i=1}^{\kappa} w_i h_i - (w_i + h_i)c\mu \tag{3.23}$$

of small jobs can always be placed into the boxes by using the NFDH algorithm. At most

$$\sum_{i=1}^{\kappa} (w_i + h_i)c\mu \tag{3.24}$$

of the space inside the boxes will be wasted. Since the total area of the boxes is at least as large as the area of all small jobs, we are left with 2 possibilities. If $\text{area}(\mathcal{J}_{\text{small}}) \leq \sum_{i=1}^{\kappa} w_i h_i - (w_i + h_i)c\mu$, all small jobs fit inside the boxes B_i and no

more work is needed. If $\text{area}(\mathcal{J}_{\text{small}}) \geq \sum_{i=1}^{\kappa} w_i h_i - (w_i + h_i)c\mu$, some amount of small jobs with a maximum area of $\sum_{i=1}^{\kappa}(w_i + h_i)c\mu$ will not fit inside. We place as many jobs as possible inside the gaps in the schedule and place the remaining jobs on top of the schedule. Since $\sum_{i=1}^{\kappa} w_i \leq D$, $\sum_{i=1}^{\kappa} h_i \leq \kappa T$ and $\mu \leq d\varepsilon^5\delta$ for any constant d, we can bound the area of the remaining small jobs by

$$\sum_{i=1}^{\kappa}(w_i + h_i)c\mu \leq \kappa DT \cdot cd\varepsilon^5\delta \leq d\varepsilon DT. \tag{3.25}$$

The last inequality only holds in general for $\kappa \leq \mathcal{O}(1/(\varepsilon^4\delta))$. The different cases for $c = \max(D, T)$ do not have an impact as we simply choose an appropriate d to counteract it. With an appropriate d, the area of the remaining jobs is small enough that they can be placed into a container of width D and height εT on top of the schedule using lemma 2.6. We can therefore always either place all small jobs inside the schedule or place them all by adding at most εT to the packing height. □

With lemma 3.16 proven, all that remains is to show that the packing of vertical jobs achieved with the configuration LP results in at most $\mathcal{O}(1/(\varepsilon^4\delta))$ jumps. For vertical jobs, the profile may change either at the border of each of the $1/\gamma$ segments or at each of the $\mathcal{O}(1/(\varepsilon\delta))$ configurations. Between each jump we introduce a box B_i with a height of at least εT. There can be at most

$$\kappa^* := \mathcal{O}(1/(\varepsilon\delta)) + 1/\gamma \tag{3.26}$$

jumps in the height profile. As $\gamma \in \mathcal{O}(\varepsilon\delta\lambda)$ and $\lambda \in \mathcal{O}(\varepsilon)$, it is clear that the number of jumps

$$\kappa^* = \mathcal{O}(1/(\varepsilon\delta)) + 1/\gamma = \mathcal{O}(1/(\varepsilon\delta)) + \mathcal{O}(1/(\varepsilon\delta\lambda)) \leq \mathcal{O}(1/(\varepsilon^4\delta)). \tag{3.27}$$

We can thus use lemma 3.16 to guarantee that all small jobs may be placed whilst adding at most εT to our packing height.

3.4 Medium and removed jobs

We are now left with the medium jobs, as well as a few vertical jobs that were previously removed. To place them, consider the following lemma.

Lemma 3.17

Let σ be a transformed packing with vertical, horizontal, large and small jobs that has a $(k, m, \mathcal{S})$-layout. It is then possible to place all medium jobs whilst adding at most $\mathcal{O}(\varepsilon)T$ to the packing height.

Proof. From lemma 3.6, we know that the total area of medium jobs is bounded by $\lambda DT/8$ with the μ and δ chosen to obtain the compliant packing. With this bound on the area, we can try to apply Steinberg's algorithm to place the medium jobs

into parts of the residual job container Ψ. First, consider all medium jobs with a width $w_j \leq w(\Psi)/8 = \lambda D/8$. Using Steinberg's algorithm, we can guarantee that these jobs will fit into one quarter of Ψ as no job has a width greater than half of a quarter of the container and the total area of the jobs $\lambda DT/8 \leq 1/2 \cdot \lambda DT/4$.

Next, consider the rest of the medium jobs, i.e. the jobs with width $w_j > \varepsilon D/8$. Consider a container of width D and height $2\lambda T$. Since the total area of medium jobs is bounded by $\lambda DT/8$ and these jobs have a width of at least $\lambda D/8$, we know that they must have a height of at most

$$\frac{\lambda DT/8}{\lambda D/8} = \lambda T. \tag{3.28}$$

We can yet again use Steinberg's algorithm to ensure that these jobs will fit into this container as the maximum height of the jobs is at most half the height of the container. This additional container can be placed on top of the schedule, adding $2\lambda T$ to the peak demand. Since all medium jobs fall either have width either greater than $\varepsilon D/8$ or not, these two cases cover all medium jobs and thus all medium jobs can be placed. Since $\lambda \in \mathcal{O}(\varepsilon)$, we add at most $\mathcal{O}(\varepsilon)T$ to the peak demand of the packing. □

We have now shown that almost all jobs can be placed into the schedule. The only jobs that remain are a few of the vertical jobs that we previously removed. To place these, we can use the remaining half of Ψ. When rounding the heights of vertical jobs, we removed jobs where the height profile jumped by more εT. In claim 3.9, we proved that it will always be possible to choose our constant γ for the number of segments such that these removed fractional jobs can be placed into one quarter of Ψ. With this claim, we now place these jobs into half of the remaining part of Ψ with width $\lambda D/4$.

When rounding the starting positions and widths of bands, a few vertical jobs also needed to be removed. We showed in claim 3.12 that these removed jobs fit inside a container of width $\lambda D/16$ as the total width of the jobs is bounded by $\lambda D/32$. We use $1/16$ of Ψ to place these jobs.

There is one remaining set of jobs left. When vertical jobs are placed using the configuration LP, some jobs end up being placed fractionally as jobs are sliced at the borders of segments or end of configurations. In claim 3.13, we showed using Steinberg's algorithm that these jobs also can be placed inside one quarter of Ψ. We have precisely one quarter left, and we will use it for this purpose. We have now shown that all medium and removed jobs either can be placed into the schedule or inside of the residual container Ψ.

3.5 Proof of theorem 3.5

When combined, the proofs from the sections 3.1 through 3.4 will allow us to prove theorem 3.5. The following proof briefly summarizes what we have shown so far.

Proof. In the previous sections, we have shown that each of the different types of job either can be placed into the schedule in a systematic way, barring the few exceptions that are removed and instead placed into the container Ψ. To prove theorem 3.5, we need to show that the previous steps result in a compliant packing. Let σ' be a packing which places the large, horizontal and small jobs, as well as the vertical jobs that don't need to be removed. It is clear that this packing contains no medium jobs, satisfying the first condition of the definition. All vertical jobs that are placed are placed into bands and the number of bands is bounded by $\mathcal{O}(1/(\varepsilon\delta^2))$. All large and horizontal jobs start at points in $\mathcal{S}$ (or a smaller subset of $\mathcal{S}$) and the horizontal jobs specifically are placed into $\mathcal{O}(1/(\varepsilon\delta))$ stacks. Finally, all small jobs either fit inside the schedule or some are added on top adding at most εT to the optimal height. The packing σ' therefore satisfies the definition of a $(k, m, \mathcal{S})$-compliant packing where $k = \mathcal{O}(1/(\varepsilon\delta^2))$ and $m = \mathcal{O}(1/(\varepsilon\delta))$.

The packing σ' packs all jobs except the medium jobs and a small amount of vertical jobs. We have previously shown that all these jobs can be placed inside Ψ in addition to adding at most $4\lambda T$ by placing some jobs on top of the schedule. As $\lambda \in \mathcal{O}(\varepsilon)$, we end up with a restructured packing with a peak height of $(1 + \mathcal{O}(\varepsilon))T$. All requirements in theorem 3.5 are thus satisfied. □

4

(ε, T)-neat packings

For packings that can be transformed into a (ε, T)-neat packing, we develop a similar structural result to the result for λ-forgiving packings. Whilst many of the techniques are similar, there are some major differences that necessitate a somewhat different approach for finding a useful structure. Most notable, we are no longer guaranteed to have a $\lambda D \times T$ sized gap to place the residual jobs.

The structural result we prove is largely based on the algorithm presented in appendix B in [Ebe+25]. Our algorithm bares much resemblance to that shown in the cited paper, with adjustments made to allow for moldable jobs. Many of the steps for proving our structural result will therefore closely resemble the algorithm in the paper above, although some steps are simplified as we are not interested in for example proving that the steps taken run in polynomial time.

Each job $j \in \mathcal{J}$ has an associated set of processing configurations C_j. For the structural result, we only need to consider the specific choice of processing configurations that are used in the optimal packing. For the rest of this section, we therefore let w_j denote w_{ij}^{σ} and let h_j denote h_{ij}^{σ}. Determining which processing configuration we need to achieve this optimal packing is left to our actual algorithm in section 5.2

Like the λ-forgiving packings, we start by partitioning the jobs into different sets depending on their shapes. This is done by comparing the jobs' width and height with the height T and parameters δ and μ. Let $T_{LB} = \max\{\text{area}(\mathcal{J})/D, \max_{j\in\mathcal{J}}\{h_j\}\}$. We define four following sets of squeezable, flat, tall and big jobs[1].

- $\mathcal{J}_{\text{squeeze}} := \{j \in \mathcal{J} \mid h_j \leq T/2,\ w_j \leq \delta D\}$
- $\mathcal{J}_{\text{flat}} := \{j \in \mathcal{J} \mid h_j \leq \mu T_{LB}\} \setminus \mathcal{J}_{\text{squeeze}}$
- $\mathcal{J}_{\text{tall}} := \{j \in \mathcal{J} \mid h_j > T/2\}$
- $\mathcal{J}_{\text{big}} := \mathcal{J} \setminus (\mathcal{J}_{\text{squeeze}} \cup \mathcal{J}_{\text{flat}} \cup \mathcal{J}_{\text{tall}})$

[1]The names here, while not standard, have been chosen to clearly distinguish these sets from the previous partitioning done for λ-forgiving packings

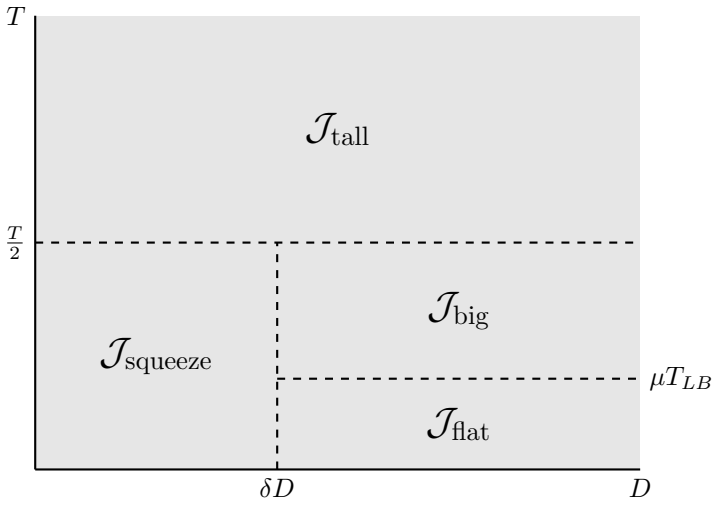


Figure 4.1: The partitioning scheme for jobs in the (ε, T)-neat case. If a job j starts in the bottom left corner, it will belong to whichever set its top right corner is located in.

Note that we no longer need to consider any medium jobs. Figure 4.1 shows the job partitioning for (ε, T)-neat packings. The partitioning depends on the parameters δ and μ. Unlike the λ-forgiving packings, our algorithm will not need to find a suitable value for these. By introducing a scaled version of ε denoted ε', we simply choose them as

$$\delta = \frac{\varepsilon}{1+\varepsilon}, \tag{4.1}$$

$$\mu = \frac{\varepsilon'^3}{\lceil \log(1/\delta) \rceil}. \tag{4.2}$$

For the structural result to hold, we need $\varepsilon' \leq \varepsilon/15$. The motivation for these choices will become clearer later when constructing the actual algorithm. For now, it is sufficient to be aware of their values. For the tall jobs, we also need some additional margins to handle potential fractional jobs. To do this, we introduce the concept of a k-translated height profiles.

