Approximation Algorithms for Geometric Packing Problems

A THESIS

SUBMITTED FOR THE DEGREE OF

Master of Technology (Research)

IN THE

Faculty of Engineering

BY

Eklavya Sharma

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

July, 2021

Declaration of Originality

I, **Eklavya Sharma**, with SR No. **04-04-00-10-22-19-1-16644** hereby declare that the material presented in the thesis titled

Approximation Algorithms for Geometric Packing Problems

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2019-2021**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: 27 April 2021

Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Arindam Khan

Advisor Name:

Advisor Signature

Acknowledgements

I am very grateful to my advisor, Prof. Arindam Khan, for how much he has cared for my academic development. His guidance on many aspects of my life as a student and researcher has been very valuable. Talking to him has been a great source of motivation and enlightenment for me, and he has been a pillar of support during my tough times.

I am thankful to my parents, for always encouraging my pursuits of knowledge and seconding my decision to pursue graduate studies. Most of my time as a student at IISc has been spent at home in their company (because of the Coronavirus pandemic) and they have been very helpful at ensuring that I get a suitable environment for study and research.

I'm grateful to the Indian Institute of Science (IISc) and its Computer Science and Automation (CSA) department for its wonderful M.Tech. Research programme, which has been a gateway to research for me.

I'm grateful to Prof. Siddharth Barman, for giving me helpful suggestions at crucial stages of my life at IISc. I'm grateful to Prof. Arindam Khan, Prof. Anand Louis, Prof. Chiranjib Bhattacharyya, Prof. Arpita Patra, and Prof. Sanjit Chatterjee for teaching courses at CSA that I really enjoyed and learned a lot from. I thank all the other professors at CSA who I got to meet and learn from.

Finally, I'm thankful to all my friends at IISc for the memorable time I spent with them at the IISc campus (and remotely afterwards, because of the Coronavirus pandemic). K.V.N. Sreenivas, in addition to being a good friend, has been a very good research partner and coauthor. Rameesh Paul has been a great source of intellectual stimulation, especially during our collaborations on course projects. I'm grateful for the many interesting academic (and non-academic) discussions with my friends Swati Allabadi, Arka Ray, Janaky Murthy and many others.

Abstract

We study approximation algorithms for the geometric bin packing problem and its variants. In the two-dimensional geometric bin packing problem (2D GBP), we are given n rectangular items and we have to compute an axis-parallel non-overlapping packing of the items into the minimum number of square bins of side length 1. 2D GBP is an important problem in computer science and operations research arising in logistics, resource allocation, and scheduling.

We first study an extension of 2D GBP called the *generalized multidimensional bin packing* problem (GVBP). Here each item i additionally has d nonnegative weights $v_1(i), v_2(i), \ldots, v_d(i)$ associated with it. Our goal is to compute an axis-parallel non-overlapping packing of the items into bins so that for all $j \in [d]$, the sum of the j^{th} weight of items in each bin is at most 1. Despite being well studied in practice, surprisingly, approximation algorithms for this problem have rarely been explored. We first obtain two simple algorithms for GVBP having asymptotic approximation ratios (AARs) $6(d+1)$ and $3(1+\ln(d+1)+\varepsilon)$. We then extend the Round-and-Approx (R&A) framework [14] to wider classes of algorithms, and show how it can be adapted to GVBP. Using more sophisticated techniques, we obtain algorithms for GVBP having an AAR of $2(1+\ln((d+4)/2))+\varepsilon$, which improves to $2.919+\varepsilon$ for the special case of $d=1$.

Next, we explore approximation algorithms for the d-dimensional geometric bin packing problem (dD GBP). Caprara [18] gave a *harmonic-based* algorithm for dD GBP having an AAR of T_∞^{d-1} (where $T_\infty \approx 1.691$). However, their algorithm doesn't allow items to be rotated. This is in contrast to some common applications of dD GBP, like packing boxes into shipping containers. We give approximation algorithms for dD GBP when items can be orthogonally rotated about all or a subset of axes. We first give a fast and simple harmonic-based algorithm, called $\texttt{fullh}_k$, having an AAR of T_∞^d. We next give a more sophisticated harmonic-based algorithm, which we call $\texttt{HGaP}_k$, having an AAR of $T_\infty^{d-1}(1+\varepsilon)$. This gives an AAR of roughly $2.860+\varepsilon$ for 3D GBP with rotations, which improves upon the best-known AAR of 4.5. In addition, we study the *multiple-choice* bin packing problem that generalizes the rotational case. Here we are given n sets of d-dimensional cuboidal items and we have to choose exactly one

item from each set and then pack the chosen items. Our algorithms $\texttt{fullh}_k$ and $\texttt{HGaP}_k$ also work for the multiple-choice bin packing problem. We also give fast and simple approximation algorithms for the multiple-choice versions of dD strip packing and dD geometric knapsack. These algorithms have AARs T_∞^{d-1} and $(1-\varepsilon)3^{-d}$, respectively.

A rectangle is said to be δ-skewed if it has width at most δ or height at most δ. We give an approximation algorithm for bin packing δ-skewed rectangles whose asymptotic approximation ratio approaches 1 as δ approaches 0. Our result indicates that hard instances in geometric bin packing arise due to items that are large in both dimensions.

A packing of rectangles into a bin is said to be guillotine-separable iff we can use a sequence of end-to-end cuts to separate the items from each other. The asymptotic price of guillotinability (APoG) is the maximum value of $\text{opt}_G(I)/\text{opt}(I)$ for large $\text{opt}(I)$, where $\text{opt}(I)$ and $\text{opt}_G(I)$ are the minimum number of bins and the minimum number of guillotine-separable bins, respectively, needed to pack I. Computing lower and upper bounds on APoG is an important problem, since proving an upper bound smaller than 1.5 would beat the state-of-the-art algorithm for 2D GBP. The best-known lower and upper bounds are 4/3 and $T_\infty \approx 1.69103$, respectively. We analyze this problem for the special case of δ-skewed rectangles, where δ is a small constant (i.e., close to 0). We give a roughly 4/3-asymptotic-approximate algorithm for 2D GBP for this case, and our algorithm's output is guillotine-separable. This proves an upper-bound of roughly 4/3 on APoG for δ-skewed rectangles. We also prove a matching lower-bound of 4/3. This shows that hard examples for upper-bounding APoG include items that are large in both dimensions.

Publications based on this Thesis

[1] Arindam Khan and Eklavya Sharma. Tight approximation algorithms for geometric bin packing with skewed items, 2021. To appear in APPROX 2021. arXiv:2105.02827.

[2] Arindam Khan, Eklavya Sharma, and K. V. N. Sreenivas. Geometry meets vectors: Approximation algorithms for multidimensional packing, 2021. arXiv:2106.13951.

[3] Eklavya Sharma. Harmonic algorithms for packing d-dimensional cuboids into bins, 2020. arXiv:2011.10963.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the classical bin packing problem, we are given a set I of items. Each item $i \in I$ has a size $s(i) \in (0, 1]$ associated with it. Our goal is to partition I into the minimum number of bins, such that the sum of sizes of items in each bin is at most 1. The classical bin packing problem and its generalizations have diverse applications in computer science and operations research, like packing goods into trucks, allocating jobs to servers, allocating memory in computers [23], or assigning advertisements to station breaks in television programming,

This work focuses on approximation algorithms for geometric variants of the classical bin packing problem. In this chapter, we will first describe the classical bin packing problem and some of its well-known variants in more detail. With this context, we will then describe the contributions of this thesis.

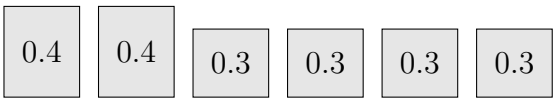
1.1 Well-Known Packing Problems

1.1.1 Classical Bin Packing

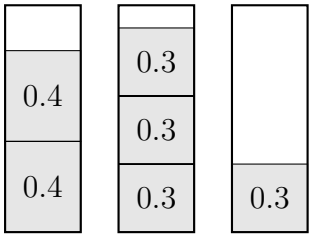
In the classical bin packing problem, we are given a set I of n items. Each item $i \in I$ has a size $s(i) \in (0, 1]$ associated with it. Our goal is to partition I into the minimum number of bins, such that the sum of sizes of items in each bin is at most 1. See Fig. 1.1 for an example.

Classical bin packing is known to be NP-hard. In fact, deciding whether a set of items can be packed into two bins is NP-complete, by a simple reduction from the partition problem. Hence, we look at approximation algorithms. Let $\mathrm{opt}(I)$ be the minimum number of bins required to pack a set I of items. A bin packing algorithm $\mathcal{A}$ is said to be α-approximate iff $\mathcal{A}$ requires at most $\alpha \,\mathrm{opt}(I)$ bins to pack I.

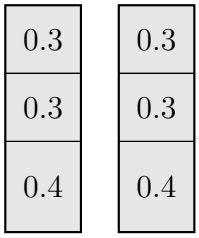
Since it is NP-complete to decide whether a set of items can be packed into two bins, it



(a) A set of six items: two items have size 0.4 and four items have size 0.3.



(b) A packing of the items into 3 bins.



(c) A packing of the items into 2 bins.

Figure 1.1: An example of classical bin packing. We want to minimize the number of bins, so the packing into 2 bins is better than the packing into 3 bins.

is NP-hard to obtain a polynomial-time algorithm for bin packing with approximation ratio less than 3/2. (Using the results of Dósa and Sgall, we can prove that the First-Fit Decreasing algorithm is 3/2-approximate [29, 28].) However, such a reasoning doesn't rule out the existence of an algorithm that uses $\mathrm{opt}(I) + 1$ bins. Therefore, we turn our attention to *asymptotic approximation algorithms*.

Definition 1.1 (Asymptotic approximation)**.** *A bin packing algorithm $\mathcal{A}$ is said to be α-asymptotic-approximate iff $\mathcal{A}$ requires at most $\alpha\,\mathrm{opt}(I) + \beta$ bins to pack I for some value $\beta \in o(\mathrm{opt}(I))$ (usually, β is a constant). α is called the asymptotic approximation ratio (AAR) of $\mathcal{A}$.*

Definition 1.2 (APTAS)**.** *A bin packing algorithm is called an Asymptotic Polynomial-Time Approximation Scheme (APTAS) if it accepts a parameter $\varepsilon > 0$ and gives an AAR of $1 + \varepsilon$. The running time for such an algorithm usually increases as ε decreases.*

For classical bin packing, an APTAS was given by Lueker and Vega [26].

1.1.2 Classical Knapsack

In the classical knapsack problem, we are given a set I of items. Each item $i \in I$ has a size $s(i) \in (0, 1]$ and profit $p(i) \in \mathbb{R}_{\geq 0}$ associated with it. Our goal is to pack the maximum profit subset of I into a bin, i.e., select a subset $S \subseteq I$ such that $\sum_{i \in S} s(i) \leq 1$ and $\sum_{i \in S} p(i)$ is maximized. In this problem, the bin is also called *knapsack*.

The classical knapsack problem is NP-hard by a reduction from the subset-sum problem. Hence, we look at approximation algorithms. Let $\mathrm{opt}(I)$ be the maximum profit of a subset of I that can be packed into the knapsack. An algorithm $\mathcal{A}$ for the knapsack problem is said to be α-approximate iff the profit of the items packed by $\mathcal{A}$ is at least $\mathrm{opt}(I)/\alpha$.

An algorithm for the knapsack problem is called a PTAS if it takes a constant $\varepsilon > 0$ as parameter, is $(1+\varepsilon)$-approximate, and runs in time polynomial in n. Additionally, if the running time is polynomial in both n and $1/\varepsilon$, then the algorithm is called an FPTAS.

FPTASes are known for the classical knapsack problem. There is a simple FPTAS that runs in $O(n^3/\varepsilon)$ time [79]. Lawler improved the running time to $O(n\log(1/\varepsilon) + 1/\varepsilon^4)$ [55].

1.1.3 Geometric Bin Packing

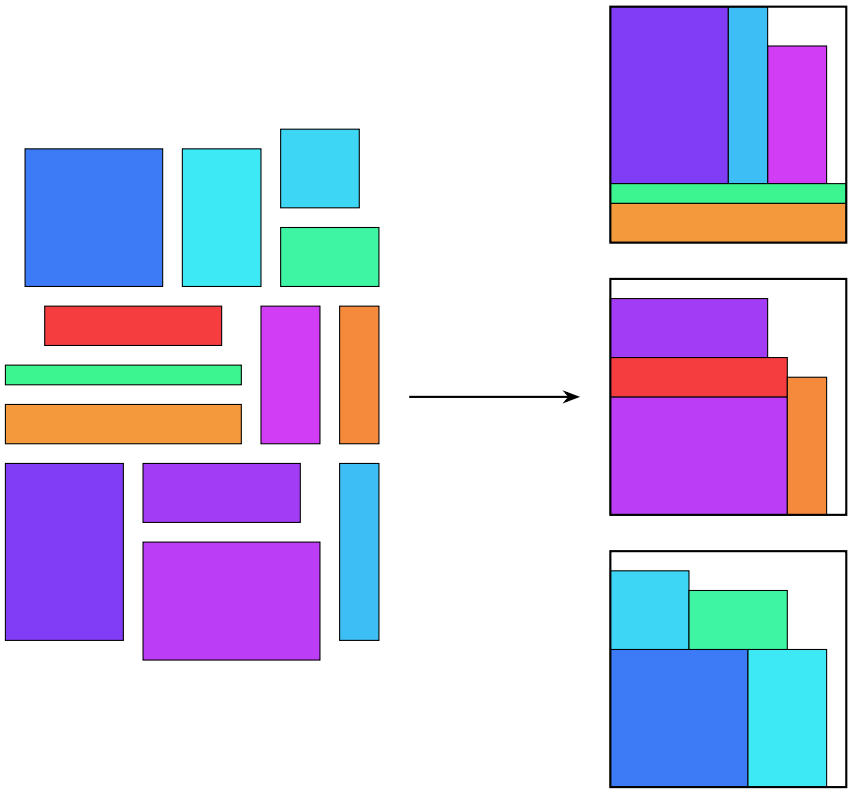
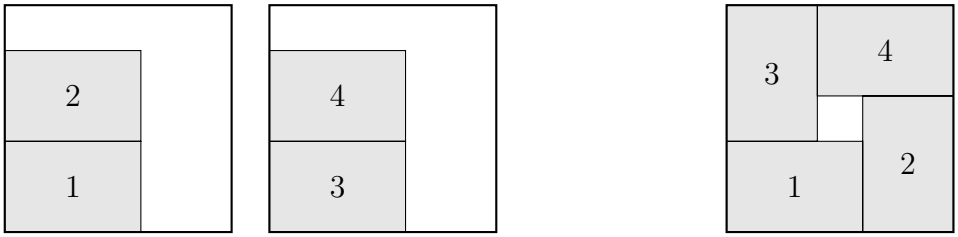


Figure 1.2: Packing 13 rectangles into 3 bins (without rotation).

In the 2-dimensional geometric bin packing problem (abbreviated as 2D GBP; also called *rectangle bin packing problem*), we are given a set I of n rectangular items and an infinite supply of identical rectangular bins. Our task is to pack the rectangles into the minimum number of bins such that in each bin, the items don't overlap. See Fig. 1.2 for an example. For each 2D

item i, let $w(i)$ denote the width and $h(i)$ denote the height. Let $a(i) := w(i)h(i)$ be the area of item i.

There are two commonly-studied versions of 2D GBP. In the non-rotational version, rotating the items is forbidden. In the rotational version, the items can be rotated by 90°. In both versions, the items and bins are oriented parallel to the coordinate axes. See Fig. 1.3 for an example. Note that if all items have width equal to the bin's width, then 2D GBP reduces to the classical bin packing problem.



(a) Packing without item rotation (b) Packing with item rotation

Figure 1.3: Packing 4 rectangular items, each of width 0.6 and height 0.4, into square bins of side length 1. Allowing rotation decreases the minimum number of bins needed to pack these items.

2D GBP finds applications in the wood-cutting, metal-cutting, paper and cloth industries, where rectangular pieces need to be cut out of standard-sized sheets, and item rotations are usually allowed. Non-rotational 2D GBP can be used for placing advertisements on web pages and newspapers.

The problem can be extended to three dimensions (3D GBP), where the bins and items are cuboids. Since a cuboid can have 6 possible orientations (see Fig. 1.4), there can be many versions of 3D GBP depending on which orientations of items are allowed. 3D GBP can be used to pack boxes into shipping containers. Here the boxes can usually be rotated in any way, but there are exceptions: for example, some boxes may need to be kept upright due to fragile contents inside. Such boxes can only be rotated around the vertical axis.

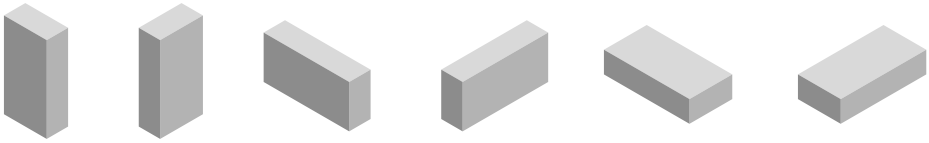


Figure 1.4: All six orientations of a cuboid of dimensions 0.2, 0.4 and 0.8.

We can extend 2D and 3D GBP to even higher dimensions. In dD GBP ($d \geq 1$), items and bins are dD cuboids. A dD cuboid is defined as the cross-product of d intervals from the real

line. A cube is a cuboid that has the same length in each dimension. For example, a dD cube of side length 1 is the set $[0, 1]^d$. Note that 1D GBP is the classical bin packing problem.

In the non-rotational version of dD GBP, we can assume without loss of generality that the bin is a cube of side length 1. This is because we can scale the dimensions of the bins and items by the same factor. Note that this assumption doesn't hold for the rotational version.

For 2D GBP, the algorithm by Bansal and Khan [14] gives the best-known AAR of $1 + \ln(1.5) + \varepsilon \approx 1.40547 + \varepsilon$. An APTAS cannot exist for 2D GBP unless P = NP [11, 21]. For non-rotational dD GBP where $d \geq 3$, the algorithm by Caprara [18] gives the best-known AAR of roughly T_∞^{d-1}, where $T_\infty \approx 1.69103$.

1.1.4 Guillotine-Separable Geometric Bin Packing

In many practical cases of 2D GBP, there are additional constraints on how items can be packed into bins, like in the *two-dimensional guillotine-separable geometric bin packing problem* (2D GuillBP).

Definition 1.3. *A packing of rectangles into a bin is said to be* k-stage guillotine-separable *or* k-stage guillotinable *iff we can separate all the items in the bin using at most k stages of end-to-end cuts (also called* guillotine cuts*), where in each stage, all cuts are parallel to the x-axis or all cuts are parallel to the y-axis. A packing of rectangles into a bin is said to be* guillotine-separable *or* guillotinable *iff it is k-stage guillotinable for some k.*

See Fig. 1.6 for an example of 3-stage guillotinable packing and Fig. 1.5 for examples of non-guillotinable packing.

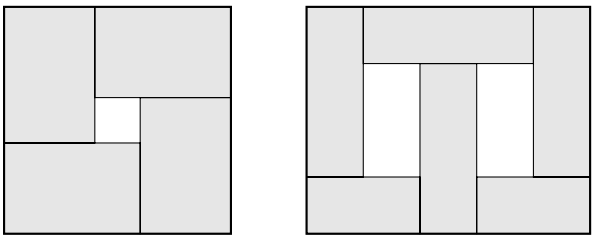


Figure 1.5: Two bins that are not guillotinable.

2D GuillBP is a variant of 2D GBP where we are given a set I of rectangles and our task is to pack them into the minimum number of bins such that each bin is guillotinable. Similarly, in the k-stage 2D bin packing problem, we have to pack a set of rectangles into the minimum number of bins such that each bin is k-stage guillotinable.

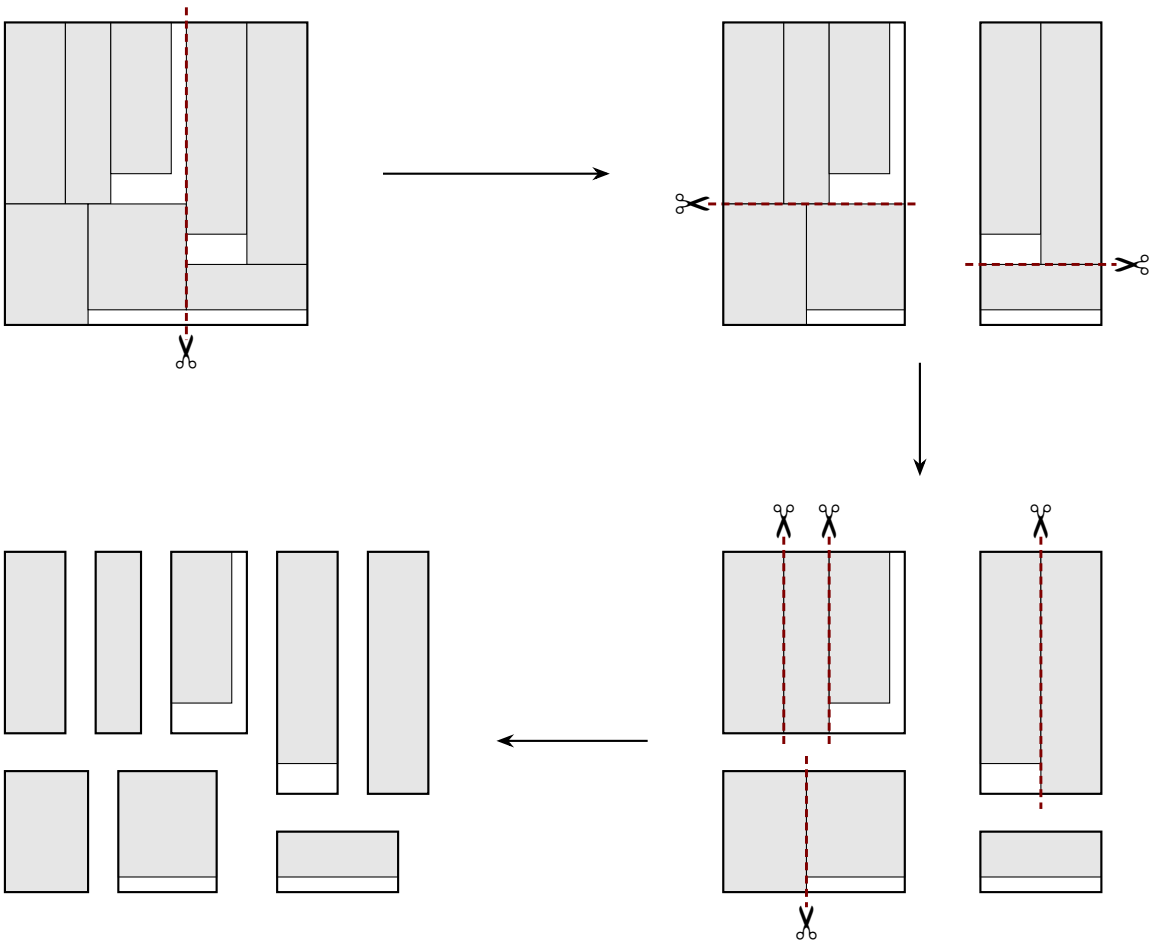


Figure 1.6: Separating items using 3 stages of guillotine cuts.

2D GuillBP is relevant in sheet-cutting industries [59, 67, 70], where cutting machines can only make guillotine cuts. Guillotine cuts simplify the design and programming of cutting machines and reduce their operational cost, which is important if the raw material being cut is relatively inexpensive.

Unlike 2D GBP, for which obtaining an APTAS is NP-hard, an APTAS was given for non-rotational 2D GuillBP by Bansal, Lodi and Sviridenko [15].

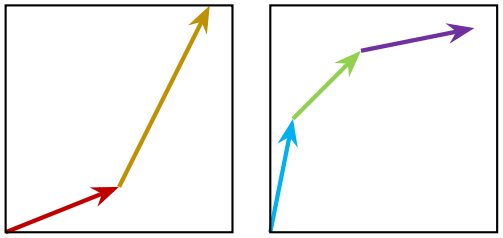
2D GuillBP has connections to other interesting problems, like guillotine-separable knapsack [51] and maximum independent set of rectangles [1, 52].

1.1.5 Vector Bin Packing

In the d-dimensional vector bin packing problem (dD VBP), we are given a set I of d-dimensional vectors, and our task is to partition the vectors into bins such that in each bin, the sum of

the vectors in that bin have ℓ_∞-norm at most 1. Formally, let $I := \{v^{(1)}, v^{(2)}, \ldots, v^{(n)}\}$ be the items, where $v^{(i)} \in (0,1]^d$ for each i. Then we have to partition the vectors into bins such that in each bin B, $\sum_{v \in B} v_j \leq 1$ for each $j \in \{1, 2, \ldots, d\}$.

(a) Five 2D vectors.



(b) Packing the vectors into two bins. A set of 2D vectors lies in a bin iff their sum lies inside a square of side length 1.

Figure 1.7: Packing five 2D vectors into two bins.

dD VBP has applications in resource allocation problems. Consider a set of tasks, each of which have a d resource requirements. The resources could be CPU time, memory, disk IO, network IO, etc. We have to assign these tasks to the minimum number of servers, where each server has a capacity on each resource. This is an example of dD VBP, where the tasks are items and servers are bins.

For dD VBP, the best-known AAR is $\ln d + O(1)$ [10, 12]. For 2D VBP, the best-known AAR is $1 + \ln(1.5) + \varepsilon \approx 1.40547 + \varepsilon$ [12].

1.2 Contributions of This Thesis

In this thesis, we address the following four problems related to geometric bin packing.

1.2.1 Generalized Multidimensional Bin Packing

(Joint work with Prof. Arindam Khan and Sreenivas Karnati.)

Geometric packing and vector packing are well-studied generalizations of the classical bin packing problem. However, often in practice, we encounter a mixture of geometric and vector constraints. Consider the following airlines cargo problem [62]: We have boxes to load in an airline cargo container. In addition to the geometric constraint that all the boxes must fit within the container, we also have a constraint that the total weight of the loaded boxes should be within a specified capacity. Thus, in this problem, three dimensions are geometric and the weight is a vector constraint.

Weight has been an important constraint to consider for packing in logistics and supply chain management, e.g., cranes and other equipment can be damaged by the bins being too heavy, or by a skewed weight distribution [2]. While the container loading problems mostly consider packing items inside a container, the container stowage planning problem considers the stacking of the containers onto and off cargo ships [61]. Even when different cargoes are packed into a fleet of aircraft for transport, one needs the individual cargoes to be not too heavy to ensure stability and less fuel consumption [3]. Similar problems find applications in vehicle routing with loading constraints [17]. Many practical heuristics [73, 75] have been proposed for these kinds of problems. Several companies (such as Driw, Boxify, Freightcom) and practical packages [81] have considered the problem. In many cases, we also want to ensure a limit on other attributes, such as the amount of magnetism, radioactivity, or toxicity. Each of these properties can be considered as additional vector dimensions.

Such multidimensional packing problems are also getting attention due to their connections with fair resource allocation [63]. In recent years, a considerable amount of research has focused on group fairness [47, 76] such that the algorithms are not biased towards (or against) some groups or categories. One such notion of fairness is *restricted dominance* [16], which upper bounds the number (or size) of items from a category. These different categories can be considered as dimensions. E.g., in a container packing problem for flood relief, one needs to ensure that the money spent on a container is fairly distributed among different types of items (such as medicine, food, garments). Hence, for each category, there is an upper bound on the value that can go into a container.