Definition 4.1

Let σ be a packing of jobs $\mathcal{J}$ and let $h(t)$ denote the packings associated height profile, i.e. $h(t) = \sum_{j \in \mathcal{J}^\sigma(t)} h_j$. The *$k$-translated* height profile $h_k(t)$ is then defined as

$$h_k(t) = \begin{cases} h(t), & t < kD \\ h(t - kD), & kD \leq t \leq \sum_{j \in \mathcal{J}} w_j \\ 0, & t > \sum_{j \in \mathcal{J}} w_j \end{cases}. \tag{4.3}$$

In order for the algorithm to work, we need to show that a suitable packing can be transformed in a particular way. To do this, consider the following definition.

Definition 4.2

Consider a packing σ of jobs $j \in \mathcal{J}$. We say that the packing σ is *cool* if

1. Tall jobs are placed next to each other from the left according to their heights in non-ascending order,
2. The profile of the tall jobs can be extended such that the profile is δ-translated.
3. Flat and big jobs start at starting positions $s \in \mathcal{S}$, where $|\mathcal{S}| \leq (2/(\delta\mu))^{1/\delta}$.
4. Squeezable jobs are placed into an empty area in the schedule with a width of at least $\frac{\varepsilon}{1+\varepsilon}D$ and height of at least OPT/2.
5. Almost all flat jobs can be placed inside containers and the number of containers is bounded by a constant.

Using definition 4.2, we state the following theorem.

Theorem 4.3

Let $\varepsilon > 0$. Given an optimal packing σ which admits a (ε, T)-neat packing, there exists a restructuring of its jobs such that almost all jobs can be placed into a new packing σ' which is cool. The few remaining jobs can be placed inside a container on top of the schedule. The peak demand of this restructured packing is at most $(1 + 15\varepsilon')T$

Once we have shown that an optimal packing can be transformed in this particular way, we can construct an algorithm which is able to find a cool packing. We will start by considering the tall jobs.

4.1 Tall jobs

We start by placing the tall jobs into the schedule. These are the simplest to handle as the way they are placed is fundamental to the definition of an (ε, T)-neat packing. In order to reduce the number of unique height present, we want to round the heights of all tall jobs. We will do this using lemma 2.11. Since $T \geq T_{LB}$, we can apply the lemma using a height of T_{LB}. For each tall job, we round the height up to the nearest multiple $\varepsilon\delta T_{LB}$ whilst adding at most εT_{LB} to the packing height.

With the jobs rounded, we sort them in non-decreasing order based on their rounded heights. Starting at the very left of the schedule at time $t = 0$, we place the jobs next to each other. Since the jobs have a height greater than $T/2$ and a maximum area of $T_{LB}D$, all tall jobs will fit in the interval $[0, D]$. All tall jobs are thus placed

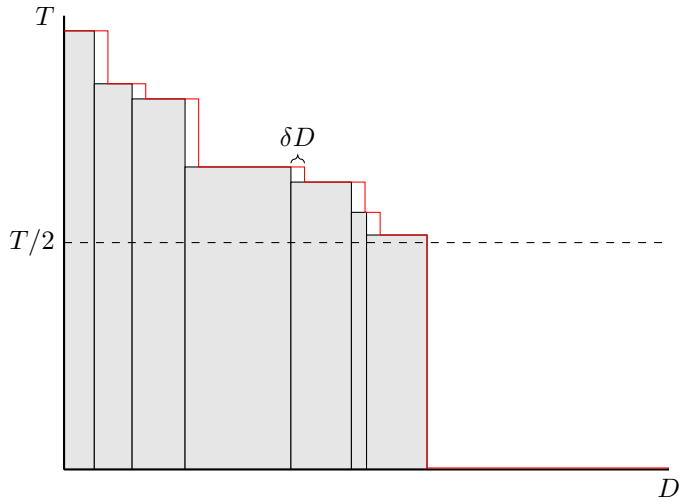


Figure 4.2: A packing of tall jobs. The corresponding δ-translated height profile is shown in red.

in such a way that they satisfy the definition of a cool packing.

For our algorithm, we need to have some additional space to fit fractionally assigned tall jobs. The definition of a cool packing therefore requires that the height profile of the tall jobs can be turned into a δ-translated profile. We therefore make the following assumption.

> **Assumption 1.1**
>
> For an optimal packing, the height profile of tall jobs can be shifted such that it is δ-translated.

We find it to be likely that this assumption holds, although it has not yet been proven. It is also possible that this shift will require some additional height to fit all jobs, which would result in our algorithm having a slightly worse approximation ratio than $3/2 + \varepsilon$. Verifying this assumption is beyond the scope of this thesis, but it will be analyzed in detail at a later date. An example of the δ-translated profile is shown in figure 4.2. For now, we will proceed with proving the structural result with this assumption.

4.2 Flat and big jobs

We will proceed with the placement of the flat jobs. We require that they can be placed inside a bounded number of containers which use a bounded set of starting points. A few jobs may not fit and will instead have to be placed on top of the schedule using Steinberg's algorithm.

Since the flat jobs do not have a minimum guaranteed height, we cannot round their

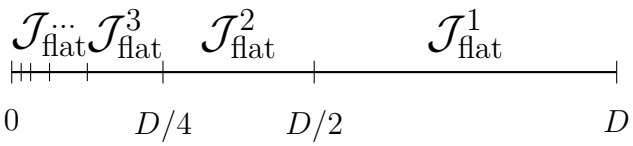


Figure 4.3: The logarithmic partitioning of flat jobs into sets $\mathcal{J}^k_{\text{flat}}$.

heights in the same way as for the tall jobs. We can, however, round their widths using using geometric grouping. Before rounding the flat jobs, we partion them into sets depending on their widths. Let

$$\mathcal{J}^k_{\text{flat}} \coloneqq \left\{ j \in \mathcal{J} \mid \frac{D}{2^k} < w_j < \frac{D}{2^{k-1}} \right\},\ k = 1, \dots, \left\lceil \log\left(\frac{1}{\delta}\right) \right\rceil. \tag{4.4}$$

For each set $\mathcal{J}^k_{\text{flat}}$, we can round the width using geometric grouping. This step will add at most $4\varepsilon' T$ to the packing height. Since the size of the jobs has increased, it is possible that some jobs may now overlap. In order to be able to use the rounded jobs, we allow for fractional placing of jobs, similarly to how we handled the rounded jobs in the λ-forgiving case. This fractional placement corresponds to performing horizontal slicing of the jobs, something which is not allowed in DSP. Later, this fractional packing will be transformed such that almost all jobs can be placed integrally instead, while a few jobs will be removed. To distinguish the integral packings σ from the fractional packings, we denote fractional packings by ς.

We have now obtained a fractional packing, which we want to transform into an integral packing. First, we reduce the number of starting positions for the flat jobs. With lemma 2.14, the number of starting positions for each set $\mathcal{J}^k_{\text{flat}}$ can be reduced to a set with $\frac{2^k-1}{\varepsilon'}$ unique elements. This procedure also reduces the starting positions for big jobs. With the reduced starting positions and rounded widths, we can obtain a feasible integral packing by removing a few additional jobs.

Claim 4.5

By increasing the packing height by at most $4\varepsilon' T$, the fractional packing ς of flat jobs can be transformed into an integral packing σ.

Proof. Consider a specific subset $\mathcal{J}^{k,\ell}_{\text{flat}}$. All of the unrounded jobs have a width $w_j \leq w_\ell$, the width of the auxiliary job. Rather than fractionally placing the jobs inside the auxiliary job, we can attempt to greedily fill them inside. For each $\mathcal{J}^{k,\ell}_{\text{flat}}$, we greedily fill as many of the jobs as possible where their total height is at most xh_{j_ℓ}. Since the jobs have are narrower than the auxiliary job and the total height is smaller, this integral packing will still be feasible.

The greedy filling of the jobs will result in some jobs not being able to fit. We need to bound the area of these removed jobs. Since we greedily fill the jobs inside

fractional segments of the auxiliary jobs, we need to determine a bound on the amount of these. Each auxiliary job fraction can be represented by a vector (s, x, j_ℓ) denoting the fraction x of job j_ℓ starting at s. The number of starting positions is bounded by $\frac{2^k-1}{\varepsilon'}$ and the number of possible fractions for each starting position is bounded by $1/\varepsilon'$. Hence, the number of auxiliary jobs for a given set $\mathcal{J}^k_{\text{flat}}$ can be bounded by

$$\frac{1}{\varepsilon'}\left(\frac{2^k-1}{\varepsilon'}+1\right). \tag{4.5}$$

The additional additive term comes from the slicing of jobs. Since flat jobs have a height of at most μT and $\mu \leq \varepsilon'$, we know that at most one job needs to be removed for each auxiliary fractional component. We also know that $k \in \{1, \ldots, 1/\delta\}$ Hence, we can bound the total area of removed jobs by

$$\log\left(\frac{1}{\delta}\right)\frac{\mu DT}{\varepsilon'^2} \leq 2\varepsilon' DT, \tag{4.6}$$

since $\mu = \varepsilon'^3/\lceil\log(1/\delta)\rceil$. Hence, we can place these removed jobs on top of the schedule in a container of height $4\varepsilon' T$ using Steinberg's algorithm. □

The placement of flat jobs as described so far results in a packing with a lot of structure. Let $\mathcal{N}$ denote the set of endpoints for jobs. For both the rounded flat and big jobs, the number of ending positions $\mathcal{N}$ can be bounded by

$$|\mathcal{N}| \leq \left(\frac{2}{\delta\mu}\right)^{1/\delta}. \tag{4.7}$$

This is done by moving the flat and big jobs to the right, ensuring that they end at D or at most $1/\delta$ flat or big jobs directly to the right.

The placement of the flat jobs can therefore be represented by a set of vectors $f_k = (\nu, h_1, \ldots, h_{|\mathcal{W}_k|})$ denoting the heights of jobs of each rounded width for a given ending position ν. Since neither the number of ending positions nor the rounded widths depend on n, our algorithm is able to iterate over all possible vectors whilst running in polynomial time. The number of starting positions $\mathcal{N}$ has already been bounded above and the number of heights for each width can be bounded further as the stack of flat jobs at any given time must be an integer multiple of μT. The number of widths for each group is also bounded by $1/\varepsilon'$. If $\mathcal{F}_k$ is the set containing all vectors f_k for a given $k \in \{1, \ldots, 1/\delta\}$, we can bound the number of vectors by

$$|\mathcal{F}_k| \leq \left(\frac{2}{\delta\mu}\right)^{1/\delta} \cdot \mu^{1/\varepsilon'} \leq \left(\frac{2}{\delta\mu^2}\right)^{1/\delta}. \tag{4.8}$$

These vectors containing information about the scheduling of flat jobs can model the containers needed for the definition of a cool packing. In fact, since the total number of vectors is bounded by a constant, an algorithm could iterate over all possible vectors until a feasible packing is found.

Because the flat jobs are shifted such that all jobs inside each of the $1/\varepsilon'$ layers have the same ending position, the placement of all flat jobs can be represented by at most $1/\varepsilon'$ vectors from the set $\mathcal{F}_k$. Barring the few jobs that are removed when rounding the widths and starting positions, all flat jobs are placed inside these containers, fulfilling the definition of a cool packing.

In order to pack the jobs, we for each group $k \in \{1, \ldots, \log(1/\delta)\}$ guess $1/\varepsilon'$ vectors from $\mathcal{F}_k$ and starting times for all the big jobs. If the guess results in a feasible packing, we check how tall the resulting packing is. Placing the jobs according to the vectors in $\mathcal{F}_k$ will result in a fractional packing of the flat jobs. Once we have a fractional packing, we can use claim 4.5 to obtain a feasible integral packing instead.

Claim 4.6

If T is sufficiently tall to allow for a feasible packing, then the height of the restructured packing is at most $(3/2 + \varepsilon)T$.

Proof. Suppose that the set of jobs $\mathcal{J}$ and deadline D does admit a (ε, T)-neat packing σ. If this is the case, the height of the packing must be at most $\frac{3}{2}T$. Consider the excess jobs we place on top of the schedule. When rounding the widths of flat jobs using geometric grouping, we add $4\varepsilon' T$ to the packing height. For the fractional packing, we reduce the number of starting positions for the flat and big jobs. This adds $2\varepsilon' T$ to the packing height. In this step, we also remove some additional jobs when slicing the jobs such that the height of the stacked jobs is a multiple of μT. The rounding of the height of tall jobs also increases the packing height by another $\varepsilon' T$.