Formally, we are given n items $I := \{1, 2, \ldots, n\}$ that are (d_g, d_v)-dimensional, i.e., item i is a d_g-dimensional cuboid of lengths $\ell_1(i), \ell_2(i), \ldots, \ell_{d_g}(i)$ and has d_v non-negative weights $v_1(i), v_2(i), \ldots, v_{d_v}(i)$. A (d_g, d_v)-dimensional bin is a d_g-dimensional cuboid of length 1 in each geometric dimension and weight capacity 1 in each of the d_v vector dimensions. A feasible packing of items into a bin is a packing where items are packed parallel to the axes without overlapping, and for all $j \in [d_v]$, the sum of the j^{th} weight-dimension of the items in the bin is at most 1. In the (d_g, d_v) bin packing problem (BP), we have to feasibly pack all items into the minimum number of bins. In the (d_g, d_v) knapsack problem (KS), each item i also has an associated nonnegative profit $p(i)$, and we have to feasibly pack a maximum-profit subset of the items into a single bin (also called *knapsack*). (d_g, d_v) packing problems generalize both d_g-dimensional geometric packing (when $d_v = 0$) and d_v-dimensional vector packing (when $d_g = 0$). Already for vector packing, if d_v is part of the input, there is an approximation hardness of ${d_v}^{1-\varepsilon}$ unless NP=ZPP [12]. Thus, throughout the paper we assume both d_g and d_v to be constants.

1.2.1.1 Our Results

We study the first approximation algorithms for general (d_g, d_v) BP, with a focus on $d_g = 2$. We give two simple algorithms for $(2, d)$ BP, called `simplePack` and `betterSimplePack`, having AARs of $6(d+1)$ and $3(1+\ln(d+1))+\varepsilon$, respectively, for any $\varepsilon > 0$. For $d = 1$, `betterSimplePack`'s AAR improves to $\approx 4.21640 + \varepsilon$.

Next, we modify the Round-and-Approx (R&A) framework [10, 14] so that it works for (d_g, d_v) BP. We combine R&A with the `simplePack` algorithm to get an AAR of $2(1+\ln(3(d+1)))+\varepsilon$ for $(2, d)$ BP. This improves upon the AAR of `betterSimplePack` for $d \geq 3$.

Next, we obtain a more sophisticated algorithm for $(2, d)$ BP, called `cbPack` (abbreviation for *container-based packing*), that fits into the R&A framework and has an even better AAR.

Theorem 1.1. *The* `cbPack` *algorithm for (2, d) BP accepts a parameter $\varepsilon > 0$, and has an AAR of $2(1+\ln((d+4)/2))+\varepsilon$ (improves to $2(1+\ln((d+3)/2))+\varepsilon$ when items can be rotated by $90°$). For $d = 1$, the AAR improves to $2(1+\ln(19/12))+\varepsilon \approx 2.919+\varepsilon$ (further improves to $2(1+\ln(3/2))+\varepsilon \approx 2.811+\varepsilon$ when items can be rotated).*

Table 1.1: Comparison of asymptotic approximation ratios of our algorithms for $(2, d)$ BP.

Algorithm	AAR for $(2, d)$ BP	AAR for $(2, 1)$ BP
`simplePack`	$6(d+1)$	12
`betterSimplePack`	$3(1+\ln(d+1))+\varepsilon$	$3(1+\ln(3/2))+\varepsilon \approx 4.216+\varepsilon$
`simplePack` with R&A	$2(1+\ln(3(d+1)))+\varepsilon$	$2(1+\ln 6)+\varepsilon \approx 5.5835+\varepsilon$
`cbPack` (without rotation)	$2(1+\ln(\frac{d+4}{2}))+\varepsilon$	$2(1+\ln(19/12))+\varepsilon \approx 2.919+\varepsilon$
`cbPack` (with rotation)	$2(1+\ln(\frac{d+3}{2}))+\varepsilon$	$2(1+\ln(3/2))+\varepsilon \approx 2.811+\varepsilon$

We also show how to extend `simplePack` and `betterSimplePack` to (d_g, d_v) BP to obtain AARs $2b(d_v+1)$ and $b(1+\ln(d_v+1)+\varepsilon)$, respectively, where $b := 9$ for $d_g = 3$ and $b := 4^{d_g}+2^{d_g}$ for $d_g > 3$. We also give a similar algorithm for (d_g, d_v) KS having approximation ratio $b(1+\varepsilon)$.

One of our main contributions is the enhancement of the R&A framework [10, 14] to a wider class of algorithms. The R&A framework is a simple but powerful technique, originally given by Bansal, Caprara and Sviridenko [10], to improve the AAR of a bin packing algorithm by combining it with randomized rounding of a linear program. R&A may have the potential to improve the AARs of several packing problems, but its applicability is limited because it only works with *subset-oblivious* bin packing algorithms, and proving that an algorithm is subset-oblivious is difficult.

Bansal and Khan [14] partially removed this limitation by proving that a large class of algorithms for geometric and vector bin packing, called *Rounding-based* algorithms, is subset-oblivious. We make further improvements on this front by showing that an even larger class of algorithms is subset-oblivious. This class includes some of our algorithms for $(d_g,\ d_v)$ BP, like `simplePack` and `cbPack`. We expect that our progress will help in better understanding the power of R&A and extending it to other set-cover type problems, like round-SAP and round-UFP [30].

1.2.2 Harmonic Algorithms for dD GBP with Rotations

In his seminal paper, Caprara [18] devised a polynomial-time algorithm for non-rotational dD GBP called $\texttt{HDH}_k$ (Harmonic Decreasing Height), where $k \in \mathbb{Z}$ is a parameter to the algorithm. $\texttt{HDH}_k$ has AAR equal to T_k^{d-1}, where T_k is a decreasing function of k and $T_\infty := \lim_{k\to\infty} T_k \approx 1.69103$. The algorithm $\texttt{HDH}_k$ is based on an extension of the harmonic algorithm [56] for classical bin packing. For $d \geq 3$ and large k, $\texttt{HDH}_k$ has the best AAR among all known algorithms for dD GBP.

A limitation of $\texttt{HDH}_k$ is that it does not allow rotation of items. This is in contrast to some real-world problems, like packing boxes into shipping containers ($d = 3$), where items can often be rotated orthogonally, i.e., 90° rotation around all or a subset of axes.

Furthermore, known algorithms for rotational 3D GBP have a significantly larger AAR than what the $\texttt{HDH}_k$ algorithm gives for non-rotational 3D GBP. When items can be rotated about all axes, Miyazawa and Wakabayashi [60] gave a 4.89-asymptotic-approximation algorithm. Epstein and van Stee [31] improved the AAR to 4.5 when the base of the bin is a square. On the other hand, $\texttt{HDH}_k$ gives an AAR of roughly $T_\infty^2 \approx 2.85958$ for large k.

Orientation constraints may sometimes limit the vertical orientation of a box to one dimension (e.g., some items require a face labeled 'This side up' to always be on top) or to two dimensions (e.g., long but low and narrow box should not be placed on its smallest surface). These constraints are introduced to deter goods and packaging from being damaged and to ensure the stability of the load. Current algorithms for 3D GBP either allow rotating all items along the same set of axes or don't allow rotating any item, i.e., they enforce the same orientation constraints over all items in the input.

We address these issues by giving algorithms for dD GBP that are similar to $\texttt{HDH}_k$ but allow items to be rotated and allow orientation constraints to vary across items. We first give a fast and simple algorithm called $\texttt{fullh}_k$ that has an AAR of T_k^d. We next give a more sophisticated algorithm called $\texttt{HGaP}_k$ that has an AAR of $T_k^{d-1}(1+\varepsilon)$.

1.2.2.1 Multiple-Choice Packing

$\texttt{fullh}_k$ and $\texttt{HGaP}_k$ can be extended to work for the dD multiple-choice geometric bin packing problem (dMCBP), which we will describe now. dMCBP generalizes both the rotational and non-rotational versions of dD GBP. It also captures the concept of variable orientation constraints across items. This perspective will help us design algorithms for the rotational case.

In dMCBP, we are given a set $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, where for each j, I_j is a set of items, henceforth called an *itemset*. We have to pick exactly one item from each itemset and pack those items into the minimum number of bins. See Fig. 1.8 for an example.

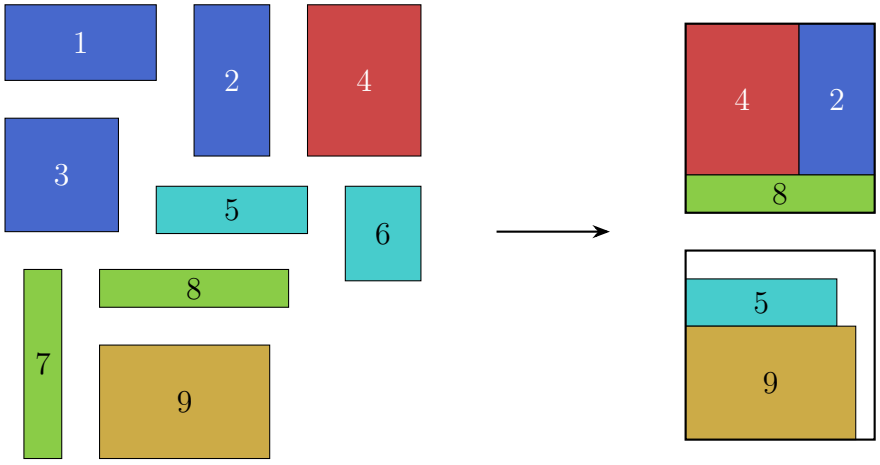


Figure 1.8: 2MCBP example: packing the input $\mathcal{I} = \{\{1,2,3\},\{4\},\{5,6\},\{7,8\},\{9\}\}$ into two bins. Here items of the same color belong to the same itemset.

We can model rotation using multiple-choice packing: Given a set I of items, for each item $i \in I$, create an itemset I_i that contains all allowed orientations of i. Then the optimal solution to $\mathcal{I} := \{I_i : i \in I\}$ will tell us how to rotate and pack items in I.

Some algorithms for 2D bin packing with rotations assume that the bin is square [10, 41, 14]. This assumption holds without loss of generality when rotations are forbidden, because we can scale the items. But if rotations are allowed, this won't work because items i_1 and i_2 that are rotations of each other may stop being rotations of each other after they are scaled. Multiple-choice packing algorithms can be used in this case: for each item $i \in I$, we will create an itemset I_i that contains scaled orientations of i.

Multiple-choice packing problems have been studied before. Lawler gave an FPTAS for the multiple-choice classical knapsack problem [55]. Patt-Shamir and Rawitz gave an algorithm for multiple-choice vector bin packing having AAR $O(\log d)$ and a PTAS for multiple-choice vector knapsack [64]. Similar notions have been studied in the scheduling of malleable or moldable jobs [82, 38].

1.2.2.2 Our Results

In our work, we extend and generalize $\texttt{HDH}_k$ to dMCBP. dMCBP subsumes the rotational case for geometric bin packing, and we believe dMCBP is an important natural generalization of geometric bin packing that may be of independent interest.

In Section 6.2, we show an $O(N+n\log n)$-time algorithm for dMCBP, called $\texttt{fullh}_k$, having an AAR of T_k^d, where n is the number of itemsets and N is total number of items across all the n itemsets. In Section 6.3, we show an algorithm for dMCBP, called $\texttt{HGaP}_k$, having an AAR of $T_k^{d-1}(1+\varepsilon)$ and having a running time of $N^{O(1/\varepsilon^2)}n^{(1/\varepsilon)^{O(1/\varepsilon)}}$. For $d \geq 3$, this matches the present best AAR for the case where rotations are forbidden. Also, for large k, this gives an AAR of roughly $T_\infty^2 \approx 2.85958$ for 3D GBP when orthogonal rotations are allowed, which is an improvement over the previous best AAR of 4.5 [31], an improvement after fourteen years.

Our techniques can be extended to some other packing problems, like strip packing and geometric knapsack. In Section 6.6, we define the dD multiple-choice strip packing problem (dMCSP) and extend Caprara's $\texttt{HDH}_k$ algorithm [18] to dMCSP. The algorithm has AAR T_k^{d-1} and runs in time $O(N+n\log n)$, where n is the number of itemsets and N is the total number of items across all itemsets. In Section 6.7, we define the dD multiple-choice geometric knapsack problem (dMCKS), and for any $0<\varepsilon<1$, we show an $O(N\log N + Nn/\varepsilon)$-time algorithm that is $3^d(1+\varepsilon)$-approximate.

Our algorithms produce shelf-based packings (we formally define *shelf-based* later). An interesting property of Caprara's $\texttt{HDH}_k$ algorithm is that it is, in some sense, optimal for shelf-based packing. Formally, Caprara [18] showed that no shelf-based algorithm for 2D GBP can get an AAR better than $T_\infty \approx 1.69103$, and his $\texttt{HDH}_k$ algorithm achieves an AAR of T_k^{d-1} for dD GBP. In Section 6.9, we extend that result to show that no shelf-based algorithm for dD GBP can get an AAR better than T_∞^{d-1}.

1.2.3 Guillotine-Separable Packing of Skewed Rectangles

(Joint work with Prof. Arindam Khan.)

Unlike 2D GBP, for which obtaining an APTAS is NP-hard, an APTAS was given for non-rotational 2D GuillBP by Bansal, Lodi and Sviridenko [15]. A natural question, then, is whether the optimal solution to 2D GuillBP a good approximation for 2D GBP.

Formally, let $\mathrm{opt}_g(I)$ be the minimum number of guillotinable bins needed to pack I. Let α be the smallest constant such that $\mathrm{opt}_g(I) \leq \alpha\,\mathrm{opt}(I)+\beta$, where $\beta \in o(\mathrm{opt}(I))$. We call α the *Asymptotic Price of Guillotinability* (APoG). Hence, if we use the APTAS for 2D GuillBP as an approximation algorithm for 2D GBP, then the AAR would be $\alpha(1+\varepsilon)$. Therefore, we

would like to obtain tight upper and lower bounds on APoG.

It is a simple and well-known fact that APoG $\geq 4/3$[1]. Caprara's $\texttt{HDH}_k$ algorithm [18] outputs a 2-stage packing, so APoG $\leq T_\infty \approx 1.69103$.

We consider the special case where the rectangles are δ-skewed, i.e., each rectangle has height at most δ or width at most δ, where δ is a small constant. We prove an upper-bound on APoG for skewed rectangles by giving an algorithm for 2D GBP, called `skewed4Pack`, that outputs a 4-stage packing and has an AAR of

$$\frac{4}{3}\left(1+\frac{4\delta}{1-\delta}\right)(1+\varepsilon).$$

We also prove a lower-bound of 4/3 on APoG by giving a set of δ-skewed rectangles that cannot be efficiently packed by a guillotine-separable packing. Therefore, when δ is close to zero, we get that APoG is roughly 4/3.

This indicates that to tighten the bound on APoG for the general case, we should focus on big rectangles, i.e., rectangles that have width and height more than a constant δ.

The δ-skewed case has also recently received attention in the 2D strip packing problem [34].

1.2.4 Almost-Optimal Bin Packing of Skewed Rectangles

(Joint work with Prof. Arindam Khan.)

For a constant $\delta > 0$, a rectangle is said to be δ-skewed iff either its width is at most δ or its height is at most δ. We give an approximation algorithm for bin packing δ-skewed rectangles where the algorithm's AAR approaches 1 as δ approaches 0. Formally, we give an algorithm for 2D GBP, called `skewedCPack` (abbreviates *skewed compartmental packing*), that accepts a parameter ε, and we show that for every constant $\varepsilon \in (0,1)$, there exists a constant $\delta \in (0,\varepsilon)$ such that the algorithm has an AAR of $1+\varepsilon$ when all items in the input are δ-skewed rectangles.

Our result shows that the approximability of the δ-skewed case is very different from the general 2BP problem, since it is NP-hard to obtain an asymptotic approximation ratio better than $1 + 1/2196$ for general 2BP [21].

The best-known AAR for 2D GBP is $1+\ln(1.5)+\varepsilon \approx 1.405+\varepsilon$. Our result indicates that to improve upon algorithms for 2D GBP, we should focus on big rectangles, i.e., rectangles whose width and height are both more than a constant δ.

[1]Consider a set I of items containing $2m$ rectangles of width 0.6 and height 0.4 and $2m$ rectangles of width 0.4 and height 0.6. Then $\text{opt}(I) = m$ and $\text{opt}_g(I) = \lceil 4m/3 \rceil$.

1.3 Organization of the Thesis

- In Chapter 2, we describe prior work on well-known packing problems (problems of Section 1.1 and other related problems).
- In Chapter 3, we describe the notation and preliminaries needed for the rest of the thesis.
- In Chapter 4, we describe our simple algorithms for generalized multidimensional bin packing ((2, d) BP) and our extension of the Round-and-Approx framework.
- In Chapter 5, we describe the `cbPack` algorithm for (2, d) BP, which improves upon the algorithms of Chapter 4.
- In Chapter 6, we describe our algorithms for d-dimensional multiple-choice geometric packing problems (which generalize packing with item rotations). We mostly focus on multiple-choice bin packing but also describe algorithms for multiple-choice strip packing and multiple-choice knapsack.
- In Chapter 7, we show how to bound the asymptotic price of guillotinability (APoG) for skewed rectangular items. We show a simple lower bound of 4/3 and prove an upper bound of roughly 4/3 by giving a 4-stage $\frac{4}{3}(1+4\delta)(1+\varepsilon)$-asymptotic-approximation algorithm for 2D GBP for δ-skewed rectangles.
- In Chapter 8, we give an algorithm for almost-optimally packing δ-skewed rectangles into bins.
- In Chapter 9, we give concluding remarks and future directions.

Chapter 2

Related Problems and Prior Work

2.1 Classical Bin Packing

In the online version of bin packing, the items arrive one-by-one, and for each item, we have to immediately and irrevocably pack it into a bin. In the online version, the (asymptotic) approximation ratio is also called the (asymptotic) *competitiveness ratio*. (The non-online version is called the offline version, i.e., where we can read the whole input before we start packing.)

The Next-Fit algorithm [46] is one of the simplest algorithms for the online classical bin packing problem. In this algorithm, we start with an empty bin, and designate it as the *open bin*. We repeatedly pack items into the open bin till we come across an item that doesn't fit in the open bin. We then *close* that bin and *open* a new bin and resume. Let $s(I)$ denote the sum of sizes of all items in I. It is easy to prove that Next-Fit uses at most $\lceil 2s(I) \rceil$ bins. Since $s(I) \leq \mathrm{opt}(I)$, we get that Next-Fit is 2-approximate.

Lee and Lee [56] gave an algorithm for online classical bin packing, called the Harmonic$_k$ algorithm. This algorithm takes as input a set I of items and an integer parameter $k \geq 2$. The number of bins used by Harmonic$_k$ to pack I is less than $T_k \,\mathrm{opt}(I) + k$, where T_k is a decreasing function of k and $T_\infty := \lim_{k\to\infty} T_k \approx 1.6910302$.

Many other algorithms have been devised for the online classical bin packing problem. See Table 2.1 for examples and [23] for a detailed survey. The best algorithm we are aware of is the Advanced Harmonic algorithm by Balogh, Békési, Dósa, Epstein and Levin [6] that has an AAR of 1.57829.

Balogh, Békési and Galambos gave a lower bound of $248/161 \approx 1.54037$ on the asymptotic competitive ratio [8]. This bound was recently improved to 1.54278 [7], which is the best-known

Table 2.1: Online algorithms for classical bin packing.

Algorithm	Approximation guarantee
Next-Fit [46]	$\leq 2\,\mathrm{opt}(I)$
First-Fit [29]	$\leq \lfloor 1.7\,\mathrm{opt}(I) \rfloor$
Harmonic$_k$ [56]	$< T_k\,\mathrm{opt}(I) + k \quad (T_\infty \approx 1.69103)$
Advanced Harmonic [6]	$\leq 1.57829\,\mathrm{opt}(I) + O(1)$

lower bound we are aware of. See [23] for a detailed survey on lower bounds.

For the offline version of classical bin packing, First-Fit Decreasing (FFD) is a popular algorithm. Johnson [46] showed that the number of bins used by FFD is at most $(11/9)\,\mathrm{opt}(I) + 4$. The additive constant 4 was improved by Dósa to 2/3, which is tight [28].

Lueker and Vega gave the first APTAS for classical bin packing [26]. This was later improved upon by Karmarkar and Karp to an algorithm that uses at most $\mathrm{opt}(I) + O(\log^2 \mathrm{opt}(I))$ bins [48]. Rothvoss gave an algorithm that uses at most $\mathrm{opt}(I) + O(\log \mathrm{opt}(I) \log\log \mathrm{opt}(I))$ bins [68]. Hoberg and Rothvoss gave an algorithm that uses at most $\mathrm{opt}(I) + O(\log \mathrm{opt}(I))$ bins [37] (see Table 2.2).

Table 2.2: Offline algorithms for classical bin packing.

Algorithm	Approximation guarantee
First-Fit Decreasing (FFD) [28]	$\leq \frac{11}{9}\,\mathrm{opt}(I) + \frac{2}{3}$
Lueker, Vega [26]	$\leq (1+\varepsilon)\,\mathrm{opt}(I) + O(1/\varepsilon^2)$
Karmarkar, Karp [48]	$\leq \mathrm{opt}(I) + O(\log^2 \mathrm{opt}(I))$
Rothvoss [68]	$\leq \mathrm{opt}(I) + O(\log \mathrm{opt}(I) \log\log \mathrm{opt}(I))$
Hoberg, Rothvoss [37]	$\leq \mathrm{opt}(I) + O(\log \mathrm{opt}(I))$

On the hardness side, to the best of our knowledge, an $\mathrm{opt}(I) + 1$ algorithm hasn't yet been proven to not exist.

2.2 Classical Knapsack

There is a simple FPTAS for classical knapsack that runs in $O(n^3/\varepsilon)$ time [79]. Lawler improved the running time to $O(n \log(1/\varepsilon) + 1/\varepsilon^4)$ [55]. The running time was improved to $O(n \log(1/\varepsilon) + (1/\varepsilon)^{12/5})$ by Chan [19] and to $O(n \log(1/\varepsilon) + (1/\varepsilon)^{9/4})$ by Jin [45].

2.3 Geometric Bin Packing

The Next-Fit Decreasing Height (NFDH) algorithm by Coffman, Garey, Johnson and Tarjan [24] is one of the simplest algorithms for 2D GBP (for both the rotational and non-rotational versions), which is 4-asymptotic-approximate. Chung, Garey and Johnson [22] gave an algorithm for non-rotational 2D GBP, which we call FFDH-FF, and proved that it has an AAR of $17/8 = 2.125$. Kenyon and Rémila [49] gave an APTAS for a problem called *2D strip packing*, which gives a $(2+\varepsilon)$-asymptotic-approximation algorithm for non-rotational 2D GBP.

Caprara [18] gave an algorithm for dD GBP, called $\texttt{HDH}_k$, that takes a parameter k as input and has an AAR of roughly T_∞^{d-1} (recall that $T_\infty \approx 1.69103$) when k is large. Caprara also simplified and improved the proofs of approximation of some known algorithms, e.g., the AAR of FFDH-FF is $187/90 = 2.0\overline{78}$. Their results, along with those of Baker and Coffman [5], imply that the AAR of NFDH is $2T_\infty \approx 3.38206$. All the results in [18] (including the $\texttt{HDH}_k$ algorithm) only work for the non-rotational version.

Bansal, Caprara and Sviridenko gave an algorithm for 2D GBP, for both the non-rotational version and the rotational version when the bin is square, having an AAR of $1 + \ln(T_\infty) + \varepsilon \approx 1.52534 + \varepsilon$ [10]. Their algorithm was a combination of the Round-and-Approx (R&A) framework [10] and the HDH algorithm by Caprara [18]. Jansen and Prädel gave a $(1.5+\varepsilon)$-asymptotic-approximation algorithm for 2D GBP [39, 41] for both the non-rotational version and the rotational version when the bin is square. Bansal and Khan [14] combined Jansen and Prädel's algorithm with the Round-and-Approx framework to improve the AAR to $1 + \ln 1.5 + \varepsilon \approx 1.40547 + \varepsilon$.

Table 2.3: Algorithms for 2D geometric bin packing.

Algorithm	Asymptotic Approximation Ratio	Works for rotational version?
NFDH [24]	4	Yes
NFDH [24, 18, 5]	$2T_\infty \approx 3.38206$	No
FFDH-FF [24, 18]	$187/90 \approx 2.0\overline{78}$	No
HDH [18]	$T_\infty \approx 1.69103$	No
HDH with R&A [10]	$1 + \ln(T_\infty) + \varepsilon \approx 1.52534 + \varepsilon$	Yes (square bin only)
Jansen, Prädel [39, 41]	$1.5 + \varepsilon$	Yes (square bin only)
Bansal, Khan [14]	$1 + \ln(1.5) + \varepsilon \approx 1.40547 + \varepsilon$	Yes (square bin only)

Bansal, Correa, Kenyon and Sviridenko [11] give an APTAS for dD BP for the case where all items are dD cubes (recall that a cube is a cuboid having the same length in each dimension).

Bansal, Correa, Kenyon and Sviridenko [11] proved that if P≠NP, then an APTAS does not exist for 2D GBP. Chlebík and Chlebíková [21] proved that it is NP-hard to solve 2D GBP with an AAR better than $1 + 1/3792 \approx 1.000264$ for the non-rotational version and an AAR better than $1 + 1/2196 \approx 1.000455$ for the rotational version. Hence, unlike classical bin packing, an APTAS cannot exist for 2D GBP.

For the version of 3D GBP where items can be rotated about all axes, Miyazawa and Wakabayashi [60] gave a 4.89-asymptotic-approximation algorithm. Epstein and van Stee [31] improved the AAR to 4.5 when the base of the bin is a square.

2.4 Geometric Knapsack

In the dD geometric knapsack problem (dD GKS), we are given a set I of items, where each item $i \in I$ is a dD cuboid and has a profit $p(i) \geq 0$ associated with it. Our goal is to pack the maximum profit subset of I into a dD cuboidal bin. In this problem, the bin is also called *knapsack*. We can have different versions of the problem depending on whether items are allowed to be rotated or not.

Jansen and Zhang give a $(2 + \varepsilon)$-approximation algorithm [44] for both the rotational and non-rotational versions of 2D GKS. Gálvez, Grandoni, Heydrich, Ingala, Khan and Wiese give a $17/9 + \varepsilon \approx 1.\overline{8}9 + \varepsilon$ approximation algorithm for non-rotational 2D GKS and a $(3/2 + \varepsilon)$-approximation algorithm for the rotational version when the knapsack is square.

For the special case of 2D GKS where $p(i)/a(i)$ is lower and upper bounded by constants for each item $i \in I$, there is a PTAS to pack a maximum-profit subset of I into a knapsack [9]. This works for both the non-rotational and rotational version.

For 3D GKS, [27] give a $(7 + \varepsilon)$-approximation algorithm for the non-rotational version. When items can be rotated by 90° either around the vertical axis only or around all axes, they obtain algorithms with approximation ratios $6 + \varepsilon$ and $5 + \varepsilon$, respectively, assuming that the knapsack is a cube.

2.5 Strip Packing

In the dD strip packing problem (dD SP), we are given a set I of dD cuboidal items. Our goal is to pack the items into a single dD cuboid, called *strip*, where the first $d-1$ dimensions of the strip are given and the d^{th} dimension of the strip (called *height*) must be minimized. We get different versions of the problem depending on whether items are allowed to be rotated or not.