When guessing the structure of the packing, we verify that the packing achieved has a low enough height. Our guessed fractional structure should have a height of at most $h(\sigma) + (4 + 2 + 1)\varepsilon' T \leq (3/2 + 7\varepsilon')T$, since σ is (ε, T)-neat. In order to transform the fractional packing to an integral packing, we need to consider the jobs we removed earlier. The jobs we remove when greedily filling the auxiliary jobs fit inside a container of height $4\varepsilon' T$ using Steinberg's algorithm. The jobs we remove to achieve a rounded stack height of $i\mu T$ for flat jobs have a maximal area of $\log(1/\delta)\frac{2\mu}{\varepsilon'^2} DT \leq 2\varepsilon' DT$.

These jobs can therefore also be placed on top of the schedule inside a container of height $4\varepsilon' T$ using Steinberg's algorithm. When adding these containers, the maximal height of our integral packing is at most

$$\left(\frac{3}{2} + 7\varepsilon'\right) T + 8\varepsilon' T = \left(\frac{3}{2} + 15\varepsilon'\right) T = \left(\frac{3}{2} + \varepsilon\right) T. \tag{4.9}$$

If the jobs and deadline admit a (ε, T)-neat packing, the packing with structure described will have a maximal height of $(3/2 + \varepsilon)T$. □

4.3 Squeezable jobs

With the tall, flat and big jobs placed, all that remains is the squeezable jobs. To place these jobs into the schedule, we make use of a an algorithm that allows these jobs to be "squeezed" into the schedule without increasing the packing height.

Lemma 4.7: Lemma 2.2 in [Ebe+25]

Let $T \geq \max\left\{\frac{\text{area}(\mathcal{J})}{D}, \text{OPT}\right\}$ and let σ be an (ε, T)-neat packing of $\mathcal{J} \setminus \mathcal{J}_{\text{squeeze}}$. The squeezing algorithm can then compute an (ε, T)-neat packing of $\mathcal{J}$.

Proof. To squeeze the jobs into the schedule, we consider a left-shifting approach. Consider a single squeezable job $j \in \mathcal{J}_{\text{squeeze}}$, i.e. a job with a height $h_j \leq T/2$ and width $w_j \leq \delta D = \frac{\varepsilon}{1+\varepsilon} D$. Since the packing of all other jobs is (ε, T)-neat, we know that it has a maximum height of $(3/2 + \varepsilon)T$. Starting from $t = 0$, we find the first point at which the total height of the packing is at most $(1+\varepsilon)T$. Let τ denote this position. From this position, we locate the first non-tall job ($h_j \leq T/2$) to the right and slide it left such that it now starts at τ instead. Let τ' denote the jobs original starting position.

We repeat the procedure, finding the next point with height $\leq (1+\varepsilon)T$ to the right of τ and sliding the jobs left. This is done until there are no more non-tall jobs to shift. During a single sliding step, the height of the packing can only increase on the interval $[\tau, \tau')$, as no jobs are moved elsewhere. Since the only jobs starting in $[\tau, \tau')$ are tall (as τ' is the first starting position of a non-tall job), and all tall jobs in a (ε, T)-neat packing are placed in non-increasing order, we can be sure that the packing height is at most $(1 + \varepsilon)T$ on $[\tau, \tau')$. Since the job we shift has a height of at most $T/2$, we can be sure that the packing height will not exceed $(3/2 + \varepsilon)T$ anywhere when performing a shift.

After all non-tall jobs have been shifted, some space will have been freed. The position τ still represents the first point at which the height of the packing is at most $(1+\varepsilon)T$, but there are no longer any non-tall jobs to shift left. Since we also can bound the total area of the jobs by DT, we can obtain an upper bound on the final value of τ by

$$\tau \leq \frac{DT}{(1+\varepsilon)T} = \frac{D}{1+\varepsilon} = \left(1 - \frac{1}{1+\varepsilon}\right) D. \tag{4.10}$$

We can see that a gap of at width $\frac{\varepsilon}{1+\varepsilon} D$ will appear towards the right of the schedule. Remember that $\delta = \frac{\varepsilon}{1+\varepsilon}$. The gap is therefore wide enough to contain a single squeezable job. Morevoer, there cannot be any non-tall jobs placed in the gap, since they would then have been shifted left previously. Since there are only tall jobs in the gap and they have are placed in a non-increasing order, we can also be sure that there is at least $T/2$ height left over. We can therefore squeeze the job into this gap, and the final packing will still be (ε, T)-neat.

To place all squeezable jobs, we simply repeat the procedure once for each job $j \in \mathcal{J}_{\text{squeeze}}$. As the packing always remains (ε, T)-neat, the procedure will work for all squeezable jobs. □

4.4 Proof of theorem 4.3

Theorem 4.3 states a given optimal packing σ which admits a (ε, T)-neat structure. can be transformed into a cool packing. We will now verify that the procedure as described above results in a packing that is cool.

For the tall jobs, we require that they are placed next to each other in non-increasing order with respect to their heights, i.e. that the placement of the tall jobs results in a staircase shape starting at $t = 0$ and $h \leq T$ and continuing to a height of $T/2$. We simply place the jobs in this order, as the original (ε, T)-neat packing required.

For the flat and big jobs, we need to be able to guess their placement in polynomial time. Their starting positions are bounded by $\left(\frac{2}{\delta\mu}\right)^{1/\delta} \in \mathcal{O}_n(1)$. For the flat jobs, we have shown that their placement can be represented by at most $1/\varepsilon'$ vectors, picked from $\left(\frac{2}{\delta\mu^2}\right)^{1/\delta}$ possibilities. These vectors represent the containers required as part of the definition of a cool packing. Transforming a packing of tall, flat and big jobs in the described ways results in a packing that still is (ε, T)-neat, importantly that the maximum height is $(3/2 + \varepsilon)T$. With the squeezing procedure, all squeezable jobs can fit inside gaps in the schedule without increasing the packing height. Hence, the restructured packing σ' is cool.

5

Algorithm

With the structural results from chapters 3 and 4, we can proceed with constructing our algorithm. As discussed previously, we will start by constructing two subalgorithms, one for each type of packing. In section 5.3, we combine these with a binary search procedure to obtain our complete algorithm.

5.1 Algorithm for λ-forgiving packings

With the structural result proven, we proceed with constructing an algorithm. In this section, we present an algorithm which for a given height T either generates a D-feasible packing of the jobs or concludes that no such packing is possible. For the full algorithm, we make use of binary search over T to find an optimal solution.

In order to obtain an approximation algorithm, we need all steps to run in polynomial time with respect to the number of jobs n. With the structural result, we can in several cases remove or reduce the dependency on n and instead have certain steps run in constant time with respect to n.

Since the structural result is based on the partitioning of jobs into different sets, the first step of the algorithm must also perform this. This is controlled by the parameters δ and μ. We know from lemma 3.6 that it always is possible to choose the values of μ and δ as subsequent elements from a sequence ρ_i with $\lceil 8/\lambda \rceil$ elements.

Vertical and horizontal jobs should be placed inside bands and stacks respectively. Before we can start placing the jobs, we therefore need to place the bands and stacks into the schedule. Afterwards, we can greedily fill the bands and stacks with their corresponding jobs. To do this, we need to guess both their starting positions, sizes and the number of bands/stacks.

For the heights of bands and stacks and for the widths of bands, we know which values to guess as the rounded heights and widths are multiples of T or D respectively. For the widths of stacks, we have used geometric grouping to round the number of widths to $\mathcal{O}(\log(1/\delta)/\varepsilon)$ options, but the rounded widths still depend on some of the original widths. Since we have not yet assigned which jobs are flat, we cannot

know which values to guess. To handle this, we must first guess which widths appear among our jobs. As each job has at most q processing configurations and there are at most n jobs, the maximum number of unique widths is nq. We now simply guess which $\mathcal{O}(\log(1/\delta)/\varepsilon)$ of these widths to pick, which can be done in $\binom{nq}{\mathcal{O}(1/\delta)/\varepsilon} \leq n^{\mathcal{O}(\varepsilon)}$

We have bounded these quantities in theorem 3.5. The number of bands is bounded by $\mathcal{O}(1/(\varepsilon\delta^2))$ and the number of stacks is bounded by $\mathcal{O}(1/(\varepsilon\delta))$. For bands, their rounded widths belong to one of $\mathcal{O}(1/(\mu\delta))$ possibilities and the number of unique heights is in $\mathcal{O}(1/(\varepsilon\delta))$ as the heights of vertical jobs are rounded. The starting positions for bands are also bounded by $(1/\varepsilon)^{(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}}$. The number of starting position for stacks is bounded by $\mathcal{O}(1/(\varepsilon\delta))$ and their rounded widths are in $\mathcal{O}(\log(1/\delta)/\varepsilon)$. The heights of stackes are rounded to multiples $i\mu T$. Since neither the starting positions, sizes or amounts of bands and stacks depend on n, we can iterate through all possible combinations while still running in polynomial time. Let $\mathcal{K}^{\text{band}}$ and $\mathcal{K}^{\text{stack}}$ denote the set of bands and stacks. For each $k \in \mathcal{K}$, we let w_k, h_k denote the width and height of said band or stack.

For a feasible guess, we must now address the moldability of the jobs. The structural result holds for a specific choice of job configuration. If we use the right job configuration, the structural result will allow us to guarantee that our obtained solution deviates by at most $\alpha = 3/2 + \varepsilon$. Simply guessing and trying all possible job configurations until we find the configuration which yields the optimal value is not possible. Since we for each of the n jobs has multiple options, it is clear that the time complexity of this "brute-force" technique is exponential. Instead, we consider a different approach were we find a suitable configuration by using a linear program.

Definition 5.1

The *Job Assignment LP (JALP)* is a linear program which can be used to assign each job a suitable processing configuration and assign.

In this LP, we let $x_{j,k} \in [0, 1]$ be our decision variable, denoting the fraction of job j that is assigned to bin k. Rather than considering all possible job configurations for each job, we can get away with looking at only a few. For each of the five categories of jobs, we introduce a constraint ensuring that no single category of jobs is assigned too many jobs. When a job is assigned to a category, there may be several processing configurations which would have the job belong to said bin. In this case, we simply choose the configuration which minimizes either the width, height or area of the job, depending on which bin it is placed in. For each category $s \in \{\text{large}, \text{small}, \text{medium}\}$, we let w_j^s and h_j^s denote the width and height of the job associated with the "smallest" processing configuration. For vertical and horizontal jobs, we further introduce one bin for each possible rounded height or width. Let $\mathcal{H}$ and $\mathcal{W}$ denote the sets of rounded heights and widths for bands and stacks

respectively.

$$(w_j^h, h_j^h) = \underset{\substack{(w_{ij},h_{ij})\in C_j \\ w_{ij}<\mu D, h_{ij}\leq h}}{\arg\min} \; w_{ij} \qquad \forall h \in \mathcal{H} \tag{5.1}$$

$$(w_j^w, h_j^w) = \underset{\substack{(w_{ij},h_{ij})\in C_j \\ w_{ij}\leq w, h_{ij}<\mu T}}{\arg\min} \; h_{ij} \qquad \forall w \in \mathcal{W} \tag{5.2}$$

$$(w_j^{\text{large}}, h_j^{\text{large}}) = \underset{\substack{(w_{ij},h_{ij})\in C_j \\ w_{ij}>\delta D, h_{ij}\geq \delta T}}{\arg\min} \; w_{ij}h_{ij} \tag{5.3}$$

$$(w_j^{\text{small}}, h_j^{\text{small}}) = \underset{\substack{(w_{ij},h_{ij})\in C_j \\ w_{ij}<\mu D, h_{ij}<\mu T}}{\arg\min} \; w_{ij}h_{ij} \tag{5.4}$$

$$(w_j^{\text{medium}}, h_j^{\text{medium}}) = \underset{\substack{(w_{ij},h_{ij})\in C_j \\ \text{classified as medium}}}{\arg\min} \; w_{ij}h_{ij} \tag{5.5}$$

It is possible that each job wont be able to be assigned into each bin since there is no guarantee that the job has a suitable processing configuration to choose. A job could in practice have just one configuration and thus potentially only be assignable into one bin. When a job cannot be assigned, i.e. arg min not being able to find an argument, we let arg min return (D, T). This ensures that no job will be assigned into a category for which they do not have a viable processing configuration.