For any dD cuboid, the first $d-1$ dimensions are called *base dimensions*, and the d^{th} dimension is called the *height*. When item rotations are not allowed, we can assume without loss of generality that the strip has length 1 in each base dimension, and that the maximum height of an item is at most 1. This is because we can scale the dimensions of the bins and items by the same factor.

Definition 2.1 (Asymptotic approximation for SP)**.** *A strip packing algorithm $\mathcal{A}$ is said to be α-asymptotic-approximate iff for each input I, $\mathcal{A}$ packs I into a strip of height at most $\alpha \operatorname{opt}(I) + \beta h_{\max}$ for some value $\beta \in o(\operatorname{opt}(I))$ (usually, β is a constant), where $h_{\max}$ is the maximum height of any item in I. α is called the asymptotic approximation ratio (AAR) of $\mathcal{A}$.*

We can similarly define what it means for a strip packing algorithm to be an APTAS.

When all items in a dD SP instance have the same height, and item rotations are not allowed, then the problem reduces to $(d-1)$D GBP. Therefore, dD SP is a generalization of $(d-1)$D GBP.

Coffman, Garey, Johnson and Tarjan [24] gave two algorithms for non-rotational 2D SP, called Next-Fit Decreasing Height (NFDH) and First-Fit Decreasing Height (FFDH). NFDH packs items into a strip of height less than $2\operatorname{opt}(I) + h_{\max}$, where $h_{\max}$ is the maximum height of any item in the input. FFDH packs items into a strip of height less than $1.7\operatorname{opt}(I) + h_{\max}$. The approximation guarantee of NFDH also holds for the rotational version, regardless of how we orient the items in the packing. FFDH can be made to work for the rotational version with a bit more work.

Kenyon and Rémila gave an APTAS for non-rotational 2D SP [49]. Jansen and van Stee gave an APTAS for rotation 2D SP [43], under the assumptions that items have width and height at most 1 and that the strip has width equal to 1.

Caprara [18] gave an algorithm for the non-rotational dD SP, called $\texttt{HDH}_k$, that takes a parameter k as input and has an AAR of roughly T_∞^{d-1} for large k (recall that $T_\infty \approx 1.69103$).

Li and Cheng [57] give a 3.25-asymptotic-approximation algorithm for non-rotational 3D SP. Jansen and Solis-Oba [42] improve the AAR to $2+\varepsilon$. Bansal, Han, Iwama, Sviridenko and Zhang [13] improve the AAR to $T_\infty \approx 1.69103$. Jansen and Prädel [40] further improve the AAR to $1.5+\varepsilon$.

Miyazawa and Wakabayashi [60] gave a 2.64-asymptotic-approximation algorithm for 3D SP when items can be rotated about all axes. Epstein and van Stee [31] give a 2.25-asymptotic-approximation algorithm for 3D SP where the base of the strip is a square and the items can be rotated either about all axes or about the height axis only.

2.6 Vector Bin Packing

Lueker and Vega [26] gave a $(d+\varepsilon)$-asymptotic-approximation algorithm for dD VBP.

Chekuri and Khanna [20] showed that if d is not a constant, then for all $\varepsilon > 0$, it is NP-hard to obtain an algorithm for dD VBP having an AAR of $d^{1/2-\varepsilon}$. They showed this by reducing from the graph coloring problem. Bansal, Khan and Elias [12] show how to improve the hardness to $d^{1-\varepsilon}$. Due to these hardness results, we now focus only on the case where d is a constant.

Chekuri and Khanna [20] gave an algorithm for dD VBP (for constant d) having an AAR of $2 + H_{d-1}$, where $H_k := 1 + 1/2 + \ldots + 1/k$ is the k^{th} harmonic number. Using the well-known inequality $H_k \leq \ln(k+1) + \gamma$, where $\gamma \approx 0.5772156649$ is the Euler–Mascheroni constant, we get that the AAR is at most $2 + \gamma + \ln d$.

Bansal, Caprara and Sviridenko [10] improved the AAR to $1 + \ln d + \varepsilon$ by combining the R&A framework with Lueker and Vega's $(d+\varepsilon)$-asymptotic-approximation algorithm.

Bansal, Khan and Elias [12] gave an algorithm for 2D VBP having an AAR of $1+\ln(1.5)+\varepsilon \approx 1.40547+\varepsilon$. They also give an algorithm for dD VBP having an AAR of $(1.5-\ln 2)+\ln(d+1)+\varepsilon$, which improves upon the previous $(1+\ln d+\varepsilon)$-asymptotic-approximation algorithm [10] when $d \geq 5$. Their algorithms also use the R&A framework. They also give an algorithm having an absolute approximation ratio of $3/2 + \varepsilon$, which is tight (hardness of $3/2$ follows from classical bin packing).

Table 2.4: Algorithms for vector bin packing.

Algorithm	Asymptotic Approximation Ratio
Lueker, Vega [26]	$d + \varepsilon$
Chekuri, Khanna [20]	$2 + H_{d-1} \leq 2.57722 + \ln d$
Bansal, Caprara, Sviridenko [10]	$1 + \ln d + \varepsilon$
Bansal, Khan and Elias [12]	$(1.5 - \ln 2) + \ln(d+1) + \varepsilon \approx 0.80685 + \ln(d+1) + \varepsilon$
Bansal, Khan and Elias [12]	$1 + \ln(1.5) + \varepsilon \approx 1.40547 + \varepsilon \quad (\text{for } d = 2)$

Woeginger [80] proved that it is NP-hard to obtain an APTAS for 2D VBP. Recently, Sandeep [69] showed that there exists a constant $c > 0$ such that it is NP-hard to obtain an AAR less than $c \log d$ for dD VBP.

2.7 Vector Knapsack

In the dD vector knapsack problem (dD VKS), we are given a set I of dD vectors, where each vector has a non-negative profit associated with it, and we have to pack a maximum-profit subset of I into a single bin (here the bin is also called *knapsack*).

A PTAS for dD VKS was given by Frieze and Clarke [33].

Chapter 3

Notation and Preliminaries

3.1 Notation

3.1.1 General

For an integer $n \geq 0$, define $[n] := \{1, 2, \ldots, n\}$. Define $\text{poly}(n)$ as the set of polynomial and sub-polynomial functions of n. For a set X, define $|X|$ as the cardinality of the set and define $\text{sum}(X) := \sum_{x \in X} x$.

For a vector $\mathbf{x} \in \mathbb{R}^d$, we denote the i^{th} entry by $\mathbf{x}_i$ or $\mathbf{x}[i]$. Define $\text{sum}(\mathbf{x}) := \sum_{i=1}^{d} \mathbf{x}_i$. Define the support of $\mathbf{x}$ as the set of indices of $\mathbf{x}$ with non-zero value, i.e., $\text{support}(\mathbf{x}) := \{i \in [d] : \mathbf{x}_i \neq 0\}$. The ℓ_p-norm of $\mathbf{x}$, denoted as $\|\mathbf{x}\|_p$, is defined as

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{d} |\mathbf{x}_i|^p \right)^{1/p}.$$

Note that if all entries of $\mathbf{x}$ are non-negative, then $\text{sum}(\mathbf{x}) = \|\mathbf{x}\|_1$.

For a matrix A, we define the entry at the i^{th} row and j^{th} column by $A_{i,j}$ or $A[i, j]$.

3.1.2 Items

Let i be a dD cuboidal item (this means that i is the cross-product of d intervals on the real line). Denote the length of i along the j^{th} dimension by $\ell_j(i)$. The first $d - 1$ dimensions of i are called *base dimensions* and the d^{th} dimension of i is called *height*, denoted by $h(i) := \ell_d(i)$.

The volume of i is defined as

$$\text{vol}(i) := \prod_{i=1}^{d} \ell_j(i).$$

When $d = 2$, we say that i is a rectangular item. Define the width of i as $w(i) := \ell_1(i)$ and the area of i as $a(i) := \text{vol}(i) = w(i)h(i)$. For 1D items (i.e., items in a classical bin packing instance), vol is also called size.

For a set I of items, let $f : I \mapsto \mathbb{R}$ be a function mapping each item to a real number. Then for a set $X \subseteq I$, define $f(X) := \sum_{i \in X} f(i)$ (unless stated otherwise). For example, $\text{vol}(X) := \sum_{i \in X} \text{vol}(i)$.

3.1.3 Packing of Items

Let I be a set of items. Hence, $|I|$ is the number of items in I. Let P be a packing of items into bins. Define $|P|$ as the number of bins used by P. Let $\mathcal{A}$ be an algorithm for bin packing. Define $\mathcal{A}(I)$ as the packing of I output by $\mathcal{A}$. Hence, the number of bins used by $\mathcal{A}$ to pack I is $|\mathcal{A}(I)|$.

Let I be a bin packing instance. Define $\text{opt}(I)$ as the minimum number of bins needed to pack I. Sometimes, we may be considering multiple versions of bin packing, like 2D GBP with rotations (denoted as 2BPR) and 2D GBP without rotations (denoted as 2BPNR). Hence, to disambiguate, we will use the problem name in subscript, e.g., $\text{opt}_{\text{2BPR}}(I)$ and $\text{opt}_{\text{2BPNR}}(I)$. We will sometimes look at other kinds of packing problems, like strip packing (SP) and knapsack (KS). If the problem is clear from context, we will use $\text{opt}(I)$ to denote the objective value of the optimal solution to I. Otherwise, to disambiguate, we will use $\text{opt}_{\text{BP}}(I)$, $\text{opt}_{\text{SP}}(I)$, $\text{opt}_{\text{KS}}(I)$, etc.

3.2 Approximation Algorithms

3.2.1 Minimization Problems

In a minimization problem, there is a set of feasible solutions associated with each input, and each feasible solution has a cost associated with it. Our task is to output a feasible solution of minimum cost for the given input. For an input I, let $\text{opt}(I)$ denote the cost of the minimum-cost feasible solution. The bin packing problem is an example of a minimization problem.

Let $\mathcal{A}$ be an algorithm for the minimization problem. Define $\mathcal{A}(I)$ as the output of $\mathcal{A}$ on

input I. Algorithm $\mathcal{A}$ is said to be α-approximate iff for each input I, the cost of $\mathcal{A}(I)$ is at most $\alpha \operatorname{opt}(I)$. The minimum value α for which $\mathcal{A}$ is α-approximate is called the *approximation ratio* of $\mathcal{A}$. Note that α is always at least 1.

The algorithm $\mathcal{A}$ is called a PTAS (Polynomial-Time Approximation Scheme) iff it takes a constant $\varepsilon > 0$ as a parameter, is $(1+\varepsilon)$-approximate, and runs in time polynomial in n. Note that ε is a constant, so $n^{1/\varepsilon}$ is polynomial in n. A PTAS is called an FPTAS iff it is a PTAS and runs in time polynomial in both n and $1/\varepsilon$.

3.2.2 Maximization Problems

In a maximization problem, there is a set of feasible solutions associated with each input, and each feasible solution has a score. Our task is to output a feasible solution of maximum score for the given input. For an input I, let $\operatorname{opt}(I)$ denote the score of the maximum-score feasible solution. The knapsack problem is an example of a maximization problem.

Let $\mathcal{A}$ be an algorithm for the maximization problem. Define $\mathcal{A}(I)$ as the output of $\mathcal{A}$ on input I. Algorithm $\mathcal{A}$ is said to be α-approximate iff for each input I, the score of $\mathcal{A}(I)$ is at least $\operatorname{opt}(I)/\alpha$. The minimum value α for which $\mathcal{A}$ is α-approximate is called the *approximation ratio* of $\mathcal{A}$. Note that α is always at least 1.

The algorithm $\mathcal{A}$ is called a PTAS (Polynomial-Time Approximation Scheme) iff it takes a constant $\varepsilon > 0$ as a parameter, is $(1+\varepsilon)$-approximate, and runs in time polynomial in n. A PTAS is called an FPTAS iff it is a PTAS and runs in time polynomial in both n and $1/\varepsilon$.

(Some authors use a slightly different definition of α-approximate. By this definition, algorithm $\mathcal{A}$ is α-approximate iff for each input I, the score of $\mathcal{A}(I)$ is at least $\alpha \operatorname{opt}(I)$. Note that α is always at most 1 by this definition.)

3.3 Simple Packing Algorithms

Lemma 3.1. *Let I be a classical bin packing instance. Then $|\text{Next-Fit}(I)| < 2\,\text{size}(I) + 1$. Equivalently, $|\text{Next-Fit}(I)| \leq \lceil 2\,\text{size}(I) \rceil$.*

Proof. Assume I is non-empty (otherwise, the lemma is trivially true). Let $m := |\text{Next-Fit}(I)|$. Let J_j be the items packed into the j^{th} bin by Next-Fit. For $j \leq m-1$, by the definition of Next-Fit, the first item in the $(j+1)^{\text{th}}$ bin didn't fit into the j^{th} bin. Therefore, $\text{size}(J_j) + \text{size}(J_{j+1}) >$

1. This gives us

$$\text{size}(I) = \frac{\text{size}(J_1) + \text{size}(J_m)}{2} + \sum_{j=1}^{m-1} \frac{\text{size}(J_j) + \text{size}(J_{j+1})}{2} > \sum_{j=1}^{m-1} \frac{1}{2} = \frac{m-1}{2}$$

$$\implies m < 2\,\text{size}(I) + 1 \iff m \leq \lceil 2\,\text{size}(I) \rceil\,. \qquad \square$$

Lemma 3.2. *Let I be a classical bin packing instance, where each item has size at most ε. Then $|\,\text{Next-Fit}(I)| \leq \lceil \text{size}(I)/(1-\varepsilon) \rceil$.*

Proof. Assume I is non-empty (otherwise, the lemma is trivially true). Let $m := |\,\text{Next-Fit}(I)|$. Let J_j be the items packed into the j^{th} bin by Next-Fit. For $j \leq m-1$, we have $\text{size}(J_j) > 1-\varepsilon$. Therefore,

$$\text{size}(I) > \sum_{j=1}^{m-1} \text{size}(J_j) \geq (m-1)(1-\varepsilon)$$

$$\implies m < 1 + \frac{\text{size}(I)}{1-\varepsilon} \quad \iff m \leq \left\lceil \frac{\text{size}(I)}{1-\varepsilon} \right\rceil. \qquad \square$$

Lemma 3.3 (NFDH for strip packing [24])**.** *Let I be a set of rectangular items of width at most 1. Then I can be packed (without rotation) into a rectangular bin of width 1 and height less than $2a(I) + max_{i \in I} h(i)$ using the Next-Fit Decreasing Height (NFDH) algorithm.*

Proof. Let there be p shelves output by NFDH. Let S_j be the items in the j^{th} shelf. Let h_j be the height of the j^{th} shelf. Let H be the sum of heights of all the shelves.

For $j \leq p-1$, all items in the j^{th} shelf have height at least h_{j+1}. The combined width of the items in S_j and the first item in S_{j+1} is more than 1. Therefore, $a(S_j) + a(S_{j+1}) > h_{j+1}$. This gives us

$$a(I) = \frac{a(S_1) + a(S_p)}{2} + \sum_{j=1}^{p-1} \frac{a(S_j) + a(S_{j+1})}{2} > \sum_{j=1}^{p-1} \frac{h_{j+1}}{2} = \frac{H - h_1}{2}$$

$$\implies H < 2a(I) + h_1 = 2a(I) + \max_{i \in I} h(i). \qquad \square$$

Lemma 3.4 (NFDH for small items [24])**.** *Let I be a set of rectangular items where each item has width at most δ_W and height at most δ_H. Let there be a rectangular bin of width W and height H. If $a(I) \leq (W - \delta_W)(H - \delta_H)$, then the Next-Fit Decreasing Height (NFDH) algorithm can pack I into the bin (without rotating the items).*

Proof. NFDH packs the items into shelves of width W. Let the number of shelves be p. Let h_j be the height of the j^{th} shelf. Let S_j be the items in the j^{th} shelf. Let $\widetilde{H}$ be the total height of the shelves. We need to prove that $\widetilde{H} \leq H$.

For $j \leq p-1$, all items in the j^{th} shelf have height at least h_{j+1}, and the total width of the items in S_j is more than $W - \delta_W$, because otherwise we could have fit another item into the j^{th} shelf. Therefore, $a(S_j) > (W - \delta_W)h_{j+1}$ for $j \leq p-1$. This gives us

$$(W-\delta_W)(H-\delta_H) \geq a(I) > \sum_{j=1}^{p-1} a(S_j) \geq (W-\delta_W)\sum_{j=1}^{p-1} h_{j+1} = (W-\delta_W)(\widetilde{H}-h_1)$$
$$\implies H - \delta_H > \widetilde{H} - h_1 \quad \implies \widetilde{H} < H - (\delta_H - h_1) \leq H. \qquad \square$$

Lemma 3.5 (NFDH for bin packing)**.** *Let I be a set of rectangular items of width and height at most 1. Then the number of square bins of side length 1 that Next-Fit Decreasing Height (NFDH) uses to pack I is less than $4a(I) + 3$.*

Proof. The bin packing version of NFDH first packs I into shelves and then packs the shelves into bins using Next-Fit. Let H be the sum of heights of all the shelves.

By Lemma 3.3, we get $H < 2a(I) + 1$. By Lemma 3.1, the number of bins needed is less than $2H + 1$. Therefore, the number of bins needed is less than $4a(I) + 3$. $\square$

3.4 Configuration Linear Program

Let I be a bin packing instance. We will now describe a linear program (LP) associated with I, called the *configuration LP*. Configuration LPs are defined for a large class of bin packing problems, like classical bin packing, geometric bin packing and vector bin packing.

For a set I of items, define a configuration of I as a non-empty subset of I that can fit into a bin, and a way of packing those items into a bin. An equivalent definition of bin packing is that we need to find a small number of configurations such that each item belongs to at least one of those configurations.

Let C be a configuration. We say that $i \in C$ iff item i belongs to the set of items corresponding to C. We can denote each configuration as a vector $C \in \{0,1\}^n$, where $C_i = 1$ iff $i \in C$. Let $\mathcal{C}$ be the set of all possible configurations.

We can express bin packing as the problem of finding integer solutions to the following linear program, called the configuration LP, where $x_C \in \{0,1\}$ denotes the number of bins that have

configuration C:

$$\begin{aligned} \min_{x \in \mathbb{R}^{|\mathcal{C}|}} \quad & \sum_{C \in \mathcal{C}} x_C \\ \text{where} \quad & \sum_{C \ni i} x_C \geq 1 \quad \forall i \in [n] \\ \text{and} \quad & x_C \geq 0 \qquad \forall C \in \mathcal{C} \end{aligned}$$

Define the configuration matrix $A \in \{0,1\}^{n \times |\mathcal{C}|}$ as a matrix where $A[i, C]$ is 1 if configuration C contains item i and 0 otherwise. We can also write the configuration LP as

$$\min_{x \in \mathbb{R}^{|\mathcal{C}|}} \text{sum}(x) \quad \text{where} \quad Ax \geq \mathbf{1} \quad \text{and} \quad x \geq 0.$$

Linear programs are a useful tool in the area of approximation algorithms. Many optimization problems can be expressed as integer programs. *Rounding-based* algorithms first solve the LP relaxation of these integer programs, and then *round* the relaxed solution to get an approximate solution to the original problem [77, 78, 54]. This approach has also been successfully applied to bin packing: the Round-and-Approx framework [10, 14] applies randomized rounding to a solution to the configuration LP.

We would like to solve the configuration LP in polynomial time. However, the number of configurations can be up to $2^n - 1$, which is exponential in the input size. One may wonder what it would even mean to solve the LP in polynomial time, since even writing down a feasible solution to the LP can take exponential time! The key observation (which we will soon show) is that there exists an optimal solution to the configuration LP whose support contains at most n configurations. Therefore, we can restrict ourselves to solutions of poly(n)-sized support without loss of generality. Such solutions can be written down in polynomial time.

Definition 3.1 (Extreme point; Definition 1.2.1 in [54])**.** *Let $P \subseteq \mathbb{R}^d$ and $x \in P$. x is called an extreme point of P iff there is no $y \neq 0$ such that $x + y \in P$ and $x - y \in P$.*

Lemma 3.6. *If a linear program is bounded (i.e., it attains the optimal objective value at some point), then it contains an optimal solution that is an extreme point of the set of all feasible solutions.*

Lemma 3.7 (Rank Lemma; Lemma 2.1.4 in [54])**.** *Let $P := \{x : Ax \geq b \text{ and } x \geq 0\}$. Let $\widehat{x}$ be an extreme point of P. Let $S := \{a_i : (A\widehat{x})_i = b_i\}$, where a_i is the i^{th} row of A. Then $|\text{support}(\widehat{x})|$ is equal to the maximal number of linearly independent vectors in S.*

Corollary 3.8. *Let* $P := \{x : Ax \geq b \text{ and } x \geq 0\}$ *and let* $\widehat{x}$ *be an extreme point of* P*. Then* $|\operatorname{support}(\widehat{x})|$ *is at most the number of inequalities in* $Ax \geq b$*.*

By Corollary 3.8, we get that there exist extreme point optimal solutions to the configuration LP, and these solutions have at most n configurations in their support.

3.4.1 Solving the Configuration LP

Karmarkar and Karp [48] gave an FPTAS for solving the configuration LP of any classical bin packing instance. They do this by slightly modifying the Ellipsoid algorithm of Grötschel, Lovász and Schrijver [36].

The algorithm of Plotkin, Shmoys and Tardos [65] can be used to obtain a PTAS for the configuration LP of any variant of bin packing, given a PTAS for the corresponding knapsack problem. Bansal, Caprara, Jansen, Prädel and Sviridenko [9] showed how to use this to solve the configuration LP for 2D GBP (both the rotational and non-rotational versions).

Chapter 4

Generalized Multidimensional Bin Packing

In this chapter, we study the $(d_g,\ d_v)$ BP problem, also known as the generalized multidimensional bin packing problem. This problem generalizes both d_g-dimensional geometric bin packing and d_v-dimensional vector bin packing. Here d_g and d_v are constants. See Section 1.2.1 for a detailed introduction to this problem and its significance.

Overview

- In Section 4.1, we describe some preliminaries related to $(d_g,\ d_v)$ BP.
- In Section 4.2, we give two simple algorithms for $(2,\ d)$ BP, called `simplePack` and `betterSimplePack`, having AARs of $6(d+1)$ and $3(1+\ln(d+1))+\varepsilon$, respectively, for any $\varepsilon > 0$. For $d = 1$, `betterSimplePack`'s AAR improves to $\approx 4.21640 + \varepsilon$. We also extend these algorithms to $(d_g,\ d_v)$ BP.
- In Section 4.3, we describe our version of the Round-and-Approx (R&A) framework and show how it can be applied to `simplePack`. We defer some formal proofs related to R&A to Section 4.4.
- In Chapter 5, we describe a more sophisticated algorithm for $(2,\ d)$ BP, called `cbPack`, that improves upon the algorithms of this chapter.

4.1 Preliminaries

For $(d_g,\, d_v)$-dimensional items, define $\mathrm{v}_{\max}(i) := \max_{j=1}^{d_v} v_j(i)$. For convenience, let $v_0(i) := \mathrm{vol}(i)$. Define $\mathrm{span}(i) := \max(\mathrm{vol}(i), \mathrm{v}_{\max}(i)) = \max_{j=0}^{d_v} v_j(i)$. $\mathrm{span}(i)$ is, intuitively, the measure of *largeness* of item i. See Section 3.1.2 to recall the definition of $\mathrm{vol}(X)$, $\mathrm{v}_{\max}(X)$ and $\mathrm{span}(X)$, where X is a set of $(d_g,\, d_v)$-dimensional items.

Note that unlike geometric packing problems, we cannot trivially handle items of zero volume in $(d_g,\, d_v)$ BP. Assume without loss of generality that $\mathrm{vol}(i) = 0$ implies $(\forall j \in [d_g], \ell_j(i) = 0)$.

For simplicity, throughout this chapter, we will assume that the bins are squares of side length 1. This assumption isn't true when items can be rotated, but our algorithms `simplePack` and `betterSimplePack` can be easily extended to handle rotation and non-square bins.

Lemma 4.1. *For $(d_g,\, d_v)$ items I, $\lceil \mathrm{span}(I) \rceil \le (d_v{+}1)\,\mathrm{opt}(I)$. This holds for both the rotational and non-rotational versions.*

Proof. Let $J_1, \ldots, J_m$ be an optimal bin packing of I. Therefore,

$$\begin{aligned}\lceil \mathrm{span}(I) \rceil &= \left\lceil \sum_{k=1}^{m} \sum_{i \in J_k} \max_{j=0}^{d_v} v_j(i) \right\rceil \le \left\lceil \sum_{k=1}^{m} \sum_{i \in J_k} \sum_{j=0}^{d_v} v_j(i) \right\rceil = \left\lceil \sum_{k=1}^{m} \sum_{j=0}^{d_v} \sum_{i \in J_k} v_j(i) \right\rceil \\ &\le \left\lceil \sum_{k=1}^{m} \sum_{j=0}^{d_v} 1 \right\rceil = (d_v + 1)m.\end{aligned}$$

□

Lemma 4.2. *For $(d_g,\, d_v)$ items I, $\lceil \mathrm{v}_{\max}(I) \rceil \le d_v\,\mathrm{opt}(I)$. This holds for both the rotational and non-rotational versions.*

Proof sketch. In the proof of Lemma 4.1, replace $j \in \{0, \ldots, d_v\}$ by $j \in \{1, \ldots, d_v\}$. □

4.2 Simple Algorithms

In this section, we look at simple algorithms for $(2,\, d)$ BP and $(d_g,\, d_v)$ BP. We will not rotate the items while packing them. Nevertheless, we will show that our approximation guarantees hold for both the rotational and non-rotational versions.

4.2.1 Steinberg's Algorithm

A key ingredient in our algorithms for $(2,\, d)$ BP is Steinberg's algorithm [74] for rectangle packing.