We start by addressing the large jobs. Rather than placing them using the LP, we need to guess their placement into the schedule. As the set of large jobs can have a maximal area of DT and each large job must have an area of at least $\delta^2 DT$, the maximum amount of possible large jobs is

$$|\mathcal{J}_{\text{large}}| \leq \frac{DT}{\delta^2 DT} = \frac{1}{\delta^2}. \tag{5.6}$$

We can round both the heights and widths of large jobs using lemma 2.11 and 2.12 respectively. The number of starting positions for large jobs is bounded by $(1/\varepsilon)^{(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}}$ by claim 3.15. Since none of the guessable quantities depend on n, we guess both the amount of large jobs, their starting positions and their sizes in polynomial time. We do this guessing before solving the JALP.

To place the other jobs, we introduce a set of bins. We introduce one bin each for the small and medium jobs, called β_{small} and β_{medium}. For the vertical and horizontal jobs, we instead introduce one bin for each possible height of band or width of stack respectively. Let $\mathcal{K}_h^{\text{band}}$ denote the set of bands of height h and let $\mathcal{K}_w^{\text{stack}}$ denote the set of all stacks of width w. Let $\mathcal{B}$ denote the set of all bins.

For the vertical jobs, the structural result assumes that some vertical jobs are removed and instead placed inside the residual item container. To account for this, we also need to include some buffer when assigning jobs as vertical into the bins for the different height levels. For each height level h, we introduce a buffer $Б_h$. For each buffer, we guess its width to some multiple $i\mu D$. The total amount of buffer needs to be large enough that for a suitable guess, the total width of vertical jobs fits inside

the bands together with the buffer. The structural result expects vertical jobs with a total width of at most $9\lambda D/32$ to be removed to accommodate the remaining jobs into bands. Hence, we guess all possible buffer widths such that

$$\sum_{h \in \mathcal{H}} Б_h \leq \frac{9\lambda D}{32}. \tag{5.7}$$

The number of possible guesses can be bounded by $\mathcal{O}\left(\frac{1}{\varepsilon\delta\mu}\right)$. Similarly for small jobs, we need to guess the reserved height for small and vertical jobs as we did in the structural result. This is to ensure that we do not assign too many jobs as small. From this guess, we obtain a maximum area for the small jobs. Let A denote this area. With all the guessing done, we can proceed with assigning the jobs using the JALP. It is defined as the following linear program.

$$\sum_{\beta \in \mathcal{B}} x_{j,\beta} = 1, \qquad \forall j \in \mathcal{J} \tag{5.8}$$

$$\sum_{j \in \mathcal{J}} x_{j,h} w_j^h \leq \sum_{k \in \mathcal{K}_h^{\text{band}}} w_k + Б_h \qquad \forall h \in \mathcal{H} \tag{5.9}$$

$$\sum_{j \in \mathcal{J}} x_{j,w} h_j^w \leq \sum_{k \in \mathcal{K}_w^{\text{stack}}} h_k \qquad \forall w \in \mathcal{W} \tag{5.10}$$

$$\sum_{j \in \mathcal{J}} x_{j,\beta_{\text{small}}} h_j^{\text{small}} w_j^{\text{small}} \leq A \tag{5.11}$$

$$\sum_{j \in \mathcal{J}} x_{j,\beta_{\text{medium}}} h_j^{\text{medium}} w_j^{\text{medium}} \leq \frac{\lambda D T}{4} \tag{5.12}$$

Equation (5.8) ensures that for each job, its width is assigned only once, although possible fractionally over multiple bins. The subsequent equations ensure that we do not assign too many jobs to any one category. For the vertical and horizontal bins, we assign each job to a bin corresponding to the set of all bands or stacks of a specific (rounded) height or width.

The bound on the area of the medium jobs in equation (5.12) comes from the bounded area of jobs in lemma 3.6. The rest of the structural result assumes that the area of medium jobs is bounded by $\lambda DT/4$ and thus, we must limit ourselves to the same bound here. With the LP stated, we must now verify that a solution to the LP can be obtained in polynomial time.

To solve the LP, we can make use of one of several standard algorithms. Most practical algorithms such as the simplex algorithm, while quick in practice, still have exponential time complexity [HLC09]. Instead, we use an algorithm which while slow in practice has a guaranteed polynomial running time. Karmakar's algorithm is one such algorithm which we can use to solve the JALP [Kar84]. It is possible that the LP is not solvable. If this is the case, we discard our current guess and make a new guess on either the large jobs, the structure of the bands and stacks or of the parameters μ and δ. If the LP cannot be solved for any guess of these parameters, our algorithm returns this result, indicating that a larger value of T is needed.

Whilst the LP has assigned the jobs into different categories, we still need to place them into the actual schedule.

Claim 5.2

For a correct guess of the packing structure and a sufficiently large T, the JALP will have a solution.

Proof. Across all possible guesses of μ and δ and for the placement of the bands and stacks, there will exist a guess that mimics the structure we proved to exist in the structural result. From this result, we know that there exists a feasible packing whose structure closely resembles that of the LP constraints. This holds as long as the height T is sufficiently tall. If the LP is not solvable for any guess, then the height T too low to pack the jobs.

Equation (5.12) concerns the placement of medium jobs. We know that in our correct guess, the total area of medium jobs is bounded by $\lambda DT/4$. The packing satisfies this constraint. For the small jobs, one guess of the height profile for small and vertical jobs, and by extension the allowed area for small jobs will match the structural result.

The two remaining constraints concern the placement of vertical and horizontal jobs. For the correct guess, we know that all horizontal and almost all vertical jobs can be placed inside stacks and bands respectively. For the stacks, we introduce one bin for each possible rounded width that a horizontal job/stack can have. When the stacks are correctly guessed, we know that their total height is enough to fit the total height of all horizontal jobs.

For the vertical jobs, the situation is similar. There is an additional complication as the structural result only holds if some vertical jobs are removed and instead placed in the residual item container. To account for this, each distinct height levels of bands also has a buffer $Б_h$ where $\sum_{h\in\mathcal{H}} Б_h \leq 9\lambda D/32$. These jobs that need to be removed may be distributed across the different height levels but for a suitable guess of the buffers, there is enough extra space that all vertical jobs fit inside the width of the bands and the buffers. Since a packing of the form proven to exist in the structural result satisfies the constraints and such a structure must exist in at least one of the guesses, the LP will be solvable for a suitable guess. □

Claim 5.3

If a solution to the JALP is found, almost all vertical and horizontal jobs can be placed into the schedule. The few removed jobs can be placed either on top of the schedule or inside Ψ.

Proof. To determine the placement of jobs, we must analyze the constraints. In

total, the LP will have

$$n + \mathcal{O}\left(\frac{1}{\varepsilon\delta}\right) + \mathcal{O}\left(\frac{\log 1/\delta}{\varepsilon}\right) + 2 \tag{5.13}$$

constraints, and therefore the same number of nonzero components in its solution. In turn, this means that at most $\mathcal{O}(1/(\varepsilon\delta))$ jobs may be fractionally assigned. We need to show that these fractionally assigned jobs can be placed integrally into the schedule. For any fractionally assigned vertical jobs, we can make use of Ψ. Since a vertical job has a maximum area of μDT, the total area of fractional vertical jobs can be bounded by $\mathcal{O}(\mu/(\varepsilon\delta))DT$. Since $\mu \leq c\varepsilon^5\delta$, it is clear that the area is small enough that these jobs can be placed inside a narrow strip inside Ψ using Steinberg's algorithm. We reserve a container of width $\lambda D/16$ inside Ψ for these jobs.

For fractionally assigned horizontal jobs, the solution is similar. Horizontal jobs also have a maximal area of μDT and thus we can bound the total area of fractional horizontal jobs by $\mathcal{O}(\varepsilon^4)DT$ in the same way as for the vertical jobs. The jobs are too wide to be placed inside Ψ but their total height is very small. We therefore place them inside a container of height εT and width D which we place on top of the schedule.

If jobs are fractionally assigned as small, we can yet again obtain a useful bound. Since each small job has an area of at most $\mu^2 DT$, the total area of fractional small jobs is bounded by $\mathcal{O}(\mu^2/(\varepsilon\delta))$. We can easily place these inside the same εT tall container that was used for the horizontal jobs.

If a job is fractionally assigned as medium, we cannot easily place it into the schedule as we do not have a useful bound on the area of medium jobs. However, if a job is fractionally assigned as medium, it must also be partially assigned as something else. Fractional assignment of vertical, horizontal and small jobs cause no issues, so we simply place the whole job into one of these containers instead.

With the fractional jobs handled, we want to show that almost all of the remaining jobs can be placed. Starting with the horizontal jobs, equation (5.10) guarantees that the total height of all jobs assigned as horizontal is small enough to fit inside our guessed stacks. To place the horizontal jobs into the stacks, we greedily fill the jobs inside the stacks. Since the reserved height for stacks are rounded to heights $i\varepsilon\delta T$ and horizontal jobs have a height of at most μT, we know that at most one job may need to be removed per stack in order to accommodate all horizontal jobs. As the number of stacks is bounded by $\mathcal{O}(1/(\varepsilon\delta))$, we can bound the total area of removed jobs by

$$\mathcal{O}\left(\frac{\mu}{\varepsilon\delta}\right) DT. \tag{5.14}$$

Since $\mu \ll \varepsilon$ and $\mu/(\varepsilon\delta) \in \mathcal{O}(\varepsilon^4)$, we can place these removed jobs into a container of width D and height εT using Steinberg's algorithm. This container can then later be placed on top of the schedule.

For the vertical jobs, the structural result requires that some jobs are removed before the placement into bands. In addition to the jobs we have the remove when greedily filling the bands, we must also remove some more to guarantee that the jobs will fit. It is to this end that the extra buffers $Б_h$ have been introduced.

We know that these removed jobs can fit inside $9/16$ of Ψ as they have a total width bounded by $9\lambda D/32$. Hence, when guessing the buffers $Б_h$, we guess them such that $\sum_h Б_h \leq 9/32D$. With the extra buffers added and with a correct guess mimicking that of the structural result, the total width available is enough to be able to schedule all vertical jobs. To actually place the vertical jobs, we again greedily fill them into the bands for each height level. For each band, one job of width at most μD may need to be removed. Jobs with a total width of at most $9\lambda D/32$ will also not fit and will therefore be removed. Since the number of bands is bounded by $\mathcal{O}(1/(\varepsilon\delta^2\lambda))$, $\mu \leq c\varepsilon^5\delta$, and the total height of bands is bounded, the total width of the removed jobs is bounded.

For each band, we remove one job for the greedy filling. These jobs fit inside a container of width $\lambda D/8$ inside Ψ. The remaining jobs that do not fit correspond to the jobs that the structural result requires we remove. We have reserved 3 containers inside Ψ for these jobs, with a total width of $9\lambda D/16$ where we again can place them using Steinberg's algorithm. □

We have now placed the large, vertical and horizontal jobs. Next, we place the small jobs. The structural result guarantees that these can be placed inside gaps in the schedule, in addition to potentially adding a container of height εT on top of the schedule. We have proven that the height profile of vertical jobs jumps at most $\mathcal{O}(1/(\varepsilon^4\delta))$ times and gives rise to the same number of boxes where we can place the small jobs. Using NFDH, we can place the small jobs into these boxes. We also use NFDH to potentially place a few small jobs on top of the schedule.

Finally, we place the medium jobs. From the structural result, we know that all medium jobs can be placed either inside Ψ or on top of the schedule whilst adding at most $\mathcal{O}(\varepsilon)$ to the packing height. This only holds as long as the area of the medium jobs is bounded by $\lambda DT/4$, which is enforced by the CALP. To place the medium jobs, we follow the same procedure as for the proof in the structural result.