Lemma 4.3 (Steinberg's algorithm [74])**.** *Let I be a set of rectangles. Let $w_{\max} := \max_{i \in I} w(i)$ and $h_{\max} := \max_{i \in I} h(i)$. Consider a bin of width W and height H, where $w_{\max} \leq W$ and $h_{\max} \leq H$. Then there is a $O(n \log^2 n / \log \log n)$-time algorithm to pack I into the bin if*

$$2a(I) \leq WH - \max(2w_{\max} - W, 0) \cdot \max(2h_{\max} - H, 0).$$

Lemma 4.4. *Let I be a set of rectangles, where $a(I) \leq 1$. Then there is a $O(n \log^2 n / \log \log n)$-time algorithm to pack I into 3 square bins of side length 1.*

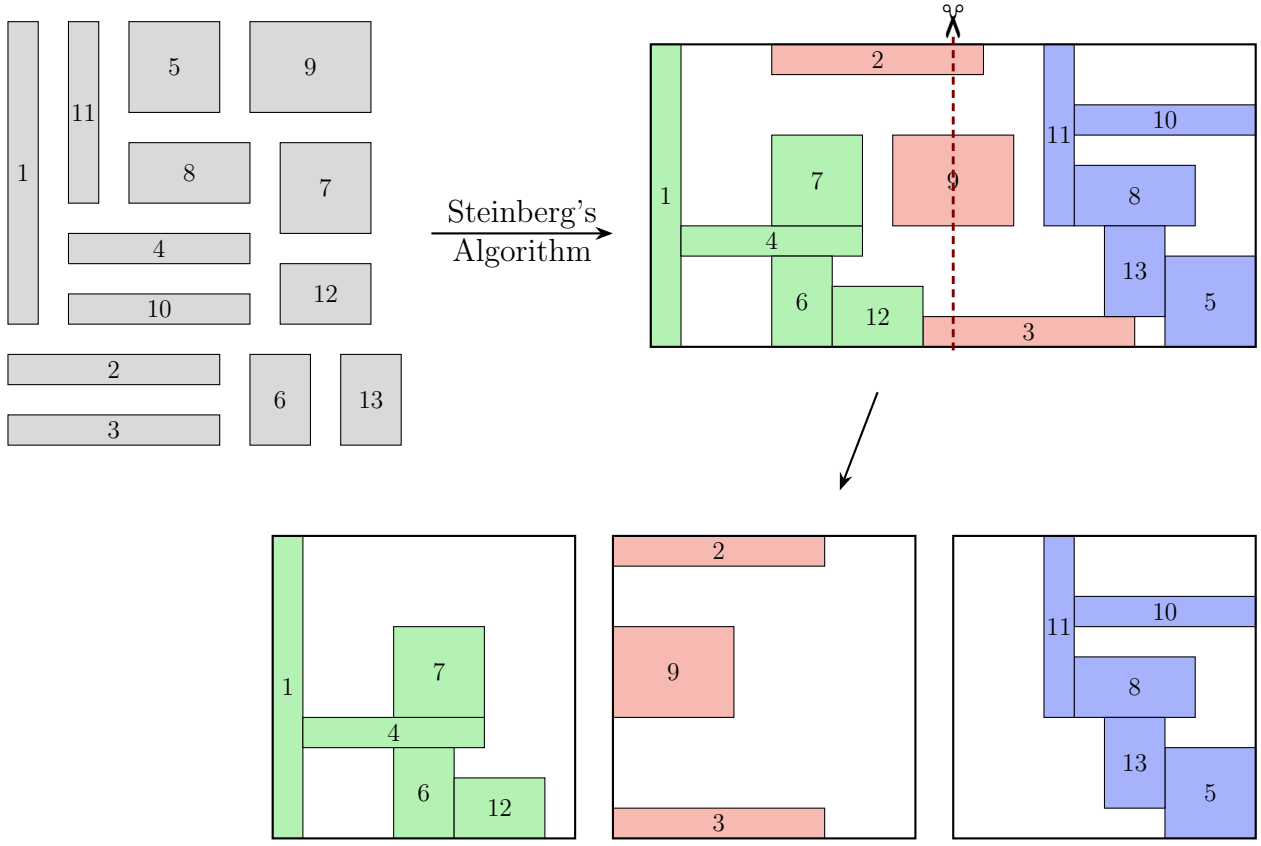


Figure 4.1: Packing items of area at most 1 into three square bins: First use Steinberg's algorithm to pack the items into a bin of width 2 and height 1. Then make a vertical cut in the middle of the bin and use that to split the items into three bins.

Proof. Pack I into a bin of width 2 and height 1 using Steinberg's algorithm (see Lemma 4.3). Then cut the bin vertically into 2 unit-sized squares. The items which lie completely inside the left half can be packed into a unit-sized bin. The items which lie completely inside the right half can be packed into a unit-sized bin. The items which lie on the cutting line are stacked one-over-the-other, so we can pack them into a unit-sized bin. See Fig. 4.1 for an example. □

4.2.2 Algorithms `simplePack` and `betterSimplePack`

Let I be a (2, d) BP instance. Let $\widehat{I} := \{\text{span}(i) : i \in I\}$, i.e., $\widehat{I}$ is a classical bin packing instance. The algorithm `simplePack`(I) first runs the Next-Fit algorithm on $\widehat{I}$. Let $[\widehat{J}_1, \widehat{J}_2, \ldots, \widehat{J}_m]$ be the resulting bin packing of $\widehat{I}$ into m bins. For each $\widehat{J}_j \subseteq \widehat{I}$, let J_j be the corresponding items from I. Then $\forall k \in [d_v]$, $v_k(J_j) \leq 1$ and $\text{vol}(J_j) \leq 1$. `simplePack` then uses Steinberg's algorithm to pack each J_j into at most 3 bins, giving a packing of I into at most $3m$ bins.

By the property of Next-Fit (see Lemma 3.1 in Section 3.3), we get that $m \leq \lceil 2\,\text{span}(I) \rceil$. By Lemma 4.1, we get $3m \leq 6(d+1)\,\text{opt}(I)$. This gives us the following theorem.

Theorem 4.5. *For (2, d) BP,* `simplePack`(I) *uses at most* $3\lceil 2\,\text{span}(I) \rceil$ *bins, so* `simplePack` *is a* $6(d+1)$*-approximation algorithm.* `simplePack` *runs in* $O(nd + n \log^2 n / \log\log n)$ *time, where* $n := |I|$.

The algorithm `betterSimplePack` first computes $\widetilde{I}$, which is a $(d+1)$D VBP instance obtained by replacing the geometric dimensions of each item $i \in I$ by a single vector dimension $a(i)$. It computes a bin packing of $\widetilde{I}$ using any algorithm $\mathcal{A}$. It then uses Steinberg's algorithm to obtain a bin packing of I into at most $3|\mathcal{A}(\widetilde{I})|$ bins.

Note that $\text{opt}(\widetilde{I}) \leq \text{opt}(I)$. If $\mathcal{A}$ has an AAR of α, then $|\mathcal{A}(\widetilde{I})| \leq \alpha\,\text{opt}(\widetilde{I}) + O(1)$. Therefore, `betterSimplePack` has an AAR of 3α. The $(d+1)$D VBP algorithm by Bansal, Caprara and Sviridenko [10] (parametrized by a constant $\varepsilon > 0$) gives $\alpha = 1 + \ln(d+1) + \varepsilon$ and the algorithm by Bansal, Elias and Khan [12] gives $\alpha = 1.5 + \ln((d+2)/2) + \varepsilon$ (improves to $\alpha = 1 + \ln(1.5) + \varepsilon$ for $d = 1$).

Although `simplePack` has a worse AAR than `betterSimplePack`, the number of bins used by `simplePack` is upper-bounded in terms of span, which is a useful property. Because of this, we will use it as a subroutine in other algorithms (like `cbPack`).

Since $|$`simplePack`$(I)|$ is upper-bounded in terms of $\text{span}(I)$, and $|$`betterSimplePack`$(I)|$ is upper-bounded in terms of $\text{opt}(\widetilde{I})$, their approximation guarantees hold for both the rotational and non-rotational versions of (2, d) BP.

4.2.3 Extending to Higher Geometric Dimensions

The algorithms for (2, d) BP can be extended to (d_g, d_v) BP. We just need an algorithm for the following problem: *given a set J of d_g-dimensional cuboids where* $\text{vol}(J) \leq 1$, *pack J into a small number of bins.*

We used Steinberg's algorithm when $d_g = 2$. When $d_g = 3$, we can use the algorithm of [27, Section 2] to pack J into at most 9 bins. For $d_g > 3$, we can use the `fullh`$_4$ algorithm, which

is a variant of Caprara's $\texttt{HDH}_k$ algorithm [18], to pack J into at most $4^{d_g} + 2^{d_g} - 1$ bins. We fully describe and analyze $\texttt{fullh}_k$ in Section 6.2, but we give a smaller, self-contained analysis in Section 4.5.

Therefore, `simplePack` will use $b\lceil 2\,\mathrm{span}(I)\rceil$ bins, where $b := 3$ when $d_g = 2$, $b := 9$ when $d_g = 3$, and $b := 4^{d_g} + 2^{d_g} - 1$ when $d_g > 3$. Hence, `simplePack` is $2b(d_v + 1)$-approximate. Similarly, the AAR of `betterSimplePack` is $b(1 + \ln(d_v + 1) + \varepsilon)$.

4.2.4 Simple Algorithm for the Knapsack Problem

Using similar ideas, we can get an algorithm for the (d_g, d_v) KS problem. Let I be a set of (d_g, d_v)-dimensional items. Let $p(i)$ be the profit of item i. We want to pack a maximum-profit subset of I into a bin.

Let $\widetilde{I}$ be a set of $(d_v + 1)$D vectors obtained by replacing the geometric dimensions of each item i by a single vector dimension $\mathrm{vol}(i)$. Let $\mathcal{A}$ be a $(d_v + 1)$D vector knapsack (VKS) algorithm having approximation ratio $\alpha \geq 1$. $\mathcal{A}$ gives us a packing of items $\widehat{J} \subseteq \widetilde{I}$ into a bin. Let J be the corresponding items in I. Then $\mathrm{vol}(J) \leq 1$ and $\forall k \in [d_v], v_k(J) \leq 1$. We can pack J into at most b bins, where $b = 3$ for $d_g = 2$ (by Steinberg's algorithm), $b = 9$ for $d_g = 3$ (by [27]), and $b = 4^{d_g} + 2^{d_g} - 1$ for $d_g > 3$ (by the $\texttt{fullh}_4$ algorithm).

Let $J_1, J_2, \ldots, J_b$ be the bins that J is packed into. Without loss of generality, assume $p(J_1) \geq p(J_2) \geq \ldots \geq p(J_b)$. Then output the packing J_1 as the answer to the (d_g, d_v) KS problem. Since any feasible solution to the (d_g, d_v) KS instance I also gives a feasible solution to the VKS instance $\widetilde{I}$, we get $\mathrm{opt}_{\mathrm{KS}}(\widetilde{I}) \geq \mathrm{opt}_{\mathrm{KS}}(I)$. Since $\mathcal{A}$ is α-approximate, we get $p(J) \geq \mathrm{opt}(\widetilde{I})/\alpha$. Hence,

$$p(J_1) \geq \frac{p(J)}{b} \geq \frac{\mathrm{opt}(\widetilde{I})}{b\alpha} \geq \frac{\mathrm{opt}(I)}{b\alpha}.$$

Therefore, we get a $b\alpha$-approximation algorithm for (d_g, d_v) KS. Using the PTAS for $(d_v + 1)$D VKS by Frieze and Clarke [33], we get $\alpha = 1 + \varepsilon$.

4.3 Round-and-Approx Framework

The R&A framework is a simple but powerful technique, originally given by Bansal, Caprara and Sviridenko [10], to improve the AAR of a bin packing algorithm by combining it with randomized rounding of the configuration LP (recall the definition of configuration LP from Section 3.4). R&A may have the potential to improve the AARs of several packing problems, but

its applicability is limited because it only works with *subset-oblivious* bin packing algorithms, and proving that an algorithm is subset-oblivious is difficult.

Bansal and Khan [14] partially removed this limitation by proving that a large class of algorithms for geometric and vector bin packing, called *rounding-based* algorithms, is subset-oblivious. We make further improvements on this front by showing that an even larger class of algorithms is subset-oblivious. This class includes some of our algorithms for (d_g, d_v) BP, like `simplePack` and `cbPack`. Algorithms in this class are characterized as a combination of three simpler subroutines, called `round`, `complexPack` and `unround`.

We describe our version of the R&A framework as a *meta-algorithm*, i.e., the R&A framework takes four subroutines as input—`solveConfigLP`, `round`, `complexPack` and `unround`—and returns an algorithm for bin packing, called `rnaPack`. `rnaPack`(I) first uses `solveConfigLP` to (approximately) solve the configuration LP of I. Based on the (approximate) solution to the configuration LP, it packs a large fraction of the items in I. It then packs the remaining items using the subroutines `round`, `complexPack` and `unround`. We prove that if `round`, `complexPack` and `unround` satisfy some special properties, then `rnaPack` has a low AAR.

In Section 4.3.2, we describe the `rnaPack` algorithm in detail. In Section 4.3.1, we highlight the differences between our version of R&A and the previous versions of R&A [10, 14]. In Sections 4.3.3, 4.3.4, 4.3.5 and 4.3.6, we describe the subroutines `round`, `complexPack` and `unround`, and mention the properties that they should satisfy. In Section 4.3.7, we prove an upper-bound on the AAR of `rnaPack`. In Section 4.3.8, we show how to apply the R&A framework to the `simplePack` algorithm.

4.3.1 Comparison to Previous Versions of R&A

To use R&A, we need to solve the configuration LP. All previous applications (dD VBP and 2D GBP) of R&A solved the configuration LP $(1+\varepsilon)$-approximately using a $(1+O(\varepsilon))$-approximate solution to the corresponding knapsack problem. Due to the unavailability of a PTAS for $(2, d)$ KS, we had to adapt and use a different linear programming algorithm [72] that uses an η-approximation algorithm for the knapsack problem to $(1+\varepsilon)\eta$-approximately solve the configuration LP, for any constants $\eta > 1$ and $0 < \varepsilon < 1$.

Second, we introduce more freedom in choosing the packing structure. Unlike the R&A framework in [14], that worked only for container-based packing, we allow either relaxing the packing structure to non-container-based (like in `simplePack`) or imposing packing constraints in addition to being container-based (like in `cbPack`). This generalization can help in finding better algorithms for other variants of bin packing.

The *rounding-based* algorithms in [14] work by rounding up the large dimensions of items to $O(1)$ different types. In addition, we also allow *rounding down* some dimensions, if we can find a suitable way of *unrounding* a packing of rounded items. For example, in `cbPack`, we round down the width and height of some items to 0. It was shown in [50] that if the large coordinates of items are rounded to $O(1)$ types, we cannot obtain AARs better than d and 4/3 for dD VBP and 2D GBP, respectively. However, as we now allow rounding down, we may be able to use the R&A framework with algorithms having better approximation ratios.

We also fix a minor error in the R&A framework of [14] (see Section 4.4.1 for details).

4.3.2 Description of the R&A Algorithm

The algorithm `rnaPack`(I, β, ε) takes as input a set I of (d_g, d_v)-dimensional items and parameters $\beta \geq 1$ and $\varepsilon \in (0, 1)$. The steps of the algorithm are as follows (see Algorithm 1 for a more formal description).

1. **Solve the Configuration LP** of I. Use `solveConfigLP` to obtain a μ-asymptotic-approximate solution $\widehat{x}$ to the configuration LP of I. Note that each index of $\widehat{x}$ corresponds to a configuration.

2. **Randomized rounding of configuration LP**: For $T := \lceil (\ln \beta) \|\widehat{x}\|_1 \rceil$ steps, do the following: Select a configuration C with probability $\widehat{x}_C / \|\widehat{x}\|_1$. Pack T bins according to each of these selected T configurations. Let S be the remaining items that are not packed, called the *residual instance*.

3. **Rounding of items**: We define a subroutine `round` that takes items I and parameter ε as input[1]. It *discards* a set $D \subseteq I$ of items such that $\text{span}(D) \leq \varepsilon \, \text{span}(I)$ and then *modifies* each item in $I - D$ to get a set $\widetilde{I}$ of items. We denote the output of `round`(I, ε) as $(\widetilde{I}, D)$, where items in $\widetilde{I}$ are called *rounded items*. Intuitively, after rounding, the items in $\widetilde{I}$ are of $O(1)$ types, which makes packing easier. Also, since $\text{span}(D)$ is small, $D \cap S$ can be packed into a small number of bins using `simplePack`.

   We impose some restrictions on `round`, which we denote as conditions C1 and C2, that we describe in Section 4.3.4. We also allow `round` to output a $O(\text{poly}(n))$-sized list of guesses of $(\widetilde{I}, D)$.

[1]The input to `round` is I instead of S because S is random and we want to round items deterministically, i.e., the rounding of each item $i \in S$ should not depend on which other items from I lie in S. In fact, this is where the old R&A framework [50] introduced an error. See Section 4.4.1 for details.

4. **Pack rounded items**: Let $\widetilde{S}$ be the rounded items corresponding to $S - D$. Pack $\widetilde{S}$ into bins using any bin packing algorithm that satisfies 'condition C3', which we describe in Section 4.3.5. Let us name this algorithm `complexPack`.

5. **Unrounding**: Given a bin packing of $\widetilde{S}$, let `unround` be a subroutine that computes a bin packing of $S - D$. `unround` is trivial in previous versions of R&A, because they only increase dimensions of items during rounding. In our applications, we may round down items, so `unround` can be non-trivial. `unround` can be any algorithm that satisfies 'condition C4', which we describe in Section 4.3.6.

Algorithm 1 `rnaPack`(I, β, ε): Computes a bin packing of I. I is a set of (d_g, d_v)-dimensional items and $\beta \geq 1$

$\widehat{x} =$ `solveConfigLP`(I)
repeat $T := \lceil (\ln \beta) \|\widehat{x}\|_1 \rceil$ **times**
    Select a configuration C with probability $\widehat{x}_C / \|\widehat{x}\|_1$.
    Pack a bin according to C.
end repeat
Let S be the unpacked items from I. // *S is called the set of* residual items.
Initialize J_{best} to `null`.
for $(\widetilde{I}, D) \in$ `round`(I) **do** // `round`(I) *outputs a set of pairs.*
    $J_D =$ `simplePack`$(S \cap D)$
    Let π be a bijection from $I - D$ to $\widetilde{I}$. Let $\widetilde{S} := \{\pi(i) : i \in S - D\}$.
    $\widetilde{J} =$ `complexPack`$(\widetilde{S})$
    $J =$ `unround`$(\widetilde{J})$
    if J_{best} is `null` or $|J_D \cup J| < |J_{\text{best}}|$ **then**
        $J_{\text{best}} = J_D \cup J$
    end if
end for
Pack S according to J_{best}.

The R&A framework requires that `round`, `complexPack` and `unround` satisfy four conditions C1, C2, C3, C4, which we describe in Sections 4.3.4, 4.3.5 and 4.3.6. Prospective users of the R&A framework need to design these three subroutines and prove that they satisfy these four conditions. In Section 4.3.7, we prove that if these conditions are satisfied, then `rnaPack` has a small AAR.

Intuitively, `rnaPack` first packs some items into T bins using randomized rounding of $\widehat{x}$. We can prove that $\Pr[i \in S] \leq 1/\beta$, so S contains a small fraction of the items in I (see Lemma 4.8 in Section 4.4). We will then try to prove that if the rest of the algorithm

(`round` + `complexPack` + `unround`) packs I into m bins, then it will pack S into roughly m/β bins. This notion was referred to in [10] as *subset-obliviousness*. We will use subset-obliviousness to bound the AAR of `rnaPack`.

Section 4.3.8 shows how `simplePack` can be broken up into `round`, `complexPack` and `unround` and used with the R&A framework.

4.3.3 Fractional structured packing

Let $(\widetilde{I}, D)$ be an output of `round`(I) and let $\widetilde{X}$ be an arbitrary subset of $\widetilde{I}$. Our analysis of `rnaPack` is based around a concept called *fractional structured packing* of $\widetilde{X}$. Note that the notion of fractional structured packing only appears in the analysis of `rnaPack`. It is not needed to describe the algorithm.

We first define what it means to slice an item. From a geometric perspective, slicing an item perpendicular to the k^{th} dimension means cutting the item into 2 parts using a hyperplane perpendicular to the k^{th} axis. E.g., for $d_g = 2$, if $k = 1$ for item i, then we slice i using a vertical cut, and if $k = 2$, we slice i using a horizontal cut. The vector dimensions get split proportionately across the slices.

Definition 4.1 (Slicing an item)**.** *Let i be a (d_g, d_v)-dimensional item. Slicing i perpendicular to geometric dimension k with proportionality α (where $0 < \alpha < 1$) is the operation of replacing i by two items i_1 and i_2 such that: (i) $\forall j \neq k, \ell_j(i) = \ell_j(i_1) = \ell_j(i_2)$, (ii) $\ell_k(i_1) = \alpha\ell_k(i)$ and $\ell_k(i_2) = (1-\alpha)\ell_k(i)$, (iii) $\forall j \in [d_v], v_j(i_1) = \alpha v_j(i) \wedge v_j(i_2) = (1-\alpha)v_j(i)$.*

Definition 4.2 (Fractional packing)**.** *Let $\widetilde{I}$ be (d_g, d_v)-dimensional items, where for each item $i \in \widetilde{I}$, we are given a set $X(i)$ of axes perpendicular to which we can repeatedly slice i ($X(i)$ can be empty, which would mean that the item i cannot be sliced). If we slice items as per their given axes and then pack the slices into bins, then the resulting packing is called a* fractional *bin packing.*

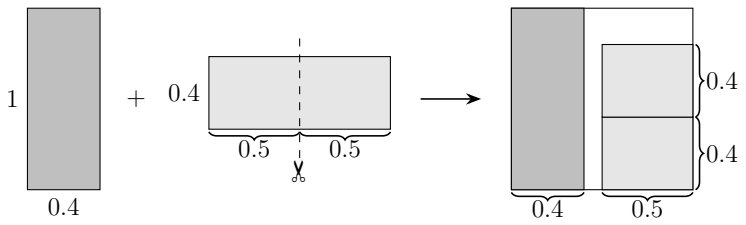


Figure 4.2: Example of a fractional packing of two items into a bin.

We will now review a common approach used in the design of approximation algorithms for packing problems, which we will use to define *fractional structured packing*.

Suppose we wanted to use a brute-force algorithm for bin packing. Such an algorithm would enumerate all possible packings of the items into bins, and pick the packing that required the minimum number of bins. Of course, such an approach wouldn't work, since the number of possible packings is exponential. So, instead of enumerating all possible packings, we will only consider a small subset of packings. We will do this by carefully choosing a set of constraints, and we will only consider packings that satisfy those constraints. We call such packings *structured*. By carefully choosing the constraints, we hope that the optimal structured packing would be a good approximation to the optimal packing, and that the optimal structured packing would be easier to find than the optimal packing.

This idea of using structured packings isn't always enough by itself. However, this idea has been successfully combined with the idea of fractional packing. For example, Jansen and Prädel [41] showed that given any packing of a 2D GBP instance into m bins, we can slice (a carefully chosen subset of) the items and then repack the items into $(1.5+\varepsilon)m+O(1)$ bins such that the resulting packing is *container-based*. *Container-based* roughly means that in each bin, items are packed into rectangular regions called containers, and containers' heights and widths belong to a fixed set of $O(1)$ values. In their algorithm, they find an almost-optimal fractional container-based packing of the input, and show how to convert such a fractional packing into a non-fractional packing without increasing the number of bins by too much.

This approach of using fractional packing together with structured packing has been used in many approximation algorithms for bin packing [41, 15] and knapsack [35]. Each of these algorithms uses a different definition of *structured*. For example, according to Jansen and Prädel's algorithm, a packing is structured iff it is container-based. The common idea in these algorithms is to first show that the optimal *fractional structured packing* is a good approximation to the optimal packing and then find a non-fractional packing that is roughly as good as the optimal fractional structured packing. We would like to formalize this idea and see if we can show that such algorithms are subset-oblivious.

Formally, a fractional configuration of $\widetilde{I}$ is a packing of slices of some items from $\widetilde{I}$ into a bin. Let $\mathcal{S}$ be a set of fractional configurations of $\widetilde{I}$. Intuitively, $\mathcal{S}$ is the set of all structured fractional configurations. For a set $\widetilde{X} \subseteq \widetilde{I}$, a fractional bin packing of $\widetilde{X}$ is said to be *structured* (with respect to $\mathcal{S}$) iff the configuration of each bin in the packing belongs to $\mathcal{S}$.

We will assume that $\mathcal{S}$ is *downward-closed*. Intuitively, downward-closed means that if we remove some items from a structured packing, the packing will remain structured. Formally, for two fractional configurations C_1 and C_2, we say that C_1 is a subconfiguration of C_2 iff we

can remove some items (or slices thereof) from C_2 to get C_1. We say that $\mathcal{S}$ is downward-closed iff for each $C \in \mathcal{S}$, all subconfigurations of C also lie in $\mathcal{S}$. This assumption is necessary in our analysis of `rnaPack`, but this is a very mild assumption, since all published examples of structured packing that we have come across are downward closed.

The R&A framework of Bansal and Khan [14], only worked with bin packing algorithms that used 'container-based' as their definition of structured packing. Our R&A framework, on the other hand, gives algorithm designers the freedom to define the notion of structured packing (i.e., deciding on $\mathcal{S}$) in any way they want, as long as it satisfies the downward closure property. Typically, the choice of which definition of structured packing to use will depend on the ease of proving Conditions C2 and C3 (which we describe in Sections 4.3.4 and 4.3.5) for that definition.

Define $\text{fsopt}(\widetilde{X})$ as the number of bins used in the optimal fractional structured packing of $\widetilde{X} \subseteq \widetilde{I}$. To analyze the AAR of `rnaPack`, we will bound the number of bins used to pack the residual instance S in terms of $\text{fsopt}(\widetilde{S})$, and then we will bound $\text{fsopt}(\widetilde{S})$ in terms of $\text{opt}(I)$.

4.3.4 Properties of `round`

Definition 4.3 (Density vector)**.** *The density vector of a (d_g, d_v)-dimensional item i is the vector* $v_{\text{span}} := [v_0(i)/\text{span}(i), v_1(i)/\text{span}(i), \ldots, v_{d_v}(i)/\text{span}(i)]$. *Recall that* $v_0(i) := \text{vol}(i)$ *and note that* $\|v_{\text{span}}\|_\infty = 1$.

The subroutine `round`(I) returns a set of pairs of the form $(\widetilde{I}, D)$. **Condition C1** is defined as the following constraints over each pair $(\widetilde{I}, D)$:

- **C1.1.** *Small discard:* $D \subseteq I$ and $\text{span}(D) \leq \varepsilon\, \text{span}(I)$.
- **C1.2.** *Bijection from $I - D$ to $\widetilde{I}$:* Each item in $\widetilde{I}$ is obtained by modifying an item in $I - D$. Let π be the bijection from $I - D$ to $\widetilde{I}$ that captures this relation.
- **C1.3.** *Homogeneity properties:* `round` partitions items in $\widetilde{I}$ into a constant number of classes: $\widetilde{K}_1, \widetilde{K}_2, \ldots, \widetilde{K}_q$. These classes should satisfy the following properties, which we call *homogeneity* properties:
  - All items in a class have the same density vector.
  - For each class $\widetilde{K}_j$, we decide the set X of axes perpendicular to which we can slice items in $\widetilde{K}_j$. If items in a class $\widetilde{K}_j$ are not allowed to be sliced perpendicular to dimension k, then all items in that class have the same length along dimension k.

(For example, if $d_g = 2$ and vertical cuts are forbidden, then all items have the same width.)

- **C1.4.** *Bounded expansion:* Let C be any configuration of I and $\widetilde{K}$ be any one of the classes of $\widetilde{I}$. Let $\widetilde{C} := \{\pi(i) : i \in C - D\}$. Then we need to prove that $\text{span}(\widetilde{K} \cap \widetilde{C}) \leq c_{\max}$ for some constant $c_{\max}$. Let us call this result 'bounded expansion lemma'.

Intuitively, the homogeneity properties allow us to replace (a slice of) an item in a fractional packing by slices of other items of the same class. Thus, while trying to get a fractional packing, we can focus on the item classes, which are constant in number, instead of focusing on the n items. Intuitively, the bounded expansion lemma ensures that we do not round up items too much.

Condition C2 (also called *structural theorem*): For some constant $\rho > 0$ and for some $(\widetilde{I}, D) \in \texttt{round}(I)$, $\text{fsopt}(\widetilde{I}) \leq \rho \, \text{opt}(I) + O(1)$.

Intuitively, the structural theorem says that allowing slicing as per `round` and imposing a structure on the packing does not increase the minimum number of bins by too much. The AAR of `rnaPack` increases with ρ, so we want ρ to be small.