Consider all jobs $j \in \mathcal{J}_{\text{medium}}$ with $w_j \leq \lambda D/4$. These jobs can all be placed inside the first vertical half Ψ using Steinberg's algorithm. Now consider the remaining medium jobs, i.e. those with a width $w_j > \lambda D/4$. Introduce a container of width D and height $2\lambda T$. Since the medium jobs have an area of at most $\lambda DT/4$ and a width greater than $\lambda D/4$, they must have a maximum height of

$$\frac{\lambda DT/4}{\lambda D/4} = \lambda T. \tag{5.15}$$

The remaining medium jobs can therefore be placed inside this container using Steinberg's algorithm. This container can then be placed on top of the schedule,

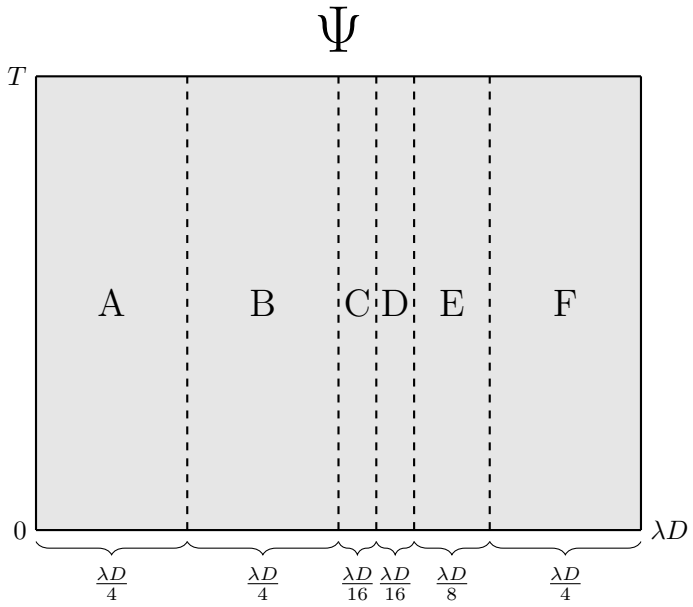


Figure 5.1: The partitioning of the residual item container Ψ. The 6 segments represent the containers used to place some of the jobs that we remove from the packing.

adding $2\lambda T$ to the packing height.

We have used the residual item container for several jobs in a few different cases. Figure 5.1 illustrates the partitioning of Ψ and which jobs we remove to achieve the desired packing. The jobs removed in claim 3.8 are placed in A. The jobs removed from the configuration LP as covered in claim 3.13 are placed in B. The jobs removed when rounding the bands as described in claim 3.12 fit inside C.

In segment D, we place the vertical jobs that are removed when greedily filling the bands. Segment E contains the fractionally assigned vertical jobs. Finally, segment F contains some of the medium jobs.

5.1.1 Placing Ψ into the schedule

The algorithm as described so far returns two packings, σ which a height of $(1 + \mathcal{O}(\varepsilon))T$ containing most jobs and Ψ containing a few jobs inside a container of width λD and height T. For our complete algorithm, we need to place Ψ into the schedule and show that the height of this resulting packing is low enough.

Claim 5.4

If $(\mathcal{J}, D)$ admits a λ-forgiving packing, we can obtain a packing with peak demand $(3/2 + \varepsilon)T$ in polynomial time.

Proof. Let j_Ψ denote an auxiliary job with the same dimensions as our container Ψ, i.e. $w_{j_\Psi} = \lambda D$ and $h_{j_\Psi} = T$. If $(\mathcal{J}, D)$ admits a λ-forgiving packing, then

$$\mathrm{OPT}(\mathcal{J} \cup \{j_\Psi\}) \leq \frac{3}{2}\mathrm{OPT}(\mathcal{J}). \tag{5.16}$$

To place Ψ into the schedule, we can run our algorithm on the set $\mathcal{J} \cup \{j_\Psi\}$ instead. The maximum height of the packing achieved from this will then be bounded by

$$(1 + \mathcal{O}(\varepsilon))\,\mathrm{OPT}(\mathcal{J} \cup \{j_\Psi\}) \leq \left(\frac{3}{2} + \mathcal{O}(\varepsilon)\right)\mathrm{OPT}. \tag{5.17}$$

To obtain our desired packing, we remove j_Ψ from the packing found by our algorithm, which frees up a λD wide slot. All jobs that were assigned to Ψ are then moved into this slot instead. With an appropriate scaled version of ε, we obtain a packing with peak demand $(3/2 + \varepsilon)\mathrm{OPT}$. □

Claim 5.5

The algorithm as described runs in polynomial time with respect to n.

Proof. Whenever guessing is performed, the number of possible guesses is constant. Iterating through all possible bands, stacks and values for μ and δ can therefore be done in constant time. Finding the minimal processing configurations for each bin can be done in linear time w.r.t q, the maximal number of configurations per job. We guess the starting positions and sizes for a constant number of large jobs, which is done in constant time. We also guess the buffer $Б_h$ in constant time.

Next, we run the JALP to assign the jobs to the correct bins. By using Karmakar's algorithm, we can obtain a solution in polynomial time. The solution we obtain has fractionally assigned jobs which we need to handle. We schedule some medium jobs, as well as some fractionally assigned and previously removed jobs using Steinberg's algorithm, which runs in polynomial time. The small jobs are placed using NFDH, which also runs in polynomial time. The algorithm therefore runs in polynomial time. □

5.2 Algorithm for (ε, T)-neat packings

To construct an algorithm for (ε, T)-neat packings, we will make use of theorem 4.3, guaranteeing that any such packing can be restructured into a cool packing. For a given height H, we present an algorithm which either generates a D-feasible packing or concludes that no such packing is possible. If $(\mathcal{J}, D)$ admit a (ε, H)-neat packing, the algorithm is able to construct such a packing.

The structural result proven shows that for a specific choice of processing configurations, there will exist a packing which can be restructured such that it is cool.

We will now make use of this structure to create an algorithm which finds a feasible packing.

To start, there are several components of the structure which can be guessed. We know that for each $k = 1, \ldots, \log(1/\delta)$, almost all flat jobs fit inside at most $1/\varepsilon'$ containers, picked from a set $\mathcal{F}_k$ with $|\mathcal{F}_k| \in \mathcal{O}_n(1)$. We do not know which widths we use after the geometric grouping. We therefore start by guessing at most $\log(1/\delta)/\varepsilon'$ widths from the total set of at most nq options, which can be done in $\binom{nq}{\mathcal{O}(1/\delta)/\varepsilon} \leq n^{\mathcal{O}(\varepsilon)}$ steps. From this, we also calculate the set of reduced starting and ending positions. Since a big job has a width of at least δD and a height of at least μT_{LB}, there can be at most

$$\frac{DT}{\delta D \cdot \mu T_{LB}} \leq \frac{2T}{\delta \mu T} = \frac{2}{\delta \mu} \tag{5.18}$$

big jobs. Both the widths and heights of big jobs are rounded as well. To place big jobs, we therefore simply guess the ending position, width and height of at most $2/(\delta\mu)$ big jobs. We guess the $1/\varepsilon'$ containers from $\mathcal{F}_k$ as well for each $k = 1, \ldots, \log(1/\delta)$.

There are no restrictions imposed on the width of tall jobs, with any width between 0 and D being permissible. If we were to only consider a subset of tall jobs with a minimum width however, the total number of such jobs could be bounded by some constant similarly to what we do for the big jobs. To make use of this observation, we split the tall jobs into two sets depending on whether or not a job has a width $w_j \leq \varepsilon'\delta^2 D/18$. The number of tall jobs with a width $w_j > \varepsilon'\delta^2 D/18$ is bounded by

$$\frac{18 \cdot DT}{\varepsilon'\delta^2 D \cdot T/2} = \frac{36}{\varepsilon'\delta^2}. \tag{5.19}$$

Just like the big jobs, we can guess their rounded heights to multiples $i\varepsilon\delta T$. We can also pick at most $36/(\varepsilon'\delta^2)$ widths from the nq possible choices, which can be done in polynomial time. We do not need to guess the starting times however, as the placement requirements of tall jobs will force the guessed jobs into specific positions, when combined with the narrower tall jobs. For each height level h, let $Б_h$ denote the widths of the guessed tall jobs.

Using the guessed packing of big and flat jobs, we can get an upper bound on the height profile for the tall jobs. We assume that it is possible to obtain a height profile that is δ-translated, i.e. that there exists a gap of at least δD between the height profile and the actual placement of the jobs. An example of the height profile we use after the guesses for flat and big jobs is shown in figure 5.2. From this height profile, we can guess how many tall jobs of a certain width we will have. Since the number of ending points and the widths of both big and flat jobs are bounded, the number of starting points will also be bounded. For each possible height $h \in \mathcal{H}$ of tall jobs, we guess a width ω_h within which the tall jobs must be placed. The guessed width does not have to directly correspond to the height profile obtained from the flat and big jobs, we may e.g. not want to place any jobs of height T even if the current guess for flat and big jobs would allow it to be done. We have one more quantity

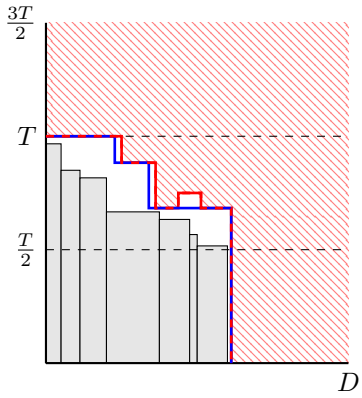


Figure 5.2: An example of the packing structure after the guessed big jobs and containers for flat jobs. The red hatched area contains big and flat jobs. The red line denotes the height profile after the guess. The blue line denotes the δ-translated height profile which we can use as a bound when guessing the placement of tall jobs. Below the guessed height profile, we want to place tall jobs in a staircase shape inside the guessed widths ω_h.

that we want to guess. Whilst we want to assign squeezable jobs using the JALP, we cannot do so fully as we would not be able to bound the area of fractionally assigned squeezable jobs. Instead, we guess some squeezable jobs ahead of time. Consider squeezable jobs with a width $w_j > \varepsilon'\delta^2 D/18$ and height $h_j > \delta^2 T/18$. The number of such jobs is bounded by

$$\frac{18^2 \cdot DT}{\varepsilon'\delta^2 D \cdot \delta^2 T} \leq \mathcal{O}\left(\frac{1}{\varepsilon'\delta^4}\right). \tag{5.20}$$

Again, we guess at most $\binom{nq}{\mathcal{O}(1/(\varepsilon'\delta^4))}$ widths and heights to use for these squeezable jobs, and then guess which widths and heights to use. We still have some narrow and short squeezable jobs left to place but those will be handled by the JALP. Figure 5.3 shows an overview of our job partitioning and which jobs we guess.

To determine the optimal choice of processing configurations for the remaining jobs, we yet again consider a variant of the job assignment LP (JALP) from definition 5.1. To reduce the number of configurations we need to consider, it is enough to only look at one possible configuration for each type of job. As we have already guessed the big jobs, the assignment will be done for tall, flat and squeezable jobs. For tall jobs, we introduce one bin for each allowed rounded height. For flat jobs, we introduce one bin for each rounded width. Let $\mathcal{H}$ and $\mathcal{W}$ denote the sets of rounded heights and widths for tall and flat jobs respectively.

Rather than considering all possible configurations that would lead to a job being assigned to a specific class of job, it is enough to consider only one per class. For squeezeable jobs, we pick the pair (w_{ij}, h_{ij}) which minimizes the jobs area. Let w_j^{squeeze} and h_j^{squeeze} denote the width and height of the job associated with the "smallest" processing configuration such that the job is squeezable. For tall and flat

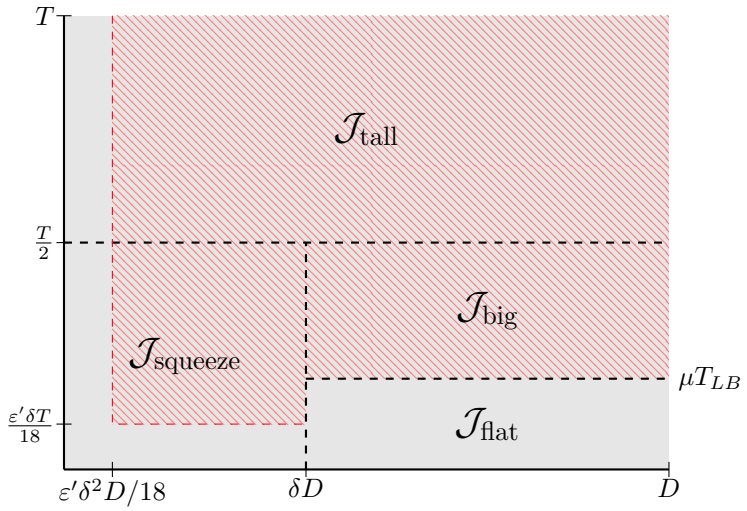


Figure 5.3: An overview of the job partitioning for (ε, T)-neat packings. The jobs in the red hatched area are guessed, as they have a minimum area and thus a bounded amount. The remaining jobs are assigned using the JALP.

jobs, we do something similar by only considering the narrowest tall job and the shortest flat job.