4.3.5 `complexPack`

Condition C3: For some constant $\alpha > 0$ and for any $(\widetilde{I}, D) \in \texttt{round}(I)$ and any $\widetilde{X} \subseteq \widetilde{I}$, $\texttt{complexPack}(\widetilde{X})$ packs $\widetilde{X}$ into at most $\alpha \, \text{fsopt}(\widetilde{X}) + O(1)$ bins.

Intuitively, condition C3 says that we can find a packing that is close to the optimal fractional structured packing. The AAR of `rnaPack` increases with α, so we want α to be small.

4.3.6 `unround`

Condition C4: For some constant $\gamma > 0$, if $\texttt{complexPack}(\widetilde{S})$ outputs a packing of $\widetilde{S}$ into m bins, then `unround` modifies that packing to get a packing of $S - D$ into $\gamma m + O(1)$ bins.

Intuitively, condition C4 says that unrounding does not increase the number of bins by too much. The AAR of `rnaPack` increases with γ, so a small γ is desirable. If `round` only increases the dimensions of items, then unrounding is trivial and $\gamma = 1$.

4.3.7 AAR of R&A

Recall that `simplePack` is a $2b(d_v + 1)$-approximation algorithm for (d_g, d_v) BP (see Section 4.2.2), where $b := 3$ when $d_g = 2$, $b := 9$ when $d_g = 3$, and $b := 4^{d_g} + 2^{d_g} - 1$ when

$d_g > 3$.

Lemma 4.6. *Let $\widetilde{S}$ be as computed by* `rnaPack`(I, β, ε). *Then* $\mathrm{fsopt}(\widetilde{S}) \le \mathrm{fsopt}(\widetilde{I})/\beta + 2b\mu\varepsilon\,\mathrm{opt}(I) + O(1/\varepsilon^2)$ *with high probability.*

Lemma 4.6 (proved in Section 4.4) is the key ingredient in the analysis of R&A. Our proof of Lemma 4.6 is inspired by the analysis in [50]. We prove it by analyzing the *fractional structured* configuration LP of $\widetilde{I}$. By the homogeneity property (C1.3), the number of constraints in this LP is a constant. So by rank lemma (Corollary 3.8) and downward closure property (see Lemma 4.9 in Appendix), we can show fsopt to be approximately equal to the optimal solution to the LP. We then harness the randomness of $\widetilde{S}$ and bounded expansion property (C1.4) to use the independent bounded difference inequality [58] to compare the optimal LP objectives of $\widetilde{S}$ and $\widetilde{I}$.

Theorem 4.7. *With high probability, the number of bins used by* `rnaPack`(I, β, ε) *is at most*

$$\left((\ln\beta)\mu + \frac{\gamma\alpha\rho}{\beta} + 2b(d_v + 1 + \gamma\alpha\mu)\varepsilon\right)\mathrm{opt}(I) + O(1/\varepsilon^2).$$

Proof. Let J_{LP} be the set of bins packed in the *randomized rounding of configuration LP* step (see line 4 in Algorithm 1 in Section 4.4), J_D be the set of bins used to pack the discarded items $D \cap S$, J be the set of bins used to pack the rest of the items $S - D$, and $\widetilde{J}$ be the set of bins used by `complexPack` to pack items in $\widetilde{S}$.

Then $|J_{\mathrm{LP}}| \le T = \lceil (\ln\beta)\|\widehat{x}\|_1 \rceil \le (\ln\beta)\mu\,\mathrm{opt}(I) + O(1)$.

Now, we have $|J_D| \le b\lceil 2\,\mathrm{span}(D)\rceil \le 2b\varepsilon\,\mathrm{span}(I) + b \le 2b(d_v+1)\varepsilon\,\mathrm{opt}(I) + b$. The first inequality follows from the property of `simplePack`, the second follows from C1.1 (Small Discard) and the last follows from Lemma 4.1.

$$\begin{aligned}
\text{Finally, } |J| &\le \gamma|\widetilde{J}| + O(1) && \text{(property of } \texttt{unround} \text{ (C4))}\\
&\le \gamma\alpha\,\mathrm{fsopt}(\widetilde{S}) + O(1) && \text{(property of } \texttt{complexPack} \text{ (C3))}\\
&\le \gamma\alpha\left(\mathrm{fsopt}(\widetilde{I})/\beta + 2b\mu\varepsilon\,\mathrm{opt}(I)\right) + O(1/\varepsilon^2) && \text{(by Lemma 4.6)}\\
&\le \gamma\alpha\,(\rho/\beta + 2b\mu\varepsilon)\,\mathrm{opt}(I) + O(1/\varepsilon^2).
\end{aligned}$$

Here, the last inequality follows from the structural theorem (C2), which says that $\exists(\widetilde{I}, D) \in$

$\texttt{round}(I)$ such that $\text{fsopt}(\widetilde{I}) \le \rho\,\text{opt}(I) + O(1)$. Hence, the total number of bins is at most

$$|J_{\text{LP}}| + |J_D| + |J| \le \left((\ln\beta)\mu + \frac{\gamma\alpha\rho}{\beta} + 2b(d_v + 1 + \gamma\alpha\mu)\varepsilon\right)\text{opt}(I) + O(1/\varepsilon^2). \qquad \square$$

The AAR of $\texttt{rnaPack}(I)$ is roughly $\mu\ln\beta + \gamma\alpha\rho/\beta$. This is minimized for $\beta = \gamma\alpha\rho/\mu$ and the minimum value is $\mu\,(1 + \ln(\alpha\gamma\rho/\mu))$. As we require $\beta \ge 1$, we get this AAR only when $\gamma\alpha\rho \ge \mu$. If $\mu \ge \gamma\alpha\rho$, the optimal β is 1 and the AAR is roughly $\gamma\alpha\rho$.

4.3.8 Example: `simplePack`

We will show how to use `simplePack` with the R&A framework. Recall that `simplePack` is a $2b(d_v + 1)$-approximation algorithm for $(d_g,\, d_v)$ BP (see Section 4.2.2), where $b := 3$ when $d_g = 2$, $b := 9$ when $d_g = 3$, and $b := 4^{d_g} + 2^{d_g} - 1$ when $d_g > 3$. Using the R&A framework on `simplePack` will improve its AAR from $2b(d_v + 1)$ to $b(1 + \ln(2(d_v + 1))) + O(\varepsilon)$. To do this, we need to show how to implement `solveConfigLP`, `round`, `complexPack` and `unround`.

1. $\texttt{solveConfigLP}(I)$: Using the knapsack algorithm of Section 4.2.4 and the LP algorithm of [72], we get a $b(1+\varepsilon)$-approximate solution to configLP(I). Therefore, $\mu = b(1+\varepsilon)$.

2. $\texttt{round}(I)$: returns just one pair: $(\widetilde{I}, \{\})$, where $\widetilde{I} := \{\pi(i) : i \in I\}$ and $\pi(i)$ is an item having height (d_g^{th} geometric dimension) equal to $\text{span}(i)$, all other geometric dimensions equal to 1, and all vector dimensions equal to $\text{span}(i)$. There is just one class in $\widetilde{I}$ and all items are allowed to be sliced perpendicular to the height, so the homogeneity properties are satisfied. Also, $c_{\max} = d_v + 1$ by Lemma 4.1.

3. **Structural theorem**: We take structured packing to be the set of all possible packings (i.e., $\mathcal{S}$ is the set of all possible fractional configurations). Then $\text{fsopt}(\widetilde{I}) = \lceil\text{span}(I)\rceil \le (d_v + 1)\,\text{opt}(I)$. Therefore, $\rho = d_v + 1$.

4. $\texttt{complexPack}(\widetilde{S})$: We can treat $\widetilde{S}$ as the classical bin packing instance $\{\text{span}(i) : i \in S\}$ and pack it using Next-Fit. Therefore, by Lemma 3.1, we get $|\texttt{complexPack}(\widetilde{S})| \le \lceil 2\,\text{span}(S)\rceil \le 2\lceil\text{span}(S)\rceil = 2\,\text{fsopt}(\widetilde{S})$. Therefore, $\alpha = 2$.

5. $\texttt{unround}(\widetilde{J})$: For each bin in $\widetilde{J}$, we can pack the corresponding unrounded items into b bins (using Steinberg's algorithm or [27] or $\texttt{fullh}_4$). Therefore, $\gamma = b$.

Therefore, we get an AAR of $\mu(1 + \ln(\gamma\alpha\rho/\mu)) + O(\varepsilon) \approx b(1 + \ln(2(d_v + 1))) + O(\varepsilon)$.

For $d_g = 2$, we can slightly improve the AAR by using the $(2+\varepsilon)$-approximation algorithm of [53] for $(2, d_v)$ KS. This gives us an AAR of $2(1+\ln(3(d_v+1))) + O(\varepsilon)$. This is better than the AAR of `betterSimplePack` for $d_v \geq 3$.

The above example is presented only to illustrate an easy use of the R&A framework. It doesn't exploit the full power of the R&A framework. The algorithm `cbPack`, which we outline in Chapter 5, uses more sophisticated subroutines `round`, `complexPack` and `unround`, and uses a more intricate definition of fractional structured packing to get an even better AAR of $2(1+\ln(\frac{d+4}{2})) + \varepsilon$ (improves to $2(1+\ln(19/12)) + \varepsilon \approx 2.919 + \varepsilon$ for $d = 1$).

4.4 Details of the R&A Framework

Let configLP(I) denote the configuration LP of items I and let configLP$^*(I)$ denote the optimal objective value of configLP(I).

Lemma 4.8. $\forall i \in I$, $\Pr(i \in S) \leq \exp\left(-\frac{T}{\|\widehat{x}\|_1}\right) \leq \frac{1}{\beta}$.

Proof. Let $C_1, C_2, \ldots, C_T$ be the configurations chosen during randomized rounding (line 3 in Algorithm 1). Let $\mathcal{C}_i$ be the configurations that contain the element i.

$$\begin{aligned}
\Pr(i \in S) &= \Pr\left(\bigwedge_{t=1}^{T}(C_t \notin \mathcal{C}_i)\right) = \prod_{t=1}^{T}\Pr(C_t \notin \mathcal{C}_i) && \text{(all } C_t \text{ are independent)} \\
&= \prod_{t=1}^{T}\left(1 - \sum_{C\in\mathcal{C}_i}\Pr(C_t = C)\right) = \left(1 - \sum_{C\in\mathcal{C}_i}\frac{\widehat{x}_C}{\|\widehat{x}\|_1}\right)^T \\
&\leq \left(1 - \frac{1}{\|\widehat{x}\|_1}\right)^T && \text{(constraint in configuration LP for item } i) \\
&\leq \exp\left(-\frac{T}{\|\widehat{x}_1\|}\right) \leq \frac{1}{\beta}. && \square
\end{aligned}$$

Definition 4.4 (Fractional Configuration LP)**.** *Let* $(\widetilde{I}, D) \in$ `round`(I). *Suppose* `round` *partitioned* $\widetilde{I}$ *into classes* $\widetilde{K}_1, \widetilde{K}_2, \ldots \widetilde{K}_q$. *Let* $\mathcal{S}$ *be the set of all structured fractional configurations of* $\widetilde{I}$. *The fractional structured configuration LP of* $\widetilde{S} \subseteq \widetilde{I}$, *denoted as* fsconfigLP$(\widetilde{S})$, *is*

$$\begin{aligned}
\min_{x\in\mathbb{R}^{|\mathcal{S}|}} \quad & \sum_{C\in\mathcal{S}} x_C \\
\textit{where} \quad & \sum_{C\in\mathcal{S}} \text{span}(C \cap \widetilde{K}_j) x_C \geq \text{span}(\widetilde{S} \cap \widetilde{K}_j) \quad \forall j \in [q] \\
& x_C \geq 0 \qquad\qquad\qquad\qquad\qquad\qquad\quad \forall C \in \mathcal{S}
\end{aligned}$$

The integer version of this program is denoted as $\text{fsconfigIP}(\widetilde{S})$. *The optimal objective values of* $\text{fsconfigLP}(\widetilde{S})$ *and* $\text{fsconfigIP}(\widetilde{S})$ *are denoted as* $\text{fsconfigLP}^*(\widetilde{S})$ *and* $\text{fsconfigIP}^*(\widetilde{S})$.

Lemma 4.9. $\text{fsopt}(\widetilde{S}) \leq \text{fsconfigIP}^*(\widetilde{S}) \leq \text{fsopt}(\widetilde{S}) + q$.

Proof. Due to the downward closure property, changing inequality constraints to equality constraints doesn't affect the optimum values of the above LP and IP. Therefore, $\text{fsconfigIP}(\widetilde{S})$ is equivalent to the fractional structured bin packing problem.

A problem with the above definition of $\text{fsconfigLP}(\widetilde{I})$ is that the number of variables can be infinite if certain classes allow slicing. We circumvent this problem by *discretizing* the configurations: Let δ be the smallest dimension of any item, i.e. $\delta := \min\left(\min_{j=1}^{d_g} \ell_j(i), \min_{j=1}^{d_v} v_j(i)\right)$.

In any optimal integral solution to $\text{fsconfigLP}(\widetilde{I})$ that uses m bins, we can slice out some items from each class in each bin so that the span of each class in each bin is a multiple of δ^{d_g}/n. In each class, the total size of sliced out items across all bins is at most δ^{d_g}. Therefore, for each class, slices of that class can fit into a single item of that class. If each such single item is packed in a separate bin, the total number of bins used is at most $m + q$.

Therefore, we only need to consider configurations where either the span of each class is a multiple of δ^{d_g}/n or there is a single item in the configuration. We modify the set $\mathcal{S}$ accordingly. This gives us a finite number of configurations and completes the proof. □

Lemma 4.10. $\text{fsconfigLP}^*(\widetilde{S}) \leq \text{fsconfigIP}^*(\widetilde{S}) \leq \text{fsconfigLP}^*(\widetilde{S}) + q$.

Proof. By rank lemma (Corollary 3.8), the number of non-zero variables in an extreme-point solution to a linear program is at most the number of constraints (other than the variable non-negativity constraints).

Thus, an optimal extreme-point solution to $\text{fsconfigLP}(\widetilde{S})$ has at most q positive-valued variables. Rounding up those variables to the nearest integer will give us an integral solution and increase the objective value by at most q. Hence, $\text{fsconfigIP}^*(\widetilde{S}) \leq \text{fsconfigLP}^*(\widetilde{S}) + q$. □

Recall that `simplePack` is a $2b(d_v + 1)$-approximation algorithm for $(d_g,\ d_v)$ BP (see Section 4.2.2), where $b := 3$ for $d_g = 2$, $b := 9$ for $d_g = 3$, and $b := 4^{d_g} + 2^{d_g} - 1$ for $d_g > 3$.

Lemma 4.11. *For a set* I *of* $(d_g,\ d_v)$*-dimensional items,* $\text{configLP}^*(I) \in \Theta(\text{span}(I)) + O(1)$.

Proof. Let A be the configuration matrix of I. Let x^* be the optimal solution to $\text{configLP}(I)$. In $\text{configLP}(I)$, the constraint for item i gives us $\sum_{C \in \mathcal{C}} A[i, C] x^*_C \geq 1$. Multiplying each constraint

by $\text{span}(i)$ and adding these constraints together, we get

$$\begin{aligned}\text{span}(I) &\leq \sum_{C\in\mathcal{C}}\sum_{i\in I}\text{span}(i)A[i,C]x^*_C = \sum_{C\in\mathcal{C}}\text{span}(C)x^*_C \\ &\leq (d_v+1)\sum_{C\in\mathcal{C}}x^*_C = (d_v+1)\,\text{configLP}^*(I).\end{aligned}$$

Therefore, $\text{configLP}^*(I) \geq \text{span}(I)/(d_v+1)$. We also have

$$\text{configLP}^*(I) \leq \text{opt}(I) \leq |\,\texttt{simplePack}(I)| \leq 2b\,\text{span}(I) + b.$$

Therefore, $\text{configLP}^*(I) \in \Theta(\text{span}(I)) + O(1)$. □

Lemma 4.12 (Independent Bounded Difference Inequality [58])**.** *Let $X := [X_1, X_2, \ldots, X_n]$ be random variables with $X_j \in A_j$. Let $\phi : \prod_{i=1}^n A_j \mapsto \mathbb{R}$ be a function such that $|\phi(x) - \phi(y)| \leq c_j$ whenever vectors x and y differ only in the j^{th} coordinate. Then for any $t \geq 0$,*

$$\Pr[\phi(X) - \text{E}(\phi(X)) \geq t] \leq \exp\left(-\frac{2t^2}{\sum_{j=1}^n c_j^2}\right).$$

Lemma 4.13. *Let $\widetilde{S}$ be as computed by $\texttt{rnaPack}(I,\beta,\varepsilon)$. Let $\varepsilon \in (0,1)$ be a constant. When $\text{span}(I)$ is large compared to $1/\varepsilon^2$, we get that with high probability*

$$\text{fsconfigLP}^*(\widetilde{S}) \leq \frac{\text{fsconfigLP}^*(\widetilde{I})}{\beta} + 2b\mu\varepsilon\,\text{opt}(I) + O(1).$$

Proof. Let $y \in \mathcal{C}^T$ be the configurations chosen during randomized rounding. When viewed as a vector of length T, all coordinates of y are independent. Define $\text{uncovered}(y) := I - \bigcup_{t=1}^T y_t$.

Let $\widetilde{K}_1, \widetilde{K}_2, \ldots, \widetilde{K}_q$ be the classes of $\widetilde{I}$. Let π be the bijection from $I - D$ to $\widetilde{I}$. For a set $X \subseteq I$, define $\widetilde{I}[X] := \{\pi(i) : i \in X - D\}$. For $j \in [q]$, define $\phi_j \in \mathcal{C}^T \mapsto \mathbb{R}_{\geq 0}$ as

$$\phi_j(y) := \text{span}\left(\widetilde{K}_j \cap \widetilde{I}[\text{uncovered}(y)]\right).$$

For any set $X \subseteq I$, define $g_j(X) := \text{span}(\widetilde{K}_j \cap \widetilde{I}[X])$. Then $\phi_j(y) = g_j(\text{uncovered}(y))$ and g_j is a non-negative additive function.

Let $y^{(1)}, y^{(2)} \in \mathcal{C}^T$ such that $y^{(1)}$ and $y^{(2)}$ differ only in coordinate t. Let $y_t^{(1)} = C_1$ and $y_t^{(2)} = C_2$. Let $S_1 = \text{uncovered}(y^{(1)})$ and $S_2 = \text{uncovered}(y^{(2)})$.

It is easy to see (using Venn diagrams) that $S_1 - S_2 \subseteq C_2 - C_1$ and $S_2 - S_1 \subseteq C_1 - C_2$.

$$\begin{aligned}
\left|\phi_j(y^{(1)}) - \phi_j(y^{(2)})\right| &= |g_j(S_1) - g_j(S_2)| \\
&= |g_j(S_1 - S_2) - g_j(S_2 - S_1)| && \text{(additivity of } g_j\text{)} \\
&\le \max(g_j(S_1 - S_2), g_j(S_2 - S_1)) \\
&\le \max(g_j(C_2), g_j(C_1)) \\
&\le \max_{C \in \mathcal{C}} \operatorname{span}(\widetilde{K}_j \cap \widetilde{I}[C]) \le c_{\max}. && \text{(by bounded expansion lemma)}
\end{aligned}$$

$$\begin{aligned}
\mathrm{E}(\phi_j(y)) &= \mathrm{E}(g_j(S)) \\
&= \sum_{i \in \widetilde{I}} g_j(\{i\}) \Pr(i \in S) && \text{(linearity of expectation and additivity of } g_j\text{)} \\
&\le \sum_{i \in \widetilde{I}} g_j(\{i\})(1/\beta) && \text{(by Lemma 4.8)} \\
&= \frac{g_j(\widetilde{I})}{\beta} = \frac{\operatorname{span}(\widetilde{K}_j)}{\beta}.
\end{aligned}$$

$\forall j \in [q]$, define Q_j as the smallest prefix of $\widetilde{S} \cap \widetilde{K}_j$ such that either $Q_j = \widetilde{S} \cap \widetilde{K}_j$ or $\operatorname{span}(Q_j) \ge \varepsilon \left\|\widehat{x}\right\|_1 / q$. Define $Q := \bigcup_{j=1}^{q} Q_j$. Therefore,

$$\operatorname{span}(Q) \le \varepsilon \left\|\widehat{x}\right\|_1 + q \le \varepsilon\mu \operatorname{opt}(I) + O(1).$$

$$\begin{aligned}
\operatorname{fsconfigLP}^*(\widetilde{S}) &\le \operatorname{fsconfigLP}^*(\widetilde{S} - Q) + \operatorname{fsconfigLP}^*(Q) \\
&\le \operatorname{fsconfigLP}^*(\widetilde{S} - Q) + b(2\operatorname{span}(Q) + 1) && \text{(by Section 4.2.2)} \\
&\le \operatorname{fsconfigLP}^*(\widetilde{S} - Q) + 2b\mu\varepsilon \operatorname{opt}(I) + O(1).
\end{aligned}$$

Now we will try to prove that with high probability, $\operatorname{fsconfigLP}^*(\widetilde{S} - Q) \le \operatorname{fsconfigLP}^*(\widetilde{I})/\beta$.

If $Q_j = \widetilde{S} \cap \widetilde{K}_j$, then $\operatorname{span}(\widetilde{K}_j \cap (\widetilde{S} - Q)) = 0$. Otherwise,

$$\begin{aligned}
&\Pr\left[\operatorname{span}(\widetilde{K}_j \cap (\widetilde{S} - Q)) \ge \frac{\operatorname{span}(\widetilde{K}_j)}{\beta}\right] = \Pr\left[\operatorname{span}(\widetilde{K}_j \cap \widetilde{S}) - \frac{\operatorname{span}(\widetilde{K}_j)}{\beta} \ge \operatorname{span}(Q_j)\right] \\
&\le \Pr\left[\phi_j(y) - \mathrm{E}(\phi_j(y)) \ge \frac{\varepsilon}{q}\left\|\widehat{x}\right\|_1\right] \le \exp\left(-\frac{2}{T c_{\max}^2}\left(\frac{\varepsilon}{q}\left\|\widehat{x}\right\|_1\right)^2\right) && \text{(Lemma 4.12)}
\end{aligned}$$

$$\leq \exp\left(-\frac{2\varepsilon^2}{\ln(\beta)c_{\max}^2 q^2}\left\|\widehat{x}\right\|_1\right).$$

Therefore, by union bound, we get

$$\Pr\left[\bigvee_{j=1}^{q}\left(\operatorname{span}(\widetilde{K}_j \cap (\widetilde{S}-Q)) \geq \frac{\operatorname{span}(\widetilde{K}_j)}{\beta}\right)\right] \leq q\exp\left(-\frac{2\varepsilon^2}{\ln(\beta)c_{\max}^2 q^2}\left\|\widehat{x}\right\|_1\right).$$

Since $\text{configLP}^*(I) \leq \left\|\widehat{x}\right\|_1 \leq \mu\,\text{configLP}^*(I) + O(1)$, and $\text{configLP}^*(I) \in \Theta(\operatorname{span}(I)) + O(1)$ (by Lemma 4.11), we get $\left\|\widehat{x}\right\|_1 \in \Theta(\operatorname{span}(I)) + O(1)$. When $\operatorname{span}(I)$ is very large compared to $1/\varepsilon^2$, we get that with high probability, $\forall j \in [q]$,

$$\operatorname{span}(\widetilde{K}_j \cap (\widetilde{S}-Q)) \leq \frac{\operatorname{span}(\widetilde{K}_j)}{\beta}.$$

Let x^* be the optimal solution to $\text{fsconfigLP}(\widetilde{I})$. Then with high probability, x^*/β is a feasible solution to $\text{fsconfigLP}(\widetilde{S}-Q)$. Therefore,

$$\begin{aligned}\text{fsconfigLP}^*(\widetilde{S}) &\leq \text{fsconfigLP}^*(\widetilde{S}-Q) + 2b\mu\varepsilon\,\operatorname{opt}(I) + O(1)\\ &\leq \text{fsconfigLP}^*(\widetilde{I})/\beta + 2b\mu\varepsilon\,\operatorname{opt}(I) + O(1).\end{aligned}$$

□

Lemma 4.6. *Let $\widetilde{S}$ be as computed by* `rnaPack`(I,β,ε). *Then* $\text{fsopt}(\widetilde{S}) \leq \text{fsopt}(\widetilde{I})/\beta + 2b\mu\varepsilon\,\operatorname{opt}(I) + O(1/\varepsilon^2)$ *with high probability.*

Proof. When $\operatorname{span}(I)$ is very large compared to $1/\varepsilon^2$, we get

$$\begin{aligned}\text{fsopt}(\widetilde{S}) &\leq \text{fsconfigIP}^*(\widetilde{S}) + O(1) && \text{(by Lemma 4.9)}\\ &\leq \text{fsconfigLP}^*(\widetilde{S}) + O(1) && \text{(by Lemma 4.10)}\\ &\leq \text{fsconfigLP}^*(\widetilde{I})/\beta + 2b\mu\varepsilon\,\operatorname{opt}(I) + O(1) && \text{(by Lemma 4.13)}\\ &\leq \text{fsopt}(\widetilde{I})/\beta + 2b\mu\varepsilon\,\operatorname{opt}(I) + O(1). && \text{(by Lemma 4.9)}\end{aligned}$$

Otherwise, if $\operatorname{span}(I) \in O(1/\varepsilon^2)$, we get

$$\begin{aligned}\text{fsopt}(\widetilde{S}) &\leq \rho\,\operatorname{opt}(I) + O(1) && \text{(by structural theorem)}\\ &\leq \rho|\,\texttt{simplePack}(I)| + O(1)\\ &\leq \Theta(\operatorname{span}(I)) + O(1) && \text{(by Section 4.2.2)}\\ &\leq O(1/\varepsilon^2).\end{aligned}$$

□

4.4.1 Error in previous R&A framework

Here we describe a minor error in the R&A framework of [50], and how it can be fixed.

We define $(\widetilde{I}, D)$ as an output of $\texttt{round}(I)$ and for the residual instance S, we define $\widetilde{S}$ as the corresponding rounded items of $S - D$. Our proof of Lemma 4.6 relies on the fact that for any subset of rounded items, the span reduces by a factor of at least β if we restrict our attention to the residual instance. Formally, this means that for any $\widetilde{X} \subseteq \widetilde{I}$, we have

$$\mathrm{E}(\mathrm{span}(\widetilde{X} \cap \widetilde{S})) = \sum_{i \in \widetilde{X}} \mathrm{span}(i) \Pr(i \in \widetilde{S}) \leq \mathrm{span}(\widetilde{X})/\beta.$$

The equality follows from linearity of expectation and the fact that $\mathrm{span}(i)$ is deterministic, i.e., it doesn't depend on the randomness used in the randomized rounding of the configuration LP. This is because `round` is not given any information about what S is. The inequality follows from Lemma 4.8, which says that $\Pr(i \in S) \leq 1/\beta$.

The R&A framework of [50] used similar techniques in their analysis. In their algorithm, however, they round items differently. Specifically, they define a subroutine `round` and define $\widetilde{I} := \texttt{round}(I)$ and $\widetilde{S} := \texttt{round}(S)$. They, too, claim that for any subset of rounded items, the span reduces by a factor of at least β if we restrict our attention to the residual instance. While their claim is correct for input-agnostic rounding (where items are rounded up to some constant size collection values chosen independent of the problem instance), the claim is unsubstantiated for input-sensitive rounding (where the values are chosen based on the specific problem instance). So the claim is unsubstantiated if `round` is not deterministic, as then an item can be rounded differently depending on different residual instances.

In fact, they use their R&A framework with the algorithm of Jansen and Prädel [41], which uses *linear grouping* (along with some other techniques) for rounding. Linear grouping rounds items in an *input-sensitive* way, i.e., the rounding of each item depends on the sizes of items in S, which is a random subset.

4.5 The $\texttt{fullh}_4$ Algorithm

In this section, we describe the $\texttt{fullh}_4$ algorithm and prove an important result about it. Although a more detailed analysis is given in Section 6.2, the analysis here is simpler because we only focus on the part relevant to (d_g, d_v) BP.

Define $f_4 : [0,1] \mapsto [0,1]$ and $\text{type} : [0,1] \mapsto [4]$ as

$$f_4(x) = \begin{cases} \frac{1}{q} & x \in \left(\frac{1}{q+1}, \frac{1}{q}\right] \text{ for } q \in [3] \\ 2x & x \le \frac{1}{4} \end{cases} \qquad \text{type}(x) = \begin{cases} q & x \in \left(\frac{1}{q+1}, \frac{1}{q}\right] \text{ for } q \in [3] \\ 4 & x \le \frac{1}{4} \end{cases}$$

Let i be a d_g-dimensional cuboid. Define $f_4(i)$ as the cuboid of length $f_4(\ell_j(i))$ in the j^{th} dimension. Note that $x \le f_4(x) \le 2x$. Hence, $\text{vol}(i) \le \text{vol}(f_4(i)) \le 2^{d_g} \text{vol}(i)$. For a set I of cuboids, $f_4(I) := \{f_4(i) : i \in I\}$. Define $\text{type}(i) \in [4]^{d_g}$ as a vector where $\text{type}(i)_j := \text{type}(\ell_j(i))$.

Caprara [18] (implicitly) defines a recursive algorithm $\texttt{HDH-unit-pack}_k^{[t]}(I)$ (see Section 6.1.3 for more details), that takes as input a sequence I of dD cuboidal items, such that all items have the same type t and $\text{vol}(f_4(I - \{\text{last}(I)\})) < 1$, and returns a packing of I into a dD bin. Here $\text{last}(I)$ is the last item in sequence I. $\texttt{HDH-unit-pack}_k^{[t]}(I)$ runs in $O(|I| \log |I|)$ time.

We now describe the $\texttt{fullh}_4$ algorithm. First, partition the items I by type. The number of partitions is at most $Q = 4^{d_g}$. Let $I^{[q]}$ be the partition containing items of type q. Order the items in $I^{[q]}$ arbitrarily. Then repeatedly pick the smallest prefix J of $I^{[q]}$ such that either $J = I^{[q]}$ or $\text{vol}(f_4(J)) \ge 1$, and pack J into a bin using $\texttt{HDH-unit-pack}_k^{[q]}(J)$.

Lemma 4.14. *For a non-empty d_gD GBP instance I, $|\texttt{fullh}_4(I)| < 4^{d_g} + 2^{d_g} \text{vol}(I)$.*

Proof. Suppose $\texttt{fullh}_4(I^{[q]})$ produces $m^{[q]}$ bins. Let $B_j^{[q]}$ be the j^{th} of these bins. Given the way we choose prefixes, $\text{vol}(f_4(B_j^{[q]})) \ge 1$ for $j \in [m^{[q]} - 1]$, i.e., at most 1 bin is partially-filled. Hence,

$$\text{vol}(f_4(I^{[q]})) = \sum_{j=1}^{m^{[q]}} \text{vol}(f_4(B_j^{[q]})) > m^{[q]} - 1.$$

So, the total number of bins used is

$$\sum_{q=1}^{Q} m^{[q]} < \sum_{q=1}^{Q} (1 + \text{vol}(f_4(I^{[q]}))) = Q + \text{vol}(f_4(I)) \le 4^{d_g} + 2^{d_g} \text{vol}(I). \qquad \square$$

Therefore, $\text{vol}(I) \le 1 \implies |\texttt{fullh}_4(I)| \le 4^{d_g} + 2^{d_g} - 1$.

Chapter 5

Improved Algorithm for Generalized Multidimensional Bin Packing

Here we will see an algorithm for $(2, d)$ bin packing, called `cbPack` (named after 'compartment-based packing'). See Section 1.2.1 for an introduction to the $(2, d)$ BP problem and a comparison of the approximation guarantees of algorithms for this problem, including the `cbPack` algorithm.

The `cbPack` algorithm is inspired by Jansen and Prädel's [41, 66] $(1.5 + \varepsilon)$-asymptotic-approximation algorithm for 2D GBP. Like their algorithm, the design of `cbPack` follows the two-step outline described in Section 4.3.3. In the first step, called *structural step*, we show that for any input I, we can round I to get a new instance $\widetilde{I}$ such that $\text{fsopt}(\widetilde{I}) \leq \rho \, \text{opt}(I) + O(1)$ for some constant ρ, where $\text{fsopt}(\widetilde{I})$ is the optimal fractional *compartmental* packing of $\widetilde{I}$ (we will define compartmental later). In the second step, called the *algorithmic step*, we give an algorithm for finding a packing of I that uses roughly $\text{fsopt}(\widetilde{I})$ bins. We do this by first rounding I to $\widetilde{I}$, then finding the optimal fractional compartmental packing of $\widetilde{I}$ using brute-force and linear programming, and then converting this packing to a non-fractional packing of I with only a tiny increase in the number of bins.

Our notion of structured packing, which we call *compartmental packing*, imposes roughly the following additional constraints over the *container-based packing* of [66]:

- An item i is called *dense* iff $\text{v}_{\max}(i)/a(i)$ is above a certain threshold. If a bin contains dense items, then we reserve a sufficiently-large rectangular region exclusively for dense items, and dense items can only be packed into this region.
- For a constant ε, for every $j \in [d]$, if a set B of items is packed into a bin, then $v_j(B) \leq 1 - \varepsilon$.

We give a more precise definition of *compartmental* in Section 5.5.

`cbPack` can be easily broken into subroutines `round`, `complexPack` and `unround`, and we show in Section 5.7 that it satisfies all the conditions of the R&A framework. To (approximately) solve the configuration LP, we use the linear programming algorithm from [72] and the $(2+\varepsilon)$-approximation algorithm for $(2, d)$ KS from [53].

5.1 Overview of the Algorithm and its Analysis

`cbPack` will be parametrized by a parameter ε, where $\varepsilon^{-1} \in 2\mathbb{Z}$ and $\varepsilon \leq 1/8$. It takes a set I of $(2, d)$-dimensional items as input. Recall that each item $i \in I$ has width $w(i)$, height $h(i)$ and d weights $v_1(i), v_2(i), \ldots, v_d(i)$.

In Section 5.2, we remove a small subset of problematic items and classify the rest of the items based on their geometric and vector dimensions. More precisely, we find two constants $\varepsilon_2 \ll \varepsilon_1$ (which are functions of ε) and classify items as follows:

- Big item: $w(i) > \varepsilon_1$ and $h(i) > \varepsilon_1$.
- Wide item: $w(i) > \varepsilon_1$ and $h(i) \leq \varepsilon_2$.
- Tall item: $w(i) \leq \varepsilon_2$ and $h(i) > \varepsilon_1$.
- Small item: $w(i) \leq \varepsilon_2$ and $h(i) \leq \varepsilon_2$.

We prove that the remaining items can be packed into a small number of bins (because of the way we chose ε_2 and ε_1). We call an item i *dense* if $\mathrm{v}_{\max}(i)/a(i) > 1/\varepsilon_1^2$.

In Section 5.3 we define *semi-structured* packing. Given an optimal packing of the items into m bins, we show how to round the items and obtain a fractional semi-structured packing of those items into roughly $\rho m + O(1)$ bins, for some constant ρ. This section covers a large part of the structural step of `cbPack`'s analysis.

The rounding of items in Section 5.3 is done with the knowledge of the optimal packing of the items. Section 5.4 explains how to design a rounding algorithm that works without knowing the optimal packing.

In Section 5.5, we define compartmental packing and show how to convert a semi-structured packing to a compartmental packing with only a tiny increase in the number of bins.

In Section 5.6, we show how to compute an optimal fractional compartmental packing of rounded items and how to convert that packing to a non-fractional packing with only a tiny increase in the number of bins. We then show how to *unround* the packing without increasing the number of bins.

In Section 5.7, we show how to apply the R&A framework to `cbPack`.

5.1.1 Overview of Key Ideas used in the Structural Theorem

Our definition of semi-structured is heavily influenced by the challenges posed by the presence of vector dimensions.

Given an optimal packing of the items into m bins, we show in our structural step how to round the items and obtain a fractional compartmental packing of those items into roughly $\rho m + O(1)$ bins, for some constant ρ. Our high-level strategy for doing this, which is similar to that of Jansen and Prädel [41, 66] for 2D GBP, is as follows:

1. In the first step, we round up one geometric dimension of each item and pack the items into roughly $\rho m + O(1)$ bins. We call these bins *quarter-structured* (see Sections 5.3.1 and 5.3.2).

2. In the second step, we round the remaining dimensions of items and partition them into classes such that they satisfy the *homogeneity properties* (see Section 4.3.4). We allow slicing the items and repack them into almost the same number of bins. We call the resulting bin packing *semi-structured* (see Sections 5.3.3 and 5.3.4).

3. In the third step, we transform the packing into a *compartmental* packing (see Section 5.5). Compartmental packings have nice properties which make it easy to find optimal fractional compartmental packings.

In steps 1, 2 and 3 above, [66] uses the NFDH algorithm (see Lemma 3.5) to pack items of $O(\varepsilon m)$ area into $O(\varepsilon m)$ bins. This doesn't work when vector dimensions are present, since an item of low area can have large weights. In step 2, [66] uses *linear grouping*, i.e., each item is moved in place of a geometrically larger item so that it can be rounded up. Vector dimensions make such cross-bin movement difficult, since that can violate bins' weight capacities.

Our first crucial observation is that most difficulties associated with vector dimensions disappear if items' *density* is upper-bounded by a constant. Here density of item i is defined as $v_{\max}(i)/a(i)$. Specifically, if items of bounded density (we call them non-dense items) have small area, then we can use `simplePack` to pack them into a small number of bins. To make linear grouping work, we can partition items of bounded density into a constant number of classes such that items in the same class have almost the same value of $\mathrm{v}_{\max}(i)/a(i)$. Therefore, our strategy is to segregate items as *dense* and *non-dense*. Furthermore, dense items in a bin must have low total area, due to their high density. If we reserve enough space for them in the bin, we can always pack them in their reserved region using NFDH (see Lemma 3.3). Such a guarantee means that we can essentially ignore their geometric dimensions and simply treat them as vectors.

In step 2, we want to round up vector dimensions with only a marginal increase in the number of bins. To do this, we require each quarter-structured bin to be ε-slacked. ε-slackness roughly means that for a set J of items in a bin, $\forall j \in [d], v_j(J) \leq 1 - \varepsilon$ (see Section 5.3.2 for a formal description). ε-slackness also helps us in designing the packing algorithm, because we can then use techniques from resource-augmented vector bin packing. Also, during the rounding step, we round down the weight of some dense items, and ε-slackness allows us to *unround* with no increase in the number of bins.

The observations above guide our definition of *quarter-structured*. Roughly, a packing is quarter-structured iff all of the following hold:

- Wide items have their width and x-coordinate rounded to a multiple of $\varepsilon_1^2/4$.
- Each bin is ε-slacked.
- If a bin contains dense items, a rectangular region of width $\varepsilon_1/2$ and height 1 is reserved for them, and dense items can only be packed in this region.

In step 1, Jansen and Prädel [41, 66] use a standard cutting-strip argument: They create a strip of width ε_1 next to an edge of the bin (see Fig. 5.1 for an example). Items lying completely inside the strip (called blue items), have small area and are packed separately using NFDH. Items intersecting the boundary of the strip (called red items), are removed. This creates an empty space of width ε_1 in the bin. Using this empty space, items lying outside the strip (called green items), can then have their width and x-coordinate rounded to a multiple of $\varepsilon_1^2/2$. Their key idea is how to pair up most bins so that red items from two bins can be rounded and packed together into a new bin. This is roughly why they get an AAR of $1.5 + \varepsilon$.

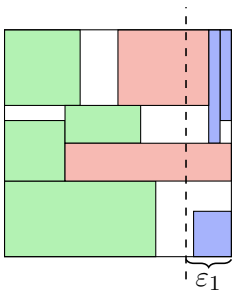


Figure 5.1: Example of classifying items as blue, red and green based on an ε_1-strip.

We use the cutting-strip argument too, but with some differences. We cannot freely mix red items from different bins if they have large weight, and we cannot simply pack blue items into a small number of bins. We also need bins to be slacked. So, we get a larger AAR of $d + 4 + \varepsilon$.

For $d = 1$, however, we allow mixing items using more sophisticated techniques, which improves the AAR to $19/6 + \varepsilon$. Also, we round green items to a multiple of $\varepsilon_1^2/4$ instead of $\varepsilon_1^2/2$, which leaves an empty strip of width $\varepsilon_1/2$ in the bin even after rounding, and we reserve this space for dense items. This gives us a quarter-structured packing.

We have finished giving an overview of the algorithm. We now turn to the details of the algorithm and its analysis.

5.2 Classifying Items

Definition 5.1. *For constants $\varepsilon_2 < \varepsilon_1$, a bin packing instance I is called $(\varepsilon_2, \varepsilon_1)$-non-medium iff $\forall i \in I$, $(w(i) \notin (\varepsilon_2, \varepsilon_1]) \wedge (h(i) \notin (\varepsilon_2, \varepsilon_1]) \wedge (\forall j \in [d], v_j(i) \notin (\varepsilon_2, \varepsilon_1])$.*

An $(\varepsilon_2, \varepsilon_1)$-non-medium instance has useful properties. Therefore, we want to remove some items from the input instance I so that it becomes $(\varepsilon_2, \varepsilon_1)$-non-medium and the removed items can be packed into a small number of bins.

Definition 5.2. *Let $\delta_0, \varepsilon \in (0, 1]$ be constants and let $f : (0, 1] \mapsto (0, 1]$ be a function such that $\forall x \in (0, 1], f(x) < x$. Let $T := \lceil (d + 2)/\varepsilon \rceil$. For $t \in [T]$, define $\delta_t := f(\delta_{t-1})$ and define*

$$J_t := \left\{ i \in I : w(i) \in (\delta_t, \delta_{t-1}] \vee h(i) \in (\delta_t, \delta_{t-1}] \vee \left(\bigvee_{j=1}^{d} v_j(i) \in (\delta_t, \delta_{t-1}] \right) \right\}.$$

Define `removeMedium`$(I, \varepsilon, f, \delta_0)$ *as the tuple $(J_r, \delta_r, \delta_{r-1})$, where $r := \operatorname{argmin}_{t=1}^{T} \operatorname{span}(J_t)$.*

Lemma 5.1. *Let $(I_{\text{med}}, \varepsilon_2, \varepsilon_1) :=$* `removeMedium`$(I, \varepsilon, f, \delta_0)$. *Then $\operatorname{span}(I_{\text{med}}) \le \varepsilon \operatorname{span}(I)$.*

Proof. Each item belongs to at most $d + 2$ sets J_t. Therefore,

$$\operatorname{span}(I_{\text{med}}) = \min_{t=1}^{T} \operatorname{span}(J_t) \le \frac{1}{T} \sum_{t=1}^{T} \operatorname{span}(J_t) \le \frac{d+2}{T} \operatorname{span}(I) \le \varepsilon \operatorname{span}(I). \qquad \square$$

By Definition 5.2, $I - I_{\text{med}}$ is $(\varepsilon_2, \varepsilon_1)$-non-medium. By Lemma 5.1 and Theorem 4.5, $\operatorname{span}(I_{\text{med}})$ can be packed into at most $6(d + 1)\varepsilon \operatorname{opt}(I) + 3$ bins using the `simplePack` algorithm.

We will choose f to be independent of I, so ε_1 and ε_2 are constants. Also note that $\varepsilon_2 := f(\varepsilon_1)$ and $\varepsilon_1 \le \delta_0$. We choose $\delta_0 := \min(1/(4d + 1), 2\varepsilon/3)$, so $\delta_0^{-1} \in \mathbb{Z}$. We will choose f later. For now, we will impose these conditions: $f(x) \le \varepsilon x^2/2$, and $(x^{-1} \in \mathbb{Z} \implies f(x)^{-1} \in \mathbb{Z})$. The second condition implies that $\varepsilon_1^{-1}, \varepsilon_2^{-1} \in \mathbb{Z}$.

Definition 5.3. *We can classify a non-medium item i by its geometric dimensions as follows:*

- *Big item:* $w(i) > \varepsilon_1$ *and* $h(i) > \varepsilon_1$.
- *Wide item:* $w(i) > \varepsilon_1$ *and* $h(i) \leq \varepsilon_2$.
- *Tall item:* $w(i) \leq \varepsilon_2$ *and* $h(i) > \varepsilon_1$.
- *Small item:* $w(i) \leq \varepsilon_2$ *and* $h(i) \leq \varepsilon_2$.

When rotating items is allowed, assume without loss of generality that there are no tall items in I.

Definition 5.4 (Dense items)**.** *Item i is dense iff either $a(i) = 0$ or $v_{\max}(i)/a(i) > 1/\varepsilon_1^2$.*

Note that big items cannot be dense.

Definition 5.5 (Heavy and light items)**.** *A dense item i is said to be heavy in vector dimension j iff $v_j(i) \geq \varepsilon_1$. Otherwise i is said to be light in dimension j. If i is heavy in some dimension, then i is said to be heavy, otherwise i is light.*

5.3 Getting a Semi-Structured Packing

Given a packing of items $I - I_{\text{med}}$ into m bins, in this section, we will see how to round the items and obtain a fractional semi-structured packing of the rounded items into roughly $\rho m + O(1)$ bins, for some constant ρ. We will also define semi-structured packing in this section.

5.3.1 Rounding One Side

In this subsection, we will show how a packing of I into bins can be modified to get a more structured packing where one of the geometric dimensions is rounded up.

Definition 5.6. *In a bin, assume a coordinate system where $(0, 0)$ is at the bottom left and $(1, 1)$ is on the top right. We define the following regions, called* strips*:*

- $S^{(T)} := [0, 1] \times [1 - \varepsilon_1, 1]$ *and* $S^{(T')} := [0, 1] \times [1 - \varepsilon_1/2, 1]$
- $S^{(B)} := [0, 1] \times [0, \varepsilon_1]$
- $S^{(L)} := [0, \varepsilon_1] \times [0, 1]$
- $S^{(R)} := [1 - \varepsilon_1, 1] \times [0, 1]$ *and* $S^{(R')} := [1 - \varepsilon_1/2, 1] \times [0, 1]$

We say that an item intersects a strip iff a non-zero volume of that item lies inside the strip.

Property 5.7. *A bin is said to satisfy Property 5.7 iff both of these conditions hold:*

(a) *The x-coordinate and width of all non-dense wide and big items is a multiple of $\varepsilon_1^2/4$.*

(b) *If the bin contains dense items, then dense items are packed inside $S^{(R')}$ and no non-dense item intersects $S^{(R')}$.*

Property 5.8. *A bin is said to satisfy Property 5.8 iff both of these conditions hold:*

(a) *The y-coordinate and height of all non-dense tall and big items is a multiple of $\varepsilon_1^2/4$.*

(b) *If the bin contains dense items, then dense items are packed inside $S^{(T')}$ and no non-dense item intersects $S^{(T')}$.*

Equivalently, we can say that a bin satisfies Property 5.8 iff its mirror image about the line $y = x$ satisfies Property 5.7.

The main result of this subsection is the following:

Lemma 5.2. *Given a packing of items into a bin, we can round up the width of some wide and big non-dense items to a multiple of $\varepsilon_1^2/4$ or round up the height of some tall and big non-dense items to a multiple of $\varepsilon_1^2/4$ and get a packing into 2 bins and 2 boxes where:*

- *Each bin satisfies either Property 5.7 or Property 5.8.*
- *v_1 of each box is at most $1/2$.*
- *One of the boxes has only dense items. Its larger dimension is 1 and its smaller dimension is $\delta_d := 2d\varepsilon_1^2 + \varepsilon_2$.*
- *One of the boxes has only non-dense items. Its larger dimension is 1 and its smaller dimension is $\varepsilon_1 + \varepsilon_2$.*
- *One of the boxes is horizontal, i.e., has width 1 and only contains wide and small items. The other box is vertical, i.e., has height 1 and only contains tall and small items.*

Before proving Lemma 5.2, we first prove a few ancillary results.

For $X \in \{T, B\}$, the items lying completely inside $S^{(X)}$ are either small or wide. Let $C^{(X)}$ be the set of small and wide items that intersect $S^{(X)}$. For $X \in \{L, R\}$, the items lying completely inside $S^{(X)}$ are either small or tall. Let $C^{(X)}$ be the set of small and tall items that intersect $S^{(X)}$. Since $2\varepsilon_1 + \varepsilon_2 \leq 1$, we get $C^{(T)} \cap C^{(B)} = C^{(L)} \cap C^{(R)} = \{\}$.

Without loss of generality, assume that $v_1(C^{(T)}) \leq 1/2$ because $v_1(C^{(B)} \cup C^{(T)}) \leq 1$ and if $v_1(C^{(T)}) > v_1(C^{(B)})$, then we can mirror-invert the bin along a horizontal axis. Similarly assume that $v_1(C^{(R)}) \leq 1/2$.

Observation 5.3. *If a bin only contains tall and small items, it trivially satisfies Property 5.7(a). If a bin only contains wide and small items, it trivially satisfies Property 5.8(a).*

Lemma 5.4. *Suppose we're given a packing of items into a bin such that no item intersects $S^{(R)}$. Then we can increase the widths of all wide and big items to a multiple of $\varepsilon_1^2/4$ and repack the items so that they satisfy Property 5.7(a) and no item intersects $S^{(R')}$.*

Proof. Let $y_b(i)$ and $y_t(i)$ be the y-coordinates of the bottom and top edge respectively of item i. If an item j intersects the strip $[0,1] \times [y_b(i), y_t(i)]$ and lies to the right of i (i.e., the left edge of j is to the right of the right edge of i), we say that $i \prec_{\text{imm}} j$ (see Fig. 5.2). Let $\preceq$ denote the reflexive and transitive closure of the relation $\prec_{\text{imm}}$. It is easy to see that $\preceq$ is a partial ordering of I. Define $i \prec j$ as $i \preceq j \wedge i \neq j$.

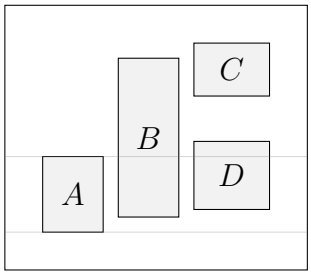


Figure 5.2: Items A, B, C and D in a bin. Here $A \prec_{\text{imm}} D$ but $A \not\prec_{\text{imm}} C$. Also, $A \prec_{\text{imm}} B \prec_{\text{imm}} C$, so $A \preceq C$.

Define $p_w(i)$ to be 1 if it is wide or big and to be 0 if it is neither wide nor big. Also, define $n_w(i) := p_w(i) + \max_{j \prec i} n_w(j)$ (if there is no $j \prec i$, define $\max_{j \prec i} n_w(i) := 0$). Intuitively, $n_w(i)$ denotes the length of the largest chain of wide items preceding i. The x-coordinate of the right edge of item i is more than $\varepsilon_1 n_w(i)$. Therefore, $n_w(i) < 1/\varepsilon_1 - 1$.

Transformation 5.9. *Move each item i to the right by $(n_w(i) - p_w(i))\varepsilon_1^2/2$. Additionally, if i is wide or big, move it further to the right so that the x-coordinate of its left edge is a multiple of $\varepsilon_1^2/4$, and increase its width so that it is a multiple of $\varepsilon_1^2/4$.*

On applying Transformation 5.9 to item i, the x-coordinate of its right edge increases by less than $n_w(i)\varepsilon_1^2/2$. Since $n_w(i) < 1/\varepsilon_1 - 1$, the increase is less than $\varepsilon_1/2$. Therefore, i will not intersect $S^{(R')}$ after this transformation. Also, after applying this transformation to all items, the bin satisfies Property 5.7(a).

We will now prove that after applying Transformation 5.9 to all items, no items overlap. If i and j are not relatively ordered by $\preceq$, they cannot overlap because we only moved items rightwards. Now assume without loss of generality that $i \prec j$. The x-coordinate of the right

edge of i increases by less than $n_w(i)\varepsilon_1^2/2$. The x-coordinate of the left edge of j increases by at least $(n_w(j) - p_w(j))\varepsilon_1^2/2$. Since $n_w(i) \leq \max_{i' \prec j} n_w(i') = n_w(j) - p_w(j)$, i and j don't overlap. □

Lemma 5.5. *Let R be a set of wide and small items that are dense and have total weight at most 1. They can be packed in polynomial time into a box of width 1 and height $\delta_d := 2d\varepsilon_1^2 + \varepsilon_2$.*

Proof. $a(R) \leq \varepsilon_1^2 v_{\max}(R) \leq d\varepsilon_1^2$.
So by Lemma 3.3, height used by NFDH is less than $2a(R) + \varepsilon_2 \leq 2d\varepsilon_1^2 + \varepsilon_2$. □

We can get an analogous result for tall and small dense items.

Proof of Lemma 5.2. Suppose the bin contains items J. Then we can use Lemma 5.5 to move dense wide items to box D_W and move dense tall and small items to box D_H. We will later repack one of D_W and D_H into a bin.

$v_1(D_W \cup D_H) \leq 1$. This gives us 2 cases: $v_1(D_W) \leq 1/2$ or $v_1(D_H) \leq 1/2$. The first case is the same as the second one with the coordinate axes swapped, so assume without loss of generality that $v_1(D_W) \leq 1/2$.

Move $C^{(R)}$ to a box of height 1 and width $\varepsilon_1 + \varepsilon_2 \leq 1/2$. $C^{(R)}$ only has tall and small non-dense items. Also, $v_1(C^{(R)}) \leq 1/2$.

Let $I^{(R)}$ be the set of big and wide items that intersect $S^{(R)}$. Move $I^{(R)}$ to a separate bin. The items in $I^{(R)}$ are stacked on top of each other. Therefore, we can round their widths to a multiple of $\varepsilon_1^2/4$. $I^{(R)}$ doesn't have dense items. Therefore, this new bin satisfies the desired properties.