$$(w_j^h, h_j^h) = \underset{\substack{(w_{ij},h_{ij})\in C_j \\ w_{ij}<\delta D, T/2<h_{ij}\leq h}}{\arg\min} \quad w_{ij} \qquad \forall h \in \mathcal{H} \tag{5.21}$$

$$(w_j^w, h_j^w) = \underset{\substack{(w_{ij},h_{ij})\in C_j \\ w_{ij}\leq w, h_{ij}<\mu T_{LB}}}{\arg\min} \quad h_{ij} \qquad \forall w \in \mathcal{W} \tag{5.22}$$

$$(w_j^{\text{squeeze}}, h_j^{\text{squeeze}}) = \underset{\substack{(w_{ij},h_{ij})\in C_j \\ w_{ij}\leq \delta D, h_{ij}\leq T/2}}{\arg\min} \quad w_{ij}h_{ij} \tag{5.23}$$

We are now ready to introduce the JALP for (ε, T)-forgiving packings. Let β denote a bin and let $\mathcal{B}$ denote the set of bins. Let $\mathcal{K}_w$ denote the set of containers components of width w. As we have guessed $1/\varepsilon'$ containers, it is clear that $|\mathcal{K}_w| \leq 1/\varepsilon'$. The JALP is then defined as the following linear program.

$$\sum_{\beta\in\mathcal{B}} x_{j,\beta} = 1, \qquad \forall j \in \mathcal{J} \tag{5.24}$$

$$\sum_{j\in\mathcal{J}} x_{j,h} w_j^h \leq \omega_h - \text{Б}_h \qquad \forall h \in \mathcal{H} \tag{5.25}$$

$$\sum_{j\in\mathcal{J}} x_{j,w} h_j^w \leq \sum_{k\in\mathcal{K}_w} h_k \qquad \forall w \in \mathcal{W} \tag{5.26}$$

$$\sum_{j\in\mathcal{J}} x_{j,\beta_{\text{squeeze}}} h_j^{\text{squeeze}} w_j^{\text{squeeze}} \leq DT - A - B \tag{5.27}$$

$$A = \sum_{j\in\mathcal{J}} \left(\sum_{h\in\mathcal{H}} x_j^h w_j^h h_j^h + \sum_{w\in\mathcal{W}} x_j^w w_j^w h_j^w \right) \tag{5.28}$$

$$B = \text{total area of guessed tall, big and squeezable jobs} \tag{5.29}$$

Equation (5.24) ensures that we do not assign a job to more than one bin, although the job may be fractionally assigned across multiple. Equation (5.25) guarantees that for each rounded height $h \in \mathcal{H}$, we do not assign more jobs as tall with height at most h that the guessed maximal width for said height, including the width of the already guessed wide tall jobs Б_h. Similarly for the flat jobs, equation (5.26) ensures that we do not assign too many flat jobs of width at most w. To be able to squeeze the squeezable jobs in at the end, equation (5.29) ensures that the total area of squeezable jobs is sufficiently small.

As before, we need to relax the integrality of the problem to be able to find a solution, i.e. allowing for fractionally assigned jobs. The LP is solved by a standard algorithm such as Karmakar's [Kar84]. If the LP cannot be solved, our current guess is wrong and we go back one guess at a time until we can obtain a solution to the LP.

Claim 5.6

For a correct guess of the packing structure and a sufficiently large T, the JALP will have a solution

Proof. Before running the JALP, the algorithm guesses multiple components of the packings structure, such as the placement of big jobs, the containers containing flat jobs and which wide tall and large squeezable jobs are included. For at least one guess, there will exist a guessed packing which mimics the structure seen in the structural result, when considering the jobs packed so far. The containers for the tall jobs are already guessed and alongside the big jobs, they define a guessed height profile. The exact placement of the guessed tall and squeezable jobs is not specified by our guess.

From the structural result, we know by assumption that the height profile of tall jobs always can be shifted such that it is δ-translated. We also know that it is possible to place all tall jobs in a staircase shape in order of non-increasing height. The δ-translated height profile can be estimated by using the height profile of big and flat jobs, which we know can be guessed in a bounded number of steps. When guessing the space ω_h for each height level, at least one appropriate guess will mimic the structure of the packing achieved from the structural result. The constraint in equation (5.25) can thus be satisifed.

For the flat jobs, we can make a similar argument. We have shown in section 4.2 that there must exist a packing where almost all flat jobs are placed inside at most $\log(1/\delta) \cdot 1/\varepsilon'$ containers. These containers have rounded widths and a total stacked height which are covered by the guessing step earlier. At least one guess of the starting position, width and height of the containers will therefore mimic the packing from the structural result. Equation (5.26) guarantees that for each rounded width, we do not assign more jobs than the available width in the guessed containers. We know that there exists a solution which satisfies this constraint thanks to our

structural result.

For the squeezable jobs, we know that they always can be squeezed into the packing as long as the total height is sufficiently tall. As per the requirements from lemma 4.7, the squeezing algorithm works if

$$T \geq \max\left\{\frac{\text{area}(\mathcal{J})}{D}, \text{OPT}\right\} \iff \text{area}(\mathcal{J}) \leq DT. \tag{5.30}$$

In order for the squeezing algorithm to work, we need to ensure that the height of the total area of jobs is sufficiently small for our current T. With all other jobs either assigned or guessed, we simply require that we assign few enough squeezable jobs that the total area is at most DT. We know from the structural result that such a packing will exist, and thus constraint (5.29) will hold.

We know that a packing like that shown to exist in the structural result in chapter 4 satisfies all the constraints in the JALP, if T is sufficiently tall. The LP will then be solvable. □

Claim 5.7

If a solution to the JALP is found, all jobs can either be placed into the schedule or inside a container on top, whilst adding at most $3\varepsilon' T$ height. If the assumption regarding δ-translated packings does not require any additional height, the total height of the obtained packing is $(3/2 + \varepsilon)T$

Proof. To verify that the jobs can be placed, we must both consider jobs that are fractionally assigned by the LP and flat jobs which overlap the borders of their guessed containers.

For the flat jobs, the LP guarantees that the total height of all jobs assigned as flat of a particular width w is less than or equal to the total height of corresponding components of the guessed containers. We have at most $1/\varepsilon'$ containers, each with $\log(1/\delta)/\varepsilon'$ widths. As we know the height of the stack with precision μT and flat jobs have a height of at most μT_{LB}, greedily filling the flat jobs into the containers ensures that we only need to remove at most one job per container segment. Hence, greedily filling the jobs placed them all except some flat jobs with a total area of at most

$$\frac{1}{\varepsilon'}\frac{\log(1/\delta)}{\varepsilon'}\mu DT \leq \varepsilon' DT. \tag{5.31}$$

Hence, these jobs fit inside a container above the schedule with height $2\varepsilon' T$ using Steinberg's algorithm.

The tall jobs can be greedily placed according to their assigned heights without removing any jobs. We still have to consider jobs that are fractionally assigned by the JALP however. As the JALP has $n + \mathcal{O}\left(\frac{1}{\varepsilon'\delta}\right) + \mathcal{O}\left(\frac{\log(1/\delta)}{\varepsilon'}\right) + 1 \leq \mathcal{O}\left(\frac{1}{\varepsilon'\delta}\right)$ constraints, there will be at most $\frac{1}{\varepsilon'\delta}$ fractionally assigned jobs.

If a job is fractionally assigned as flat, their total area will be bounded by

$$\frac{1}{\varepsilon'\delta}\mu DT \leq \varepsilon' DT \tag{5.32}$$

and we can thus simply place these jobs on top of the schedule using Steinberg's algorithm. The tall jobs are more problematic, as we no longer have a residual container to place the jobs inside. To account for fractional tall jobs, we make use of our assumption that it is possible to obtain a δ-translated height profile of the tall jobs. The only tall jobs that are assigned using the JALP have a maximal width of $w_j \leq \varepsilon\delta^2 D/18$. The total width of fractionally assigned tall jobs can thus be bounded by

$$\frac{1}{\varepsilon'\delta}\frac{\varepsilon'\delta^2 D}{18} \leq \delta D. \tag{5.33}$$

As long as there are tall jobs either guessed or assigned to at least two distinct height levels, there will exist a gap of width δD and height h where we can place the fractionally assigned jobs. We can fit all these jobs inside the tallest such gap even if the jobs may be fractionally assigned to many different height levels. If we only have a single height level for tall jobs, any fractionally assigned tall jobs must also be fractionally assigned as something else which is not tall. We can in this case simply assign the job fully to another category.

Finally, a job could be squeezable. For a feasible solution to the JALP, the total area of both guessed and assigned squeezable jobs is small enough that we can use lemma 4.7. The only squeezable jobs we assign using the JALP either has a height $h_j \leq \delta^2 T/18$ and width $w_j \leq \delta D$ or has a height $h_j \leq T/2$ and a width $w_j \leq \varepsilon'\delta^2 D/18$. Thus, any such job has a maximum area of $\frac{\varepsilon'\delta^2}{18}DT$. The total area of fractionally assigned squeezable jobs is then bounded by

$$\frac{1}{\varepsilon'\delta}\cdot\frac{\varepsilon'\delta^2}{18}DT \leq \frac{\delta DT}{18} \leq \frac{\varepsilon}{18}DT \leq \varepsilon' DT. \tag{5.34}$$

Hence, these jobs can be squeezed into the packing using lemma 4.7 by adding at most ε' to the packing height. Thus, all jobs have either been placed into the schedule or in a container on top.

For the structural result to hold, we need to increase the height of the packing by $15\varepsilon' T$. In order to place all jobs using our algorithm, the packing height further needs to be increased by a $3\varepsilon' T$. As $\varepsilon' = \frac{\varepsilon}{18}$, it is clear that the packing found by our algorithm has a peak demand of

$$\left(1 + \frac{15+3}{18}\varepsilon\right)T = (1+\varepsilon)\,T. \tag{5.35}$$

□

Claim 5.8

The algorithm as described runs in polynomial time with respect to n.

Proof. We start by performing some guesses. Guessing which widths to use for flat jobs after the geometric grouping, the widths for wide tall jobs and the widths and heights for large squeezable jobs can be done in polynomial time. The tall jobs have a constant number of rounded heights to guess from and the number of, and ending positions of containers of flat jobs can are constant.

To place the remaining jobs, we again make use of a variant of the JALP. We first find the best processing configurations for each bin, in constant time. The JALP then assigns the jobs into different categories and can be solved in polynomial time using Karmakar's algorithm. After the jobs are assigned and placed, a few jobs either have to be removed or are fractionally assigned. For flat jobs, these can be placed using Steinberg's algorithm which runs in polynomial time. For squeezable jobs, we place them using the squeezing algorithm in polynomial time. The algorithm will therefore run in polynomial time with respect to the number of jobs n. □

5.3 Complete algorithm

For both λ-forgiving and (ε, T)-neat packings, we have constructed algorithms that for a given set of jobs $\mathcal{J}$ and a height T either find a feasible packing or conclude that no such packing can exist. We are now ready to combine these algorithms to obtain an algorithm for DSP with moldable jobs.

In order to obtain an optimal packing, we perform binary search with respect to the height T. To start the binary search, we need an initial upper and lower bound on the height T. The smallest height possible would occur in the case where all jobs perfectly occupy all space leaving no empty gaps, when each job has a processing configuration which minimizes the area. If there exists a job that is taller than this height however, we can immediately know that the minimum packing height will need to be at least the height of this item. Hence, a lower bound for T can be expressed as

$$T_{LB} = \max\left\{\min_{C_j}\left\{\frac{\text{area}(\mathcal{J})}{D}\right\}, \max_{j\in\mathcal{J}}\left\{\min_{C_j} h_j\right\}\right\}. \tag{5.36}$$

To obtain an upper bound, we can yet again make use of Steinberg's algorithm. Consider a box of width D and height $T_{UB} = 2T_{LB}$. Since $T_{LB} \geq \text{area}(\mathcal{J})/D$, it is clear that the container has at least twice the area of any job. It is also clearly at least twice as tall as any job. Therefore, we can ensure that Steinberg's algorithm will be able to generate a feasible packing. It will therefore always be possible to obtain a packing with height T_{UB} using Steinberg's algorithm, and we thus use T_{UB} as an upper bound for our binary search.

We can now start performing the binary search. Keeping T_{UB} and T_{LB} unchanged,

let $\ell = T_{LB}$ and $u = T_{UB}$. In each step of the binary search, we set $T = \frac{u+\ell}{2}$ and run both our algorithms. We know from theorem 2.9 that all packings admit either a λ-forgiving or a $(\varepsilon, \mathrm{OPT})$-neat packing. At least one of the algorithms must therefore yield a feasible packing as long as our T is not too low. If one of the algorithms work, we set $u = T$ and otherwise we set $\ell = T$. We repeat the loop until the upper and lower bounds converge to within a distanc of $T_{LB}\varepsilon/4$. In order for our final obtained packing to have a peak demand of $(3/2+\varepsilon)\mathrm{OPT}$, we need to run our two subalgorithms with scaled versions of ε. Let $\varepsilon^* = \varepsilon/2$. We use this value of ε^* when calculating the λ-forgiving and (ε^*, T)-neat packings. The algorithm in full is shown below. We can now return to proving the main theorem of this thesis.

Algorithm 1 Complete approximation algorithm

Require: $\mathcal{J}, \varepsilon$
  $T_{LB} \leftarrow \max\left\{\min_{C_j}\left\{\frac{\mathrm{area}(\mathcal{J})}{D}\right\}, \max_{j\in\mathcal{J}}\left\{\min_{C_j} h_j\right\}\right\}$.
  $T_{UB} \leftarrow 2T_{LB}$
  $\ell \leftarrow T_{LB}$
  $u \leftarrow T_{UB}$
  $\varepsilon^* \leftarrow \varepsilon/2$
  $\lambda \leftarrow \min\{\frac{\varepsilon^*}{3(5+4\varepsilon^*)}, \frac{1}{60}\}$
  while $u - \ell > T_{LB}\frac{\varepsilon}{4}$ **do**
    ▷ *Binary search* ◁
    $T \leftarrow \frac{u+\ell}{2}$
    $\sigma_{\text{forgiving}} \leftarrow$ Algorithm in 5.1
    $\sigma_{\text{neat}} \leftarrow$ Algorithm in 5.2
    if $\sigma_{\text{forgiving}}$ exists $\vee$ σ_{neat} exists **then**
      $u \leftarrow T$
    else
      $l \leftarrow T$
    end if
  end while
  return $\arg\min\{h(\sigma_{\text{forgiving}}), h(\sigma_{\text{neat}})\}$

Recall theorem 1.2.

> **Theorem 1.2**
>
> If assumption 1.1 holds, there exists an approximation algorithm for the Demand Strip Packing problem with moldable jobs.

Proof. For our algorithm to be an approximation algorithm, it must run in polynomial time with respect to n, the number of jobs. Finding T_{LB} and T_{UB} requires a checking all possible configurations for all jobs and finding the choice which results in the maximum height. Each job has at most q processing configurations and there are n jobs. Finding the maxima/minima can be done in linear time, so the total time complexity for this step can be bounded by $\mathcal{O}(nq)$.

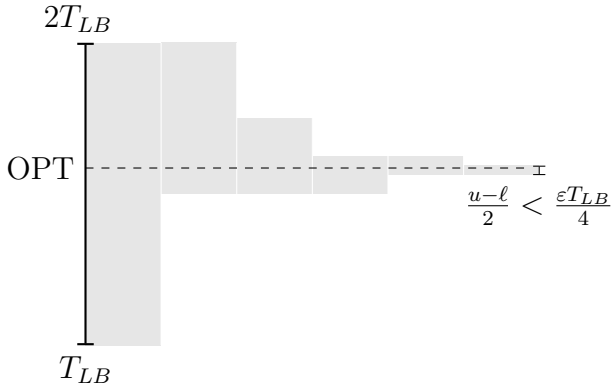


Figure 5.4: An example of how the binary search procedure converges towards OPT. In each iteration, the search space (gray) is halved. The algorithm stops when the search space is sufficiently small, when $\frac{u+\ell}{2} < \frac{\varepsilon}{4} T_{LB}$.

The binary search will converge to a value of T in logarithmic time as our initial search space of width T_{LB} is cut in half during each iteration. Inside each step, we run both our subalgorithms, which we have previously shown in claims 5.5 and 5.8 run in polynomial time. Algorithm 1 therefore runs in polynomial time with respect to n, and is thus an approximation algorithm. □

Corollary

If the assumption regarding δ-translated packings does not require the packing height to be increased, then algorithm 1 is a $(3/2 + \varepsilon)$-approximation algorithm.

Proof. From claims 5.4 and 5.7, we have shown that using ε^*, both our algorithms can find a packing with a peak demand of $(3/2+\varepsilon^*)T$ if $(\mathcal{J}, D)$ admits a λ-forgiving or (ε^*, T)-neat packing respectively.

When the algorithm has converged, we know that we are able to find a packing σ with height T but not one with height $T - \frac{\varepsilon}{4} T_{LB}$. From this, we can deduce that

$$\left(\frac{3}{2} + \varepsilon^*\right)\left(T - \frac{\varepsilon}{4} T_{LB}\right) < \left(\frac{3}{2} + \varepsilon^*\right) \mathrm{OPT} \implies T < \mathrm{OPT} + \frac{\varepsilon}{4} T_{LB}. \tag{5.37}$$

As $T_{LB} \leq \mathrm{OPT}$, we can see that

$$\begin{gathered} h(\sigma) = \left(\frac{3}{2} + \varepsilon^*\right) T \leq \left(\frac{3}{2} + \varepsilon^*\right)\left(\mathrm{OPT} + \frac{\varepsilon}{4} T_{LB}\right) \leq \left(\frac{3}{2} + \varepsilon^*\right)\left(1 + \frac{\varepsilon}{4}\right) \mathrm{OPT} \\ \left(\frac{3}{2} + \varepsilon^* + \frac{3}{8}\varepsilon + \frac{1}{4}\varepsilon\varepsilon^*\right) \mathrm{OPT} \leq \left(\frac{3}{2} + \varepsilon\left(\frac{1}{2} + \frac{3}{8} + \frac{\varepsilon}{2}\right)\varepsilon\right) \mathrm{OPT} \leq \left(\frac{3}{2} + \varepsilon\right) \mathrm{OPT}. \end{gathered} \tag{5.38}$$

The height of the packing obtained by the algorithm will therefore be at most $\left(\frac{3}{2} + \varepsilon\right) \mathrm{OPT}$. □

6

Conclusions and future work

In this thesis, we present an algorithm for the demand strip packing problem with moldable jobs. The algorithm is runs in polynomial time with respect to n, the number of jobs and is thus a valid approximation algorithm. The algorithm's approximation ratio is believed to be $\alpha = (3/2+\varepsilon)$, so long as the assumption regarding δ-translated packings holds.

Due to the assumption made, the algorithm as currently stated is not guaranteed to work. Proving this assumption would require techniques that significantly differ from the work in this thesis and it was thus not feasible to finish within the frame of this thesis. We do however strongly believe that the assumption, or another similar statement will hold. It is possible that the total height of the packing might need to be slightly increased to allow for this, and thus we would end up with a slightly worse approximation ratio than $(3/2 + \varepsilon)$.

In terms of future work in this area, there are multiple possible directions to take. Proving the assumption is the main priority to obtain a functioning algorithm. Once the assumption is proven, adjusting the algorithm such that it can model moldable jobs in a more useful way could be considered. One idea is to allow the processing configuration of jobs to be determined by a function, rather than each job j having a finite set C_j.

Bibliography

[Ala+13] Soroush Alamdari et al. "Smart-Grid Electricity Allocation via Strip Packing with Slicing". In: *Algorithms and Data Structures*. Ed. by Frank Dehne, Roberto Solis-Oba, and Jörg-Rüdiger Sack. Red. by David Hutchison et al. Vol. 8037. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 25–36. ISBN: 978-3-642-40103-9 978-3-642-40104-6. DOI: 10.1007/978-3-642-40104-6_3. URL: http://link.springer.com/10.1007/978-3-642-40104-6_3 (visited on 01/29/2026).

[BR26] William Bergquist and Malin Rau. *A new structural result for the Demand Strip Packing problem with moldable jobs*. Individual Project Report. Jan. 15, 2026.

[Chr+16] H. Christensen et al. "Multidimensional Bin Packing and Other Related Problems : A Survey". In: 2016. URL: https://www.semanticscholar.org/paper/Multidimensional-Bin-Packing-and-Other-Related-%3A-A-Christensen-Khan/bbcf4ca2524cd50fdb03b180aa5f98d2daa759ce (visited on 02/10/2026).

[Cla] Clay Mathematics Institute. *The Millennium Prize Problems*. Clay Mathematics Institute. URL: https://www.claymath.org/millennium-problems/ (visited on 02/16/2026).

[Cof+80] E. G. Coffman Jr. et al. "Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms". In: *SIAM Journal on Computing* 9.4 (Nov. 1980), pp. 808–826. ISSN: 0097-5397, 1095-7111. DOI: 10.1137/0209062. URL: http://epubs.siam.org/doi/10.1137/0209062 (visited on 02/10/2026).

[Dep+23] Max A. Deppert et al. "Peak Demand Minimization via Sliced Strip Packing". In: *Algorithmica* 85.12 (Dec. 1, 2023), pp. 3649–3679. ISSN: 1432-0541. DOI: 10.1007/s00453-023-01152-w. URL: https://doi.org/10.1007/s00453-023-01152-w (visited on 02/02/2026).

[DMT04] Pierre-Francois Dutot, Grégory Mounié, and Denis Trystram. "Scheduling Parallel Tasks: Approximation Algorithms". In: *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Ed. by Joseph T. Leung. chapter 26. CRC Press, 2004, pp. 26-1 - 26–24. URL: https://hal.science/hal-00003126.

[Ebe+25] Franziska Eberle et al. "A Tight (3/2 + ε)-Approximation Algorithm for Demand Strip Packing". In: *Proceedings of the 2025 Annual ACM-*

SIAM Symposium on Discrete Algorithms (SODA). Proceedings. Society for Industrial and Applied Mathematics, Jan. 2025, pp. 641–699. DOI: 10.1137/1.9781611978322.20. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611978322.20 (visited on 01/22/2026).

[Gál+21] Waldo Gálvez et al. *Approximation Algorithms for Demand Strip Packing*. May 19, 2021. DOI: 10.48550/arXiv.2105.08577. arXiv: 2105.08577[cs]. URL: http://arxiv.org/abs/2105.08577 (visited on 01/22/2026).

[Har+14] Rolf Harren et al. "A $(5/3 + \varepsilon)$-approximation for strip packing". In: *Computational Geometry* 47.2 (Feb. 2014), pp. 248–267. ISSN: 09257721. DOI: 10.1016/j.comgeo.2013.08.008. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925772113001016 (visited on 02/02/2026).

[HLC09] Chung-Yang (Ric) Huang, Chao-Yue Lai, and Kwang-Ting (Tim) Cheng. "Fundamentals of algorithms". In: *Electronic Design Automation*. Elsevier, 2009, pp. 173–234. ISBN: 978-0-12-374364-0. DOI: 10.1016/B978-0-12-374364-0.50011-4. URL: https://linkinghub.elsevier.com/retrieve/pii/B9780123743640500114 (visited on 01/29/2026).

[Jan12] Klaus Jansen. "A$(3/2+\varepsilon)$ approximation algorithm for scheduling moldable and non-moldable parallel tasks". In: *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*. SPAA '12. New York, NY, USA: Association for Computing Machinery, June 25, 2012, pp. 224–235. ISBN: 978-1-4503-1213-4. DOI: 10.1145/2312005.2312048. URL: https://dl.acm.org/doi/10.1145/2312005.2312048 (visited on 02/02/2026).

[JL18] Klaus Jansen and Felix Land. "Scheduling Monotone Moldable Jobs in Linear Time". In: *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS). ISSN: 1530-2075. May 2018, pp. 172–181. DOI: 10.1109/IPDPS.2018.00027. URL: https://ieeexplore.ieee.org/document/8425171/ (visited on 02/04/2026).

[JMR19] 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. ISSN: 1549-6325. DOI: 10.1145/3302250. URL: https://dl.acm.org/doi/10.1145/3302250 (visited on 01/23/2026).

[JR19] Klaus Jansen and Malin Rau. *Closing the gap for pseudo-polynomial strip packing*. Feb. 6, 2019. DOI: 10.48550/arXiv.1712.04922. arXiv: 1712.04922[cs]. URL: http://arxiv.org/abs/1712.04922 (visited on 02/04/2026).