Since we removed $C^{(R)}$ and $I^{(R)}$ from the bin, $S^{(R)}$ is empty. By Lemma 5.4, we can round the x-coordinate and width of big and wide items in the bin to a multiple of $\varepsilon_1^2/4$ and then repack the items in the bin so that the bin satisfies Property 5.7(a) and $S^{(R')}$ is empty. Observe that

$$\delta_d = 2d\varepsilon_1^2 + \varepsilon_2 \leq 2d\varepsilon_1^2 + \frac{\varepsilon\varepsilon_1^2}{2} \leq \frac{\varepsilon_1}{2}(4d+1)\varepsilon_1 \leq \frac{\varepsilon_1}{2}.$$

Since $\delta_d \leq \varepsilon_1/2$, pack D_H into $S^{(R')}$. Now this bin also satisfies Property 5.7(b). In total, we used 2 bins and 2 boxes ($C^{(R)}$ and D_W). The dense box is horizontal and the non-dense box is vertical. Refer to Fig. 5.3 for an example. □

Lemma 5.6. *When item rotation is allowed, given a packing of items into a bin, we can round up the width of some wide and big items to a multiple of $\varepsilon_1^2/4$ and round up the height of some tall and big items to a multiple of $\varepsilon_1^2/4$ and get a packing into 2 bins and 1 box where:*

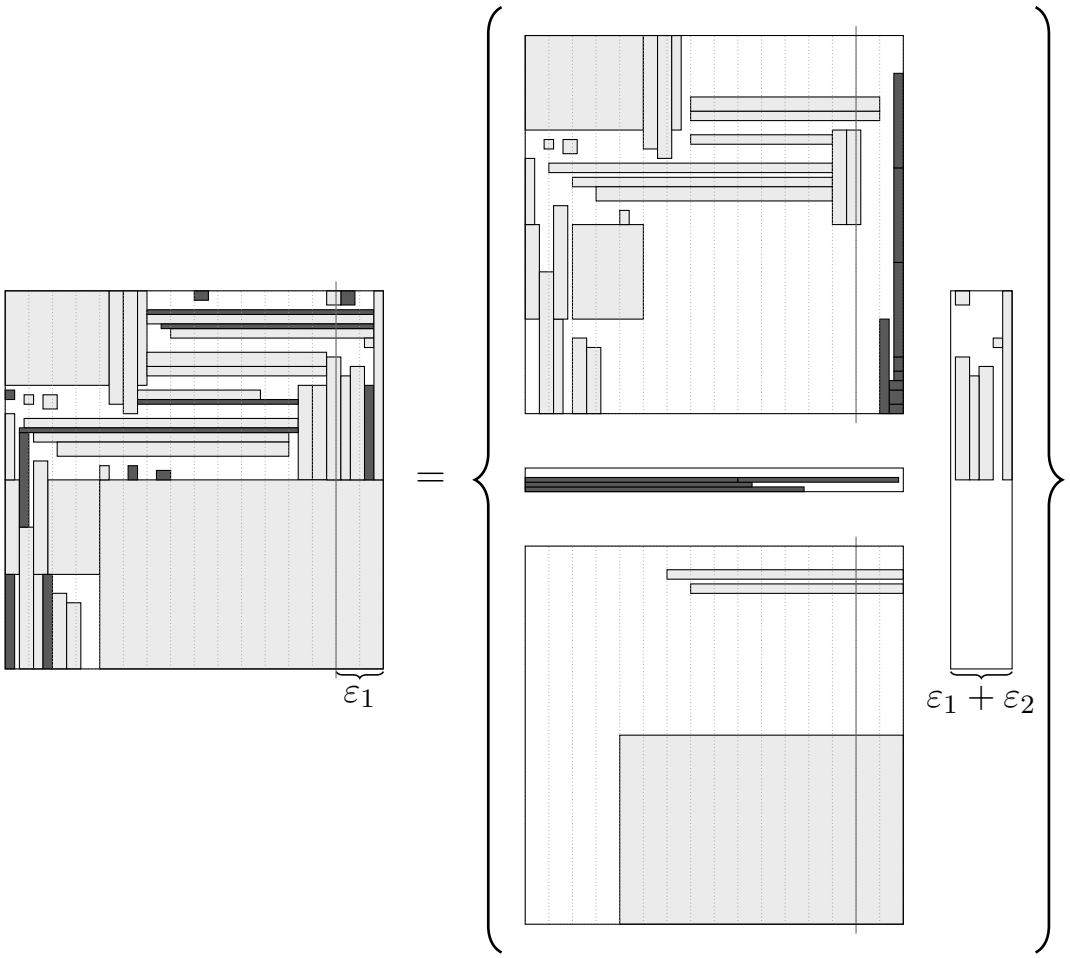


Figure 5.3: A bin is split into 2 bins and 2 boxes. Then the widths and x-coordinates of big and wide non-dense items are rounded up to a multiple of $\varepsilon_1^2/4$. Dense items are shaded dark and non-dense items are shaded light.

- *Each bin satisfies Property 5.7.*
- v_1 *of the box is at most* $1/2$.
- *The box has height 1 and width* $\varepsilon_1 + \varepsilon_2$.
- *The box only contains tall and small non-dense items.*

Proof sketch. Suppose the bin contains items J. Move dense items to a vertical box D_H using Lemma 5.5. Move $C^{(R)}$ to a box of height 1 and width $\varepsilon_1 + \varepsilon_2$. Move $I^{(R)}$ to a new bin and round item widths to a multiple of $\varepsilon_1^2/4$. Now $S^{(R')}$ is empty, so pack D_H into $S^{(R')}$. The rest of the proof is similar to that of Lemma 5.2. □

5.3.2 Getting Slack in Weight of Bins

For a bin J, if $\forall j \in [d], v_j(J) \leq 1 - \varepsilon$, then we can round up weights of items. Hence, we would like to have bins with (roughly) this property.

Definition 5.10. *A bin J is said to be ε-slacked iff at least one of these conditions holds:*

- $\forall j \in [d], v_j(J) \leq 1 - \varepsilon$.
- $|J| = 1$.
- $|J| = 2$ *and J only contains dense items and* $\forall i \in J, v_{\max}(i) \leq 1/2$.

A packing of items into multiple bins is said to be ε-slacked iff all bins in the packing are ε-slacked.

We say that packing of items in a bin is *quarter-structured* iff the bin is ε-slacked and satisfies either Property 5.7 or Property 5.8. We would like to round up the width or height of each item in I and repack the items into bins such that each bin is quarter-structured.

Lemma 5.7. *Let $D \subseteq [d]$ and we have a parameter $\delta \leq 1/4$. Let I be a set of items where $\forall j \in D, v_j(I) \leq V_j$. Then we can partition I into at most $|D| + 1$ disjoint subsets such that each subset I' satisfies one of these properties:*

- $|I'| = 1$.
- $\forall j \in D, v_j(I') \leq (1 - \delta)V_j$.

Proof. Let $I_L := \{i \in I : \exists j \in D, v_j(i) > (1 - 2\delta)V_j\}$. Move each item in I_L to a new box. Let $D' := \{j \in D : v_j(I_L) > (1-2\delta)V_j\}$. Then $|D'| \geq |I_L|$ and $\forall j \in D', v_j(I-I_L) < 2\delta V_j \leq (1-\delta)V_j$.

Order the items in $I - I_L$ arbitrarily. For each $j \in D - D'$, find the smallest prefix P_j such that $v_j(P_j) \geq \delta V_j$. Let i_j be the last item in P_j. Then $v_j(P_j - i_j) < \delta V_j$. Since we removed items from I_L, $v_j(i_j) \leq (1 - 2\delta)V_j$. Therefore, $v_j(P_j) \leq (1 - \delta)V_j$.

Now order these prefixes in non-decreasing order of cardinality. Let them be P'_1, P'_2, …, $P'_{|D-D'|}$. The sets P'_1, $P'_2 - P'_1$, $P'_3 - P'_2$, … form a partition of $P_{|D-D'|}$. Put each such set in a new box, if the set is not empty. The items which remain in the original box are $Q := I - I_L - P'_{|D-D'|}$. $\forall j \in D - D', Q \subseteq I - I_L - P_j$. Since $v_j(P_j) \geq \delta V_j$, we get that $\forall j \in D - D', v_j(Q) \leq (1 - \delta)V_j$.

Therefore, total number of boxes needed is at most $1 + |I_L| + |D - D'| \leq 1 + |D'| + |D - D'| \leq |D| + 1$. □

Lemma 5.8. *Given a packing of items I into m bins, we can round up the width of some non-dense wide and big items in I to a multiple of $\varepsilon_1^2/4$ and round up the height of some non-dense tall and big items in I to a multiple of $\varepsilon_1^2/4$ and repack the items into $(d+4)m$ ε-slacked bins such that each bin satisfies either Property 5.7 or Property 5.8.*

Proof. Let $B_1, B_2, \ldots, B_m$ be a packing of items I into m bins. For each bin B_k, we can use Lemma 5.2 to round up some items in B_k and split B_k into bins J_k and K_k and boxes W_k and H_k. Without loss of generality, assume W_k is a horizontal box. Put each box in a new bin. Then W_k satisfies Property 5.8 and H_k satisfies Property 5.7.

Let $D_k := \{j \in [d] : v_j(J_k) > (1-\varepsilon)\}$, $E_k := \{j \in [d] : v_j(K_k) > (1-\varepsilon)\}$, $F_k := \{j \in [d] : v_j(W_k) > (1-\varepsilon)\}$ and $G_k := \{j \in [d] : v_j(H_k) > (1-\varepsilon)\}$. D_k, E_k, F_k and G_k are pairwise disjoint and they are subsets of $[d]$. Now use Lemma 5.7 with parameter $\delta = \varepsilon$ on bin J_k with dimensions D_k. This splits J_k into $|D_k| + 1$ ε-slacked bins. Similarly, by splitting K_k, W_k and H_k, we get $|E_k| + 1$, $|F_k| + 1$ and $|G_k| + 1$ ε-slacked bins respectively.

The total number of bins from B_k is $|D_k| + |E_k| + |F_k| + |G_k| + 4 \leq d + 4$. Therefore, we get a total of $(d+4)m$ bins.

J_k, K_k, W_k and H_k satisfy the desired properties except possibly ε-slackness. When we split a bin into multiple bins, the position of items relative to the bin isn't changed. Therefore, the split bins continue to satisfy these properties. □

Lemma 5.9. *Given a packing of items I, if item rotations are allowed, we can round up the width of some non-dense wide and big items in I to a multiple of $\varepsilon_1^2/4$ and round up the height of some non-dense tall and big items in I to a multiple of $\varepsilon_1^2/4$ and repack I into $(d+3)m$ ε-slacked bins such that each bin satisfies Property 5.7.*

Proof sketch. Use Lemma 5.6 on each bin in the packing to get 3 bins. The rest of the proof is similar to Lemma 5.8. □

We will now try to improve upon Lemmas 5.8 and 5.9 for the special case $d = 1$.

Lemma 5.10. *Let there be m boxes of width 1 and height at most ε. Let the weight of each box be at most $k\varepsilon$ in each dimension, where $k \in \{1, 2\}$. Then we can pack these boxes into $1 + m \cdot k\varepsilon/(1-k\varepsilon)$ bins such that the resulting bins are $k\varepsilon$-slacked.*

Proof. We can pack $1/k\varepsilon - 1$ boxes in 1 bin with the resulting bin being $k\varepsilon$-slacked. This is because the sum of weights is at most $1 - k\varepsilon$ in each dimension and the total height of the boxes is at most $1/k - \varepsilon \leq 1$. The total number of bins used is at most $\lceil m/((1/k\varepsilon) - 1)\rceil$ which in turn is at most $1 + m \cdot k\varepsilon/(1-k\varepsilon)$. □

The above lemma can also be used for vertical boxes, i.e., height 1 and width at most ε.

Lemma 5.11. *Let $d = 1$. Let there be m boxes of width 1 and height at most ε containing only non-big non-dense items. Let the weight of each box be at most $1/2$. Then we can pack these boxes into $2 + m\,(1/2 + \varepsilon/(1-\varepsilon))$ bins such that the resulting bins are ε-slacked.*

Proof. Let i be an item in the box. Boxes only have non-big items, so $a(i) \leq \varepsilon_2$. Boxes only have non-dense items, so $v_1(i) \leq a(i)/\varepsilon_1^2 \leq \varepsilon/2$.

From each box, choose the smallest prefix S for which $v_1(S) \geq \varepsilon/2$. Then $v_1(S) \leq \varepsilon$. Remove S and put it in a new box of the same dimensions.

This gives us m boxes of weight at most $(1-\varepsilon)/2$. We can pair them up and pack them into $\lceil m/2 \rceil \leq m/2 + 1$ bins. Those bins will be ε-slacked.

We also get m new boxes of weight at most ε. We can pack $1/\varepsilon - 1$ such boxes into a bin. This gives us at most $1 + m \cdot \varepsilon/(1-\varepsilon)$ bins. These bins are ε-slacked.

Total number of bins used is $\left(\frac{m}{2} + 1\right) + (1 + m \cdot \varepsilon/(1-\varepsilon)) = 2 + m\,(1/2 + \varepsilon/(1-\varepsilon))$. □

Lemma 5.12. *Let $d = 1$. Let there be m boxes of width 1 and height δ_d. Suppose the boxes only have dense items, and each box has total weight at least ε and at most $1/2$. Then we can pack the items of these boxes into at most $3 + 2m/3$ bins such that the resulting bins are ε-slacked.*

Proof. Let there be t boxes that have an item of weight at least $1/2 - 2\varepsilon$. No item has weight more than half. Therefore, we can pair up these high-weight items into at most $t/2+1$ ε-slacked bins. In each of these t boxes, the remaining items have weight at most 2ε. Since $\varepsilon \geq \delta_d$, by Lemma 5.10, we can pack them into $1 + 2\varepsilon t/(1-2\varepsilon)$ number of ε-slacked bins.

$m - t$ boxes have all items of weight less than $1/2 - 2\varepsilon$. Each of these boxes has total weight between ε and $1/2$. Each box *can be* split into 2 boxes as follows: Order the items in a box and find the smallest prefix of weight at least ε. Since there are no items of weight more than $1/2 - 2\varepsilon$, such a prefix has weight between ε and $1/2 - \varepsilon$.

Arrange the $m - t$ boxes into groups of at most 3 boxes each. Let C_1, C_2, C_3 be these boxes in one such group. Split C_1 and C_2 into 2 boxes each by the above method. Let the resulting boxes be C_1', C_1'', C_2', C_2'' respectively. Assume without loss of generality that $v_1(C_j') \leq v_1(C_j'')$ for $j \in [2]$. Pack C_1', C_2', C_1'' into 1 bin. It has weight at most $1 - \varepsilon$, so it is ε-slacked. Pack C_2'' and C_3 into 1 bin. It has weight at most $1 - \varepsilon$, so it is ε-slacked. Therefore, we can convert a group of 3 boxes into 2 ε-slacked bins.

Total bins used is at most $(1 + 2\varepsilon t/(1-2\varepsilon)) + 2\lceil (m-t)/3 \rceil \leq 3 + 2m/3 - t\left(\frac{2}{3} - \frac{2\varepsilon}{1-2\varepsilon}\right)$ which is at most $3 + 2m/3$, assuming $\varepsilon \leq 1/5$. □

The above lemma can also be used for vertical boxes, i.e., height 1 and width at most δ_d.

Lemma 5.13. *Given a packing of items I into m bins, when $d = 1$, we can round up the width of some non-dense wide and big items in I to a multiple of $\varepsilon_1^2/4$ and round up the height of some non-dense tall and big items in I to a multiple of $\varepsilon_1^2/4$ and repack I into $\left(3 + \frac{1}{6} + \frac{\varepsilon}{1-\varepsilon}\right) m + 12$ ε-slacked bins such that each bin satisfies either Property 5.7 or Property 5.8.*

Proof. Let $B_1, B_2, \ldots, B_m$ be a packing of items I into m bins. For each bin B_k, we can use Lemma 5.2 to round up some items in B_k and split B_k into bins J_k and K_k and boxes W_k and H_k, where W_k is a horizontal box and H_k is a vertical box, i.e., the width of W_k is 1 and the height of H_k is 1.

Classifying bins:

We will now classify the bins $B_1, B_2, \ldots, B_m$.

Type 1: $v_1(W_k) \leq \varepsilon$ and $v_1(H_k) \leq \varepsilon$:

Among J_k and K_k, at most 1 bin will have weight more than $1 - \varepsilon$. Use Lemma 5.7 to split it into 2 bins. So for each original bin of type 1, we get at most 3 ε-slacked bins and 2 boxes, one horizontal and one vertical, each of total weight at most ε.

Both J_k and K_k satisfy either Property 5.7 or Property 5.8. When we split a bin into multiple bins, the position of items relative to the bin isn't changed. Therefore, the bins continue to satisfy these properties.

Type 2: $v_1(W_k) > \varepsilon$ and $v_1(H_k) \leq \varepsilon$:

$v_1(W_k) > \varepsilon$ implies that $v_1(J_k) \leq 1 - \varepsilon$ and $v_1(K_k) \leq 1 - \varepsilon$, so J_k and K_k are already ε-slacked. Pack W_k in a bin. Since $v_1(W_k) \leq 1/2 \leq 1 - \varepsilon$, W_k is ε-slacked. W_k satisfies Property 5.8. So for each original bin of type 2, we get at most 3 ε-slacked bins and 1 vertical box of weight at most ε.

Type 3: $v_1(W_k) \leq \varepsilon$ and $v_1(H_k) > \varepsilon$:

The analysis is similar to type 2. For each original bin of type 3, we get at most 3 ε-slacked bins and 1 horizontal box of weight at most ε.

Type 4: $v_1(W_k) > \varepsilon$ and $v_1(H_k) > \varepsilon$:

$v_1(W_k) > \varepsilon$ implies that $v_1(J_k) \leq 1 - \varepsilon$ and $v_1(K_k) \leq 1 - \varepsilon$, so J_k and K_k are already ε-slacked. So we have at most 2 ε-slacked bins and 2 boxes of weight at most $1/2$.

Repacking boxes:

We will now try to pack the boxes into bins. Each of these bins packs some dense boxes and some non-dense boxes. If multiple dense boxes were packed in a bin, we can use Lemma 5.5 to repack them into a single dense box and move that box to an edge of the bin. Bins that only pack horizontal boxes satisfy Property 5.8. Bins that only pack vertical boxes satisfy Property 5.7.

Among $B_1, B_2, \ldots, B_m$, let there be m_k bins of type k.

The number of ε-slacked bins is at most $3m_1 + 3m_2 + 3m_3 + 2m_4 \leq 3m - m_4$. We also have $m_1 + m_3$ horizontal boxes and $m_1 + m_2$ vertical boxes of weight at most ε each. Since $\delta_d \leq \varepsilon_1/2 \leq \varepsilon$ and $\varepsilon_1 + \varepsilon_2 \leq 2\varepsilon/3 + 2\varepsilon^3/9 \leq \varepsilon$, each box has the smaller geometric dimension at most ε. By Lemma 5.10, the number of bins we need to pack them is at most $2 + \frac{\varepsilon}{1-\varepsilon}(2m_1 + m_2 + m_3)$.

We have m_4 horizontal boxes and m_4 vertical boxes that each have weight between ε and $1/2$. m_4 of these are dense boxes and m_4 are non-dense boxes.

The non-dense boxes don't have big items. Since $\varepsilon \geq \varepsilon_1 + \varepsilon_2$, by Lemma 5.11, the number of bins needed to pack them is at most $4 + \left(\frac{1}{2} + \frac{\varepsilon}{1-\varepsilon}\right) m_4$.

By Lemma 5.12, we can pack the dense boxes into $6 + 2m_4/3$ bins, where each bin is ε-slacked.

The total number of bins used is at most

$$
\begin{aligned}
&(3m - m_4) + \left(2 + \frac{\varepsilon}{1-\varepsilon}(2m_1 + m_2 + m_3)\right) + \left(4 + \left(\frac{1}{2} + \frac{\varepsilon}{1-\varepsilon}\right) m_4\right) + \left(6 + \frac{2}{3}m_4\right) \\
&= 12 + \left(3 + \frac{1}{6} + \frac{\varepsilon}{1-\varepsilon}\right) m + \frac{\varepsilon}{1-\varepsilon} m_1 + \frac{m_4 - m}{6} \\
&\leq 12 + \left(3 + \frac{1}{6} + \frac{\varepsilon}{1-\varepsilon}\right) m - \left(\frac{1}{6} - \frac{\varepsilon}{1-\varepsilon}\right) m_1 && (m_4 \leq m - m_1) \\
&\leq 12 + \left(3 + \frac{1}{6} + \frac{\varepsilon}{1-\varepsilon}\right) m. && (\varepsilon \leq 1/8)
\end{aligned}
$$

□

Lemma 5.14. *Given a packing of items I into m bins, when $d = 1$ and item rotations are allowed, we can round up the width of some non-dense wide and big items in I to a multiple of $\varepsilon_1^2/4$ and round up the height of some non-dense tall and big items in I to a multiple of $\varepsilon_1^2/4$ and repack I into $\left(3 + \frac{\varepsilon}{1-\varepsilon}\right) m + 1$ ε-slacked bins such that each bin satisfies Property 5.7.*

Proof sketch. Using techniques from the proof of Lemma 5.13, we get at most $3m$ ε-slacked bins and at most m vertical boxes, where each box has total weight at most ε. Using Lemma 5.10, we get the desired results. □

We can summarize Lemmas 5.8, 5.9, 5.13 and 5.14 as follows:

Theorem 5.15. *Given a packing of I into m bins, we can round up the width of some non-dense wide and big items in I to a multiple of $\varepsilon_1^2/4$ and round up the height of some non-dense tall and big items in I to a multiple of $\varepsilon_1^2/4$ and repack I into at most $am + b$ ε-slacked bins such that each bin satisfies either Property 5.7 or Property 5.8. Here the values of a and b depend on the value of d and whether item rotations are allowed. See Table 5.1.*

Table 5.1: Values of a and b for Theorem 5.15.

	a	b	Lemma
rotations forbidden	$d+4$	0	Lemma 5.8
rotations allowed	$d+3$	0	Lemma 5.9
$d=1$ and rotations forbidden	$3+\frac{1}{6}+\frac{\varepsilon}{1-\varepsilon}$	12	Lemma 5.13
$d=1$ and rotations allowed	$3+\frac{\varepsilon}{1-\varepsilon}$	1	Lemma 5.14

5.3.3 Rounding Weights

Definition 5.11 (Weight classes)**.** *Two items i_1 and i_2 are said to belong to the same weight class iff one of these conditions hold:*

- *i_1 and i_2 are big and $\forall j \in [d], v_j(i_1) = v_j(i_2)$*
- *i_1 and i_2 are non-dense and wide and $\forall j \in [d], v_j(i_1)/h(i_1) = v_j(i_2)/h(i_2)$.*
- *i_1 and i_2 are non-dense and tall and $\forall j \in [d], v_j(i_1)/w(i_1) = v_j(i_2)/w(i_2)$.*
- *i_1 and i_2 are non-dense and small and $\forall j \in [d], v_j(i_1)/a(i_1) = v_j(i_2)/a(i_2)$.*
- *i_1 and i_2 are dense and light and $\forall j \in [d], v_j(i_1)/v_{\max}(i_1) = v_j(i_2)/v_{\max}(i_2)$.*
- *i_1 and i_2 are dense and heavy and $\forall j \in [d], v_j(i_1) = v_j(i_2)$.*

Big items that have the same geometric dimensions and the same weight class are identical. We will round the geometric dimensions of dense items to 0, so heavy items of the same weight class would be identical.

Definition 5.12 (Slicing and fractional packing)**.** *Let I be a set of items. $\widehat{I}$ is called a slicing of I iff $\widehat{I}$ can be obtained from I by one or more of the following operations:*

- *Horizontally slicing a non-dense wide item (i.e., if a wide item i is sliced into items i_1 and i_2, then $w(i) = w(i_1) = w(i_2)$ and $h(i) = h(i_1) + h(i_2)$).*
- *Vertically slicing a non-dense tall item (i.e., if a tall item i is sliced into items i_1 and i_2, then $h(i) = h(i_1) = h(i_2)$ and $w(i) = w(i_1) + w(i_2)$).*
- *Slicing a non-dense small item in one or both dimensions.*
- *Slicing a light dense item of zero area.*

A packing of $\widehat{I}$ into bins is called a fractional packing of I.

Wide non-dense items of the same width and of the same weight class can be used interchangeably while trying to get a fractional packing. Similarly, tall non-dense items of the same height and of the same weight class are interchangeable, small non-dense items of the same weight class are interchangeable and light dense items of zero area and the same weight class are interchangeable.

We will now see how to round the weights of items so that they belong to a constant number of weight classes.

5.3.3.1 Rounding Weights of Non-Dense Items

Transformation 5.13. *Given an item i, $\forall j \in [d]$,*

- *If i is big, round up $v_j(i)$ to a positive multiple of $\varepsilon_1^2\varepsilon/8$.*
- *If i is wide and non-dense, round up $v_j(i)$ to a positive multiple of $h(i)\varepsilon_1\varepsilon/8$.*
- *If i is tall and non-dense, round up $v_j(i)$ to a positive multiple of $w(i)\varepsilon_1\varepsilon/8$.*
- *If i is small and non-dense, round up $v_j(i)$ to a positive multiple of $a(i)\varepsilon/8$.*

Lemma 5.16. *Transformation 5.13 is valid, i.e., for any item i, $\forall j \in [d], v_j(i) \leq 1$ after the transformation.*

Proof. Since $\varepsilon_1^{-1}, \varepsilon^{-1} \in \mathbb{Z}$, the transformed weight of a big item can be at most 1.

Since $\varepsilon_2 \leq (1-\varepsilon)\varepsilon_1^2$, the weight of a non-dense wide/tall/small item is at most $\varepsilon_2/\varepsilon_1^2 \leq 1-\varepsilon$ and rounding it up can only increase it by at most $\varepsilon/8$. □

Lemma 5.17. *Let a bin J be μ-slacked, for $\varepsilon/8 \leq \mu \leq \varepsilon$. Then after applying Transformation 5.13, the bin will be $(\mu - \varepsilon/8)$-slacked.*

Proof. If the bin contains a single item, it will remain μ-slacked after the transformation.

Suppose the bin contains multiple items, then $\forall j \in [d], v_j(J) \leq 1 - \mu$. Let there be p big items. Let the total height of wide non-dense items be H. Let the total width of tall non-dense items be W. Let the total area of small non-dense items be A.

The total area of big items is at least $p\varepsilon_1^2$, of wide items is at least $\varepsilon_1 H$ and of tall items is at least $\varepsilon_1 W$. Since the total area of items in J is at most 1,

$$\varepsilon_1^2 p + \varepsilon_1 H + \varepsilon_1 W + A \leq 1.$$

The total increase in $v_j(J)$ is at most

$$\frac{\varepsilon}{8}(\varepsilon_1^2 p + \varepsilon_1 H + \varepsilon_1 W + A) \leq \frac{\varepsilon}{8}.$$

Therefore, the resulting bin is $(\mu - \varepsilon/8)$-slacked. □

Observation 5.18. *Since we round up weights of non-dense items in Transformation 5.13, some items may not continue to satisfy the non-denseness property, i.e., $v_j(i)/a(i)$ may exceed $1/\varepsilon_1^2$ for some items i and some $j \in [d]$.*

However, this will not affect us much. Formally:

(a) Big items will remain non-dense, since $a(i) > \varepsilon_1^2$ and $v_{\max}(i) \leq 1$.

(b) Small items will remain non-dense, since $v_{\max}(i)/a(i)$ will be rounded up to a multiple of $\varepsilon/8$, and $\varepsilon/8$ divides $1/\varepsilon_1^2$.

(c) For wide items, $v_{\max}(i)/a(i)$ may rise to at most $1/\varepsilon_1^2 + \varepsilon/8$. Furthermore, $v_{\max}(i)/h(i)$ will be rounded to a multiple of $\varepsilon_1\varepsilon/8$, and $\varepsilon_1\varepsilon/8$ divides $1/\varepsilon_1^2$, so $v_{\max}(i)/h(i)$ will continue to be at most $1/\varepsilon_1^2$.

(d) For tall items, $v_{\max}(i)/a(i)$ may rise to at most $1/\varepsilon_1^2+\varepsilon/8$. Furthermore, $v_{\max}(i)/w(i)$ will be rounded to a multiple of $\varepsilon_1\varepsilon/8$, and $\varepsilon_1\varepsilon/8$ divides $1/\varepsilon_1^2$, so $v_{\max}(i)/w(i)$ will continue to be at most $1/\varepsilon_1^2$.

Even if $v_{\max}(i)/a(i)$ of some non-dense items exceeds $1/\varepsilon_1^2$ after Transformation 5.13, we will continue to consider them non-dense items.

Lemma 5.19. *Define $n_{\text{bwc}} := (8/(\varepsilon_1^2\varepsilon))^d, n_{\text{wwc}} := (8/(\varepsilon_1^3\varepsilon))^d, n_{\text{swc}} := (8/(\varepsilon_1^2\varepsilon))^d$. After Transformation 5.13, the number of weight classes of big items is at most n_{bwc}, of wide non-dense items is at most n_{wwc}, of tall non-dense items is at most n_{wwc} and of small non-dense items is at most n_{swc}.*

5.3.3.2 Rounding Weights of Dense Items

Transformation 5.14. *For item i, if i is non-dense, do nothing. If i is dense, set $w(i)$ and $h(i)$ to 0 and for each $j \in [d]$, if $v_j(i) \leq (\varepsilon/8d)v_{\max}(i)$, then set $v_j(i)$ to 0.*

Since Transformation 5.14 rounds down weights, we need to prove that we can easily undo this transformation.

Lemma 5.20. *Let J be a set of items. Let J' be the items obtained by applying Transformation 5.14 to J. Suppose we're given a packing of J' into a bin that satisfies Property 5.7(b) and is μ-slacked, for some $\mu \geq \varepsilon/8$.*

Then there is a polynomial-time algorithm to convert the packing of J' into a packing of J that satisfies Property 5.7(b), is $(\mu - \varepsilon/8)$-slacked, and the position of non-dense items in the packing of J is the same as the position of non-dense items in the packing of J'.

(Analogous lemma holds for Property 5.8(b))

Proof. By Lemma 5.5, we can always pack dense items in J in polynomial time into a box of height 1 and width δ_d. Since $\delta_d \leq \varepsilon_1/2$, this box fits in $S^{(R')}$. Therefore, Property 5.7(b) is satisfied.

Now we will prove that the packing of J is $(\mu - \varepsilon/8)$-slacked. There are 3 cases to consider:

Case 1: $\forall j \in [d], v_j(J') \leq 1 - \mu$:

For each $j \in [d]$ and each dense item i, reverting the transformation increases $v_j(i)$ by at most $(\varepsilon/8d)v_{\max}(i)$. So by Lemma 4.2, we get

$$v_j(J) \leq v_j(J') + \frac{\varepsilon}{8d} v_{\max}(J') \leq v_j(J') + \frac{\varepsilon}{8} \leq 1 - \left(\mu - \frac{\varepsilon}{8}\right).$$

Therefore, J is $(\mu - \varepsilon/8)$-slacked.

Case 2: $|J'| = 1$:

Then $|J| = 1$, so J is μ-slacked.

Case 3: $|J'| = 2$ and J' only has dense items and $\forall i \in J', 1/2 - \mu \leq v_{\max}(i) \leq 1/2$:

The 0-value dimensions of i increase to $(\varepsilon/8d)v_{\max}(i) \leq v_{\max}(i)$, so $v_{\max}(i)$ remains the same across this transformation. So J is μ-slacked. □

As the first step in rounding dense items, we apply Transformation 5.14 to I.

Since $\varepsilon_1 \leq 1/4d$, we get $\varepsilon_2 \leq (\varepsilon/8d)\varepsilon_1$. Hence, Transformation 5.14 forces heavy items to be heavy in all non-zero dimensions.

Now we will use different transformations on heavy and light items.

Transformation 5.15. *For a dense item i, if $v_j(i) > \varepsilon_1$, round up $v_j(i)$ to a multiple of $\varepsilon_1\varepsilon/8$.*

Lemma 5.21. *Let J be a packing of items into a bin that is μ-slacked, for some $\mu \geq \varepsilon/8$. Let J' be the packing obtained by applying Transformation 5.15 to dense items in J. Then J' is a $(\mu - \varepsilon/8)$-slacked packing.*

Proof. **Case 1**: $\forall j \in [d], v_j(J') \leq 1 - \mu$:

For each $j \in [d]$, there are less than ε_1^{-1} items i in J such that $v_j(i) > \varepsilon_1$. For each such item, $v_j(i)$ increases by less than $\varepsilon_1\varepsilon/8$. Therefore, $v_j(J') < v_j(J) + \varepsilon/8$. Therefore, J is $(\mu - \varepsilon/8)$-slacked.

Case 2: $|J| = 1$:
$|J'| = 1$, so J' is μ-slacked.

Case 3: $|J| = 2$ and J only has dense items and $\forall i \in J, 1/2 - \mu \le v_{\max}(i) \le 1/2$:
$\varepsilon_1^{-1}\varepsilon^{-1} \in \mathbb{Z}$, so $1/2$ is a multiple of $\varepsilon_1\varepsilon/8$. Therefore, for each item i, $v_{\max}(i)$ increases to at most $1/2$. So J is μ-slacked. □

Lemma 5.22. *The number of distinct heavy items after Transformation 5.15 is at most*

$$n_{\text{hwc}} := \left(\frac{8}{\varepsilon}\left(\frac{1}{\varepsilon_1} - 1\right)\right)^d.$$

Proof. This is because large vector dimensions are rounded to at most $8/\varepsilon_1\varepsilon - 8/\varepsilon$ distinct values. □

Transformation 5.16. *For a dense item i, if $v_{\max}(i) \le \varepsilon_2$, then for each $j \in [d]$, round up $v_j(i)/v_{\max}(i)$ to a power of $1/(1+\varepsilon/8)$ if $v_j(i) > 0$.*

Lemma 5.23. *Let J be a packing of items into a bin that is μ-slacked, for some $\varepsilon/8 \le \mu \le \varepsilon$. Let J' be the packing obtained by applying Transformation 5.16 to dense items in J. Then J' is a $(\mu - \varepsilon/8)$-slacked packing.*

Proof. **Case 1**: $\forall j \in [d], v_j(J') \le 1 - \mu$:
For each $j \in [d]$, $v_j(i)$ increases by a factor of at most $1 + \varepsilon/8$. So,

$$v_j(J') \le v_j(J)\left(1 + \frac{\varepsilon}{8}\right) \le v_j(J) + \frac{\varepsilon}{8}.$$

Therefore, J' is $(\mu - \varepsilon/8)$-slacked.

Case 2: $|J| = 1$:
$v_{\max}(i) \le 1$ after the transformation, so the packing is valid and $|J'| = 1$. Therefore, J' is μ-slacked.

Case 3: $|J| = 2$ and J only has dense items and $\forall i \in J, 1/2 - \mu \le v_{\max}(i) \le 1/2$:
Since $\varepsilon_2 \le 1/2 - \varepsilon \le 1/2 - \mu$ and the transformation only applies when $v_{\max}(i) \le \varepsilon_2$, J remains the same after the transformation. □

Lemma 5.24. *After Transformation 5.16, the number of distinct values of the weight vector of light items is at most*

$$\left\lceil \frac{\ln(8d/\varepsilon)}{\ln(1+\varepsilon/8)} \right\rceil^{d-1} \le \left\lceil \frac{8+\varepsilon}{\varepsilon} \ln\left(\frac{8d}{\varepsilon}\right) \right\rceil^{d-1} := n_{\text{lwc}}.$$

Proof. This is because $v_j(i)/v_{\max}(i)$ is lower-bounded by $\varepsilon/8d$ because of Transformation 5.14. □

Transformation 5.17 (Weight-rounding)**.** *For a set I of items, weight-rounding is the process of applying Transformations 5.13, 5.14, 5.15 and 5.16 to all items. A set I of items is said to be* weight-rounded *iff I is invariant under Transformations 5.13, 5.14, 5.15 and 5.16.*

5.3.4 Rounding the Other Side

So far, for each item, we have seen how to round their weights and how to round one geometric dimension. In this subsection, we will see how to use linear grouping to round the other geometric dimension. We will show that after this operation, the items will belong to a constant number of homogeneous classes (see Condition C1.3 in Section 4.3.4).

5.3.4.1 Rounding Geometric Dimensions of Non-Dense Items

Transformation 5.18 (Linear grouping)**.** *Suppose we are given a packing of items I into m bins, where I is invariant under Transformation 5.13 and each bin satisfies Property 5.7(a).*

Partition the big items in I by their width and weight class. Partition the tall non-dense items in I by their weight class. The number of partitions is constant by Property 5.7(a) and Lemma 5.19. Let $\delta_{\lg} := \varepsilon\varepsilon_1/(d+1)$ (so $\delta_{\lg}^{-1} \in \mathbb{Z}$).

For each partition S of big items, do the following:

1. *Order the items in S in non-increasing order of height.*
2. *Let $k := \lfloor \delta_{\lg}|S| \rfloor + 1$. Let S_1 be the first k items, S_2 be the next k items, and so on, till S_T, where $T = \lceil |S|/k \rceil \leq 1/\delta_{\lg}$. For $t \in [T]$, S_t is called the t^{th} linear group of S. The first item in S_t is called the* leader *of S_t, denoted as* $\text{leader}(S_t)$.
3. *Increase the height of each item in S_1 to $h(\text{leader}(S_1))$. Unpack the items $S_1 - \{\text{leader}(S_1)\}$.*
4. *For each $t \in [T] - \{1\}$ and each $j \in [|S_t|] - \{1\}$, let i be the j^{th} item in S_t and let i' be the j^{th} item in S_{t-1}. Note that $h(i') \geq h(\text{leader}(S_t)) \geq h(i)$. Increase $h(i)$ to $h(\text{leader}(S_t))$ and pack i where i' was packed. Since i has the same width and weights as i', the geometric constraints are not violated and the total weights of the bin do not increase. The number of distinct heights in S now becomes at most $T \leq 1/\delta_{\lg}$.*

For each partition S of tall items, do the following (see Fig. 5.4):

1. *Order the items in S in non-increasing order of height and arrange them side-by-side on the real line, starting from the origin.*

2. *Let the total width of S be W. Let S_t be the items in the interval $[(t-1)\delta_{\lg} W, t\delta_{\lg} W]$. Slice the items if they lie on the boundaries of the interval. S_t is called the t^{th} linear group of S. The first item in S_t is called the leader of S_t, denoted as* $\text{leader}(S_t)$.

3. *Increase the height of each item in S_1 to $h(\text{leader}(S_1))$. Then unpack S_1.*

4. *For each $t \in [1/\delta_{\lg}] - \{1\}$, move the items in S_t to the space occupied by S_{t-1} (items in S_t may need to be sliced for this) and increase the height of each item $i \in S_t$ to $h(\text{leader}(S_t))$. This doesn't violate geometric constraints since S_t and S_{t-1} have the same total width and this doesn't increase the total weights of the bin because all items in S have the same weight class. The number of distinct heights in S now becomes at most $1/\delta_{\lg}$.*

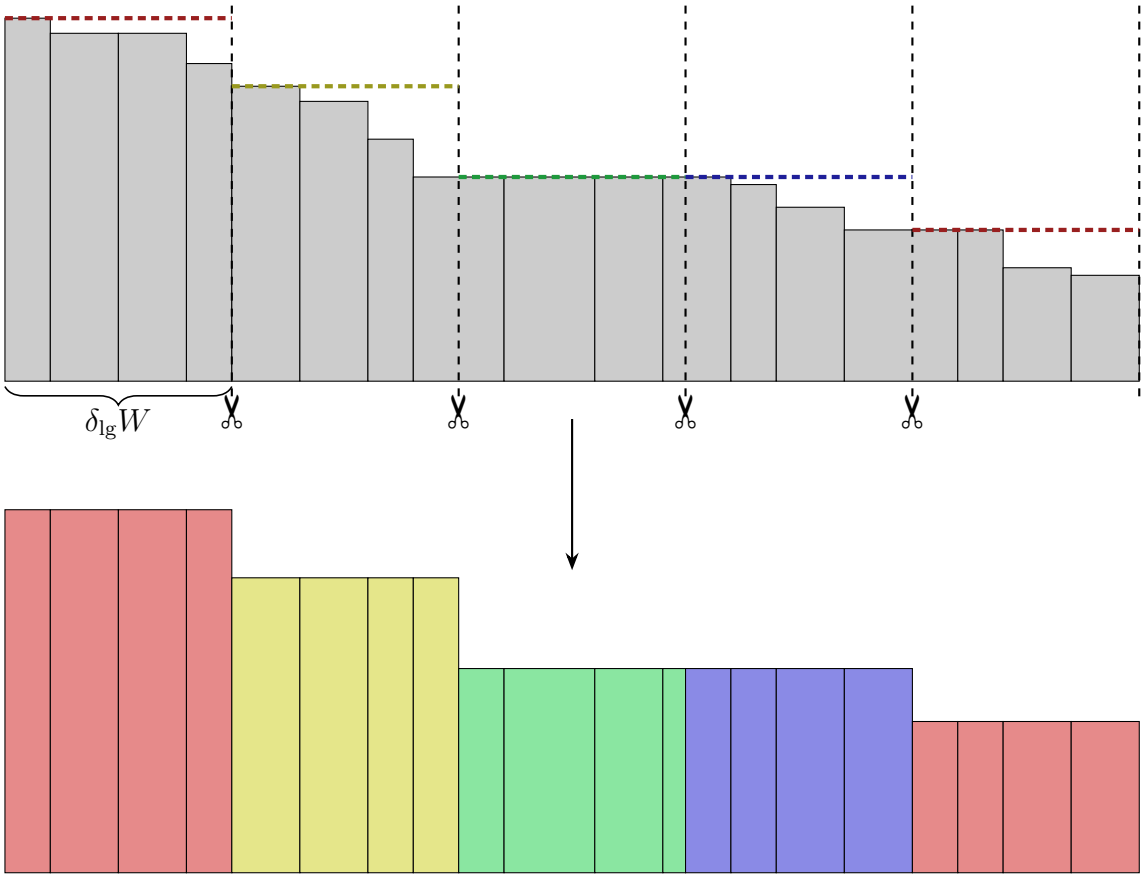


Figure 5.4: Linear grouping tall items. Here $\delta_{\lg} = 5$.

We can extend the definition of this transformation to bins satisfying Property 5.8(a) by swapping vertical and horizontal directions. (Partition big items by height and weight class and

partition wide items by weight class. Then round up the width of items in each partition using the above techniques.)

Lemma 5.25. *Let J be a set of big and tall items and let $\mu \in [0, 1)$ be a constant. Define* $\text{span}'(i) := \max\left(w(i), \min\left(\frac{v_{\max}(i)}{1-\mu}, 1\right)\right)$. *Then $\sum_{i\in J} \text{span}'(i) \le 1$ implies J can be packed into a μ-slacked bin where all items touch the bottom of the bin.*

Proof. $\sum_{i\in J} w(i) \le \sum_{i\in J} \text{span}'(i) \le 1$.

$\forall i \in J, \text{span}'(i) > 0$. So if $\text{span}'(i) = 1$ for some $i \in J$, then $|J| = 1$. So J can be packed into a bin, and the bin is μ-slacked since $|J| = 1$.

Now let $\text{span}'(i) < 1$ for all $i \in J$. So $v_{\max}(i) < 1 - \mu$ and $\forall j \in [d]$,

$$v_j(J) = \sum_{i\in J} v_j(i) \le (1-\mu) \sum_{i\in J} \frac{v_{\max}(i)}{1-\mu} \le (1-\mu) \sum_{i\in J} \text{span}'(i) \le 1 - \mu.$$

Therefore, J can be packed into a μ-slacked bin. □

Lemma 5.26. *Suppose we are given a packing of items I into m bins, where I is invariant under Transformation 5.13 and each bin satisfies Property 5.7(a). Let U be the items unpacked by linear grouping (Transformation 5.18). Then U can be repacked into $\frac{2\varepsilon}{1-\varepsilon}m + 1$ number of ε-slacked bins that satisfy Property 5.7.*

Proof. Define $\text{span}'(i) := \max\left(w(i), \min\left(\frac{v_{\max}(i)}{1-\varepsilon}, 1\right)\right)$. Let K be the set of big and tall non-dense items in I. For any $J \subseteq K$, define $\text{span}'(J) := \sum_{i\in J} \text{span}'(i)$.

Interpret each item $i \in U$ as a 1D item of size $\text{span}'(i)$. Order the items such that big items in U appear before tall non-dense items in U. Pack them on the bottom of new bins using the Next-Fit algorithm. By Lemma 5.25, they will require at most $2\,\text{span}'(U) + 1$ ε-slacked bins. These bins satisfy Property 5.7(a) since the width of all big items in U is a multiple of $\varepsilon_1^2/4$, and they satisfy Property 5.7(b) since U only contains non-dense items.

In Transformation 5.18, we partitioned all big items in I by width and weight class. Let $S \subseteq I$ be one such partition. Given the way we created the groups $S_1, S_2, \ldots$, we get $|S \cap U| \le \lfloor \delta_{\text{lg}} |S| \rfloor$. Since all items in S have the same width and weights, $\text{span}'(i)$ is the same for each $i \in S$. Therefore,

$$\text{span}'(S \cap U) = \text{span}'(i)|S \cap U| \le \text{span}'(i)\, \lfloor \delta_{\text{lg}} |S| \rfloor \le \delta_{\text{lg}}\, \text{span}'(S).$$

In Transformation 5.18, we partitioned all tall non-dense items in I by weight class. Let $S \subseteq I$ be one such partition. Given the way we created the groups $S_1, S_2, \ldots$, we get $w(S \cap$

$U) = \delta_{\lg} w(S)$. All items in S have the same weights-to-width ratio, which is at most $1/\varepsilon_1^2$ by Observation 5.18(d). Since $\varepsilon_2 \le \varepsilon_1^2(1-\varepsilon)$, we get $v_j(i) \le 1-\varepsilon$ for all $i \in S$, so $\operatorname{span}'(i)/w(i)$ is the same for each $i \in S$. Let that common ratio be α. Then,

$$\operatorname{span}'(S \cap U) = \alpha w(S \cap U) \le \alpha \delta_{\lg} w(S) = \delta_{\lg} \operatorname{span}'(S).$$

Summing over all partitions S, we get

$$\operatorname{span}'(U) = \sum_S \operatorname{span}'(U \cap S) \le \sum_S \delta_{\lg} \operatorname{span}'(S) \le \delta_{\lg} \operatorname{span}'(K). \tag{5.1}$$

For $i \in K$, we get

$$\frac{\operatorname{span}'(i)}{\operatorname{span}(i)} \le \frac{\max\left(w(i), \frac{v_{\max}(i)}{1-\mu}\right)}{\max\left(w(i)h(i), v_{\max}(i)\right)} \le \frac{\frac{1}{1-\mu}\max\left(w(i), v_{\max}(i)\right)}{\max\left(w(i)\varepsilon_1, v_{\max}(i)\right)} \le \frac{1}{(1-\mu)\varepsilon_1}.$$

The last inequality follows because for big and tall items, $h(i) \ge \varepsilon_1$.

The number of bins used to pack U is

$$\begin{aligned} 2\operatorname{span}'(U) + 1 &\le 2\delta_{\lg} \operatorname{span}'(K) + 1 \le \frac{2\delta_{\lg}}{\varepsilon_1(1-\varepsilon)} \operatorname{span}(K) + 1 \\ &\le \frac{2(d+1)\delta_{\lg}}{\varepsilon_1(1-\varepsilon)} m + 1 = \frac{2\varepsilon}{1-\varepsilon} m + 1. \end{aligned}$$

The first inequality follows from (5.1) and the third inequality follows from Lemma 4.1. □

Lemma 5.27. *Suppose we are given a packing of items I into m bins, where I is weight-rounded, each bin is μ-slacked for some $\mu \le \varepsilon$, and each bin satisfies either Property 5.7 or Property 5.8. Then after applying linear grouping (Transformation 5.18) to this packing of I, we get a packing of items $\widehat{I}$ into m' bins, where all of the following hold:*

- *$\widehat{I}$ is a rounding-up of I and contains a constant number of homogeneous classes (see Condition C1.3 in Section 4.3.4).*
- *Each bin in the packing of $\widehat{I}$ is μ-slacked and satisfies either Property 5.7 or Property 5.8.*
- $m' \le \left(1 + \dfrac{2\varepsilon}{1-\varepsilon}\right) m + 2.$

Proof sketch. Follows from the definition of linear grouping and Lemma 5.26. Note that we apply linear grouping separately to bins satisfying Property 5.7 and bins satisfying Property 5.8. □

5.3.4.2 Coarse and Fine Partitioning

Our approach so far has been to start from an optimal packing of items and show how to modify it to obtain an approximately-optimal structured packing of a rounded instance. However, the rounding algorithm must round items without knowing the optimal packing. To design such an algorithm, we first need to introduce additional concepts: *coarse partitioning* and *fine partitioning*.

At a high level, our rounding algorithm first partitions the items by weight classes to get a *coarse partitioning*. It then further partitions the coarse partitions to get a *fine partitioning*. It then rounds up the geometric dimensions of items in each fine partition to make that partition homogeneous.

We will first formally define coarse and fine partitioning. We will then restate Theorem 5.15 and Lemma 5.27 using the language of fine partitioning. Then in Section 5.4, we will see an algorithm for computing a fine partitioning of I.

- Let $B(I)$ be the set of big items in I.
- Let $W(I)$ be the set of wide non-dense items in I.
- Let $H(I)$ be the set of tall non-dense items in I.
- Let $S(I)$ be the set of small non-dense items in I.
- Let $D^{l,1}(I)$ be the set of light dense items in I that are either tall or small.
- Let $D^{l,2}(I)$ be the set of light dense wide items in I.
- Let $D^{h,1}(I)$ be the set of heavy dense items in I that are either tall or small.
- Let $D^{h,2}(I)$ be the set of heavy dense wide items in I.

When the set of items I is clear from context, we will use B, W, H, S, $D^{l,1}$, $D^{l,2}$, $D^{h,1}$, $D^{h,2}$ to refer to these sets.

Definition 5.19 (Coarse partitioning)**.** *Let I be a weight-rounded instance. Partition items I by their weight classes. Then for each partition containing dense items, split that partition into 2 partitions: one containing only tall and small items and the other containing only wide items. The resulting partitioning is called a coarse partitioning of I.*

We number the coarse partitions in B arbitrarily from 1 onwards. There will be at most n_{bwc} such partitions by Lemma 5.19. Denote the p^{th} coarse partition by B_p.

Similarly, denote the p^{th} coarse partition

- *in W by W_p, where $p \in [n_{\text{wwc}}]$.*
- *in H by H_p, where $p \in [n_{\text{wwc}}]$.*

- *in S by S_p, where $p \in [n_{\text{swc}}]$.*
- *in $D^{l,1}$ by $D^{l,1}_p$, where $p \in [n_{\text{lwc}}]$.*
- *in $D^{l,2}$ by $D^{l,2}_p$, where $p \in [n_{\text{lwc}}]$.*
- *in $D^{h,1}$ by $D^{h,1}_p$, where $p \in [n_{\text{hwc}}]$.*
- *in $D^{h,2}$ by $D^{h,2}_p$, where $p \in [n_{\text{hwc}}]$.*

Observation 5.28. *There is a unique coarse partitioning of I. Furthermore, the unique coarse partitioning can be found in $O(|I|)$ time.*

In Theorem 5.15 and Lemma 5.27, widths of wide and big items are rounded. The rounding is different for Theorem 5.15 and Lemma 5.27: In Theorem 5.15, we round the widths of some items to multiples of $\varepsilon_1^2/4$ so that the bin satisfies Property 5.7(a), and in Lemma 5.27, we round the widths of items in bins satisfying Property 5.8(a) using linear grouping. To get a rounding algorithm, we have to guess whether the bin of a wide or big item will satisfy Property 5.7 or Property 5.8. We will capture these guesses in the fine partitioning.

Definition 5.20 (Fine partitioning)**.** *Let $Q := \mathbb{Z} \cap \left[\frac{4}{\varepsilon_1} + 1, \frac{4}{\varepsilon_1^2}\right]$, $R := \left\{1, 2, \ldots, \frac{1}{\delta_{\text{lg}}}\right\}$, $Q_q := \left\{x \in \mathbb{R} : (q-1)\frac{\varepsilon_1^2}{4} < x \leq q\frac{\varepsilon_1^2}{4}\right\}$.*

Given a coarse partitioning of a set I of items, let (B^w_p, B^h_p) be a partitioning of B_p, (W^w_p, W^h_p) be a partitioning of W_p and (H^w_p, H^h_q) be a partitioning of H_p.

- *B^w_p is partitioned into sets $\{B^w_{p,q,r} : q \in Q, r \in R\}$ where $i \in B^w_{p,q,r} \implies w(i) \in Q_q$.*
- *B^h_p is partitioned into sets $\{B^h_{p,q,r} : q \in Q, r \in R\}$ where $i \in B^h_{p,q,r} \implies h(i) \in Q_q$.*
- *W^w_p is partitioned into sets $\{W^w_{p,q} : q \in Q\}$ where $i \in W^w_{p,q} \implies w(i) \leq q\varepsilon_1^2/4$.*
- *W^h_p is partitioned into sets $\{W^h_{p,r} : r \in R\}$.*
- *H^w_p is partitioned into sets $\{H^w_{p,r} : r \in R\}$.*
- *H^h_p is partitioned into sets $\{H^h_{p,q} : q \in Q\}$ where $i \in H^h_{p,q} \implies h(i) \leq q\varepsilon_1^2/4$.*

A fine partitioning of I is any partitioning of I into sets of the form $B^w_{p,q,r}$, $B^h_{p,q,r}$, $W^w_{p,q}$, $W^h_{p,r}$, $H^w_{p,r}$, $H^h_{p,q}$, S_p, $D^{l,1}_p$, $D^{l,2}_p$, $D^{h,1}_p$, $D^{h,2}_p$.

Note that for a given set I of items, there can be multiple fine partitionings.

Given a fine partitioning, we use the '$*$' character in superscript or subscript to denote the union of some partitions. For example, $B^w_{p,*,r} := \bigcup_q B^w_{p,q,r}$ and $W^w_{*,*} := \bigcup_{p,q} W^w_{p,q}$, and $D^{*,1}_p := D^{l,1}_p \cup D^{h,1}_p$.

When item rotations are allowed, the fine partitioning includes information on which items to rotate, and we can assume without loss of generality that $H(I) = D^{*,2} = B^h_{*,*,*} = W^h_{*,*} = H^h_{*,*} = \{\}$.

Transformation 5.21. *Given a fine partitioning of I, execute the following operations:*

- $\forall i \in B^w_{*,q,*} \cup W^w_{*,q}$, *increase* $w(i)$ *to* $q\varepsilon_1^2/4$.
- $\forall i \in B^h_{*,q,*} \cup H^h_{*,q}$, *increase* $h(i)$ *to* $q\varepsilon_1^2/4$.
- $\forall i \in B^w_{p,q,r}$, *increase* $h(i)$ *to* $\max_{i \in B^w_{p,q,r}} h(i)$.
- $\forall i \in B^h_{p,q,r}$, *increase* $w(i)$ *to* $\max_{i \in B^h_{p,q,r}} w(i)$.
- $\forall i \in W^h_{p,r}$, *increase* $w(i)$ *to* $\max_{i \in W^h_{p,r}} w(i)$.
- $\forall i \in H^w_{p,r}$