[JR21] Klaus Jansen and Malin Rau. "Closing the Gap for Single Resource Constraint Scheduling". In: *29th Annual European Symposium on Algorithms (ESA 2021)*. Ed. by Petra Mutzel, Rasmus Pagh, and Grzegorz Herman. Vol. 204. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 53:1–53:15. ISBN: 978-3-95977-204-4. DOI: 10.4230/

LIPIcs.ESA.2021.53. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ESA.2021.53 (visited on 01/23/2026).

[Kar84] N. Karmarkar. "A new polynomial-time algorithm for linear programming". In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing - STOC '84* (1984). Conference Name: the sixteenth annual ACM symposium ISBN: 9780897911337, pp. 302–311. DOI: 10.1145/800057.808695. URL: http://portal.acm.org/citation.cfm?doid=800057.808695 (visited on 03/11/2026).

[KK82] 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 (sfcs 1982)* (Nov. 1982). Conference Name: 23rd Annual Symposium on Foundations of Computer Science, pp. 312–320. DOI: 10.1109/SFCS.1982.61. URL: http://ieeexplore.ieee.org/document/4568405/ (visited on 02/23/2026).

[KKR25] Debajyoti Kar, Arindam Khan, and Malin Rau. *Improved Approximation Algorithms for Three-Dimensional Bin Packing*. Apr. 19, 2025. DOI: 10.48550/arXiv.2503.08863. arXiv: 2503.08863[cs]. URL: http://arxiv.org/abs/2503.08863 (visited on 04/28/2026).

[Neb24] Abraham Hizkiel Nebey. "Recent advancement in demand side energy management system for optimal energy utilization". In: *Energy Reports* 11 (June 2024), pp. 5422–5435. ISSN: 23524847. DOI: 10.1016/j.egyr.2024.05.028. URL: https://linkinghub.elsevier.com/retrieve/pii/S2352484724003093 (visited on 01/29/2026).

[Rau20] Malin Rau. "Useful Structures and How to Find Them: Hardness and Approximation Results for Various Variants of the Parallel Task Scheduling Problem". Accepted: 2019-05-24. PhD thesis. Christian-Albrechts-Universität zu Kiel, Jan. 10, 2020. URL: https://macau.uni-kiel.de/receive/macau_mods_00000126 (visited on 02/23/2026).

[Ski08] Steven S. Skiena. "Algorithm Analysis". In: *The Algorithm Design Manual*. Ed. by Steven S. Skiena. London: Springer, 2008, pp. 31–64. ISBN: 978-1-84800-070-4. DOI: 10.1007/978-1-84800-070-4_2. URL: https://doi.org/10.1007/978-1-84800-070-4_2 (visited on 02/16/2026).

[Ste97] A. Steinberg. "A Strip-Packing Algorithm with Absolute Performance Bound 2". In: *SIAM Journal on Computing* 26.2 (Mar. 1997), pp. 401–409. ISSN: 0097-5397. DOI: 10.1137/S0097539793255801. URL: https://epubs.siam.org/doi/10.1137/S0097539793255801.

[Svi12] Maxim Sviridenko. "A note on the Kenyon–Remila strip-packing algorithm". In: *Information Processing Letters* 112.1 (Jan. 2012), pp. 10–12. ISSN: 00200190. DOI: 10.1016/j.ipl.2011.10.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S0020019011002742 (visited on 02/23/2026).

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. 1st ed. Cambridge University Press, Apr. 26, 2011. ISBN: 978-0-511-92173-5. DOI: 10.1017/CBO9780511921735. URL: https:

//www.cambridge.org/core/product/identifier/9780511921735/type/book (visited on 01/20/2026).

[Yaw+14] Sean Yaw et al. "Peak demand scheduling in the Smart Grid". In: *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. 2014, pp. 770–775. DOI: 10.1109/SmartGridComm.2014.7007741.

A

Computational complexity

In this chapter, we present a basic theoretical overview of the field of computational complexity. For our applications, the most interesting topics are complexity classes and the famous open problem P = NP. For a more complete and rigorous background, see other sources such as [Ski08] or [HLC09].

When analyzing the complexity of an algorithm, we are interested in how the time needed for the algorithm to finish scales as the size of the algorithm's input increases. More specifically, we care about the maximum time that might be needed for the algorithm to finish for a specific input size n. To illustrate this, consider algorithm 2 below.

Algorithm 2 Insertion sort

Require: A : Array of objects $i \in \mathbb{N}$, $|A| = n$
  $i \leftarrow 1$
  while $i \leq n$ **do**
    $i \leftarrow j$
    while $j > 0$ & $A[j-1] > A[j]$ **do**
      ▷ *Swap* $A[j]$ *and* $A[j-1]$ ◁
      $x \leftarrow A[j]$
      $A[j] \leftarrow A[j-1]$
      $A[j-1] \leftarrow x$
    end while
    $i \leftarrow i + 1$
  end whilereturn A

The insertion sort algorithm sorts a list A of items $i \in \mathbb{N}$. It does this by inserting one item at a time into its correct position. Starting with a single sorted item, it compares each item to all the sorted items to its left. Each item is moved one step left at a time until it is in the correct position. To analyze the algorithm's time complexity, we need to determine the maximum number of comparisons that need to be done in the worst case scenario. If we're lucky, item k is already in the correct position and we only need to make one comparison. It is however possible that item k is the smallest item in the list and needs to be compared with all previous sorted items. If we use a list that is sorted in reverse order as input to the algorithm, we get

the worst case scenario. Each of the n items must be compared against all previous items. The number of comparisons becomes

$$\sum_{i=1}^{n} i - 1 = \frac{n \cdot (n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}. \tag{A.1}$$

A.1 Asymptotic behavior

Rather than looking at the exact number of steps the algorithm takes, it is usually enough to look at the algorithms asymptotic time complexity. As the size of the input n increases, some terms may dominate the others. In the above example, the n^2 term will dominate the n term as $n \to \infty$. We formalize this using $\mathcal{O}$-notation.

Definition A.1

A nonnegative function $f(x)$ belongs to the set $\mathcal{O}(g(x))$ if there exists a positive constant c such that $f(x) \leq cg(x)$ for all x larger than some sufficiently value x'.

In the above example, one can say that $f(n) = n^2/2 - n/2 \in \mathcal{O}(n^2)$. Since $n^3 \geq n^2 \ \forall n \geq 1$, it is also the case that $f(x) \in \mathcal{O}(n^3)$. The same will hold for any polynomial of degree larger than 2. If we however consider a function such as $f(x) = 2^x$, it is no longer possible to find a suitable constant c such that $2^x \leq cx^p$ as $x \to \infty$. In a function with both a polynomial term and an exponential term, the exponential would dominate the expression. A comparison of a few common functions and how they dominate each other is shown in equation A.2.

$$\mathcal{O}(1) \subseteq \mathcal{O}(\log n) \subseteq \mathcal{O}(n^p | p \geq 1) \subseteq \mathcal{O}(c^n | c > 1) \subseteq \mathcal{O}(n!) \tag{A.2}$$

In order to analyze the complexity of more complicated functions, we make use of the following rules.

$$\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(\max(f(n), g(n)) \tag{A.3}$$
$$\mathcal{O}(f(n)) \cdot \mathcal{O}(g(n)) = \mathcal{O}(f(n) \cdot g(n)) \tag{A.4}$$
$$\mathcal{O}(c \cdot f(x)) = \mathcal{O}(f(x)) \tag{A.5}$$

Using these rules, we can compare other functions built out of several components. A more detailed overview of $\mathcal{O}$-notation, as well as the closely related Ω-notation is available at [Ski08], among many others.

In a multivariate case, it is important to distinguish the time complexity with respect to the different variables. Sometimes, we may only care about the time complexity with respect to a single variable. We denote the variable of interest in the subscript. For example, A function belonging to the set $\mathcal{O}_\varepsilon(1)$ is constant with respect to ε (no dependency on ε), but may have higher time complexities for other variables.

A.2 Complexity classes

When comparing optimization problems, it is common to categorize them using complexity classes.

Definition A.2

A *complexity class* is a set of problems where algorithms exist that can solve the problem in the class's time complexity.

For each time complexity, we can introduce a complexity class. The complexity class P for instance contains all problems where there exists a polynomial-time algorithm. Since a logarithmic time algorithm also runs in polynomial time (as we see in equation A.2), the class L of logarithmic type problems must be a subset of P. In order to cover all relevant complexity classes, we must introduce the concepts of determinstc and nondeterministic computers/Turing machines.

Definition A.3

A *deterministic computer* is a computer which solves problems using deterministic algorithms. For a given algorithm and input, the result of any algorithm can be uniquely determined.

Definition A.4

A *nondeterministic computer* is a computer which is not deterministic. At any point in an algorithm, there are several possible next steps from which the computer will make a nondeterministic choice.

All "real" computers that we're familiar with are deterministic computers since they always make a deterministic choice with the inputs they're given [HLC09]. When trying to employ randomness in algorithms, we have to resort to pseudo-random numbers which are still deterministic given a set seed. A nondeterministic computer is only conceptual but could theoretically solve harder problems by checking all possible options and picking the best solution of those obtained.

When defining complexity classes, where define classes for both deterministic and nondeterministic computers. The class P as defined earlier actually contains all problems that can be solved in polynomial time using a deterministic computer. For nondeterministic computers, we can similarly introduce the class NP containing all problems that could be solved in polynomial time if we were to use a nondeterministic computer. Since a nondetermenistic computer is more capable than a conventional deterministic one, it is clear that $\text{P} \subseteq \text{NP}$.

Conjecture

P = NP

The proof is left as an exercise to the reader. This famous conjecture states that all NP problems actually lie in P, i.e. that there exists some algorithm to solve them in polynomial time even using a deterministic computer. The conjecture has been considered important enough that is is named as one of the seven millennium price problems, with a one million dollar reward available for anyone who can find a proof [Cla]. It is as of yet unsolved, with no proof or counterexamples.

Another way of viewing the class NPcomes up when comparing algorithms for finding optimal solutions against the process of checking that a given solution is optimal. For problems in NP, we may need a nondeterministic computer to discover the optimal solution in polynomial time. Given a feasible solution however, a deterministic computer is able to verify wether or not the solution is optimal in polynomial time.

Unless P = NP, there must exist some class of problems that belong to NP but not P. These are the hardest problems in the NP class, where the algorithms we've developed only run in polynomial time on a nondeterministic computer. To analyze these problems, we introduce the classes of NP-hard and NP-complete.

Definition A.5

A problem $\mathcal{P}$ belongs to the complexity class NP-hard if every problem in NP can be reduced to $\mathcal{P}$ in polynomial time.

Definition A.6

A problem $\mathcal{P}$ belongs to the complexity class NP-complete if it belongs to both NP-hard and NP.

By reducing a problem, we mean that for arbitrary problems $\mathcal{P}$ and a problem $\mathcal{P}$, it is possible to transform a $\mathcal{P}$ into $\mathcal{P}'$. We may not have an algorithm to solve $\mathcal{P}$ but if we reduce $\mathcal{P}$ to $\mathcal{P}'$ and solve it instead, we can then transform the solution back to obtain a solution to $\mathcal{P}$.

The class NP-complete consists of all problems in NP that can be transformed into. Since any NP problem can be reduced into a problem belonging to NP-complete, it is enough to analyze the class of NP-complete problems. If one could show that just a single problem in NP-complete actually belongs to P, then all problems in NP must also belong to P, and thus P = NP. Naturally, this has not yet been shown.

As NP-complete problems still have to lie in NP, it must be possible to verify if a solution is correct in polynomial time using a deterministic computer. For some problems such as the Demand Strip Packing problem covered in this thesis, this is

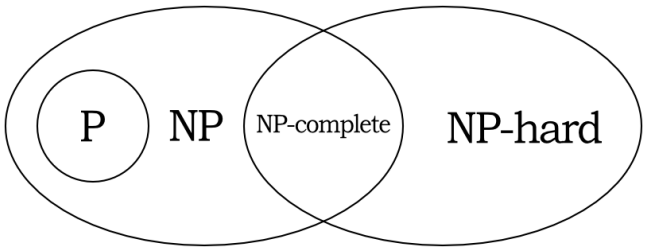


Figure A.1: An Euler diagram comparing the different complexity classes, under the assumption that P $\neq$ NP

.

not the case. These problems belong to the class NP-hard. Often, the only way to verify that a solution is optimal is by iterating over all possibilities, something which cannot be done in polynomial time with a deterministic computer.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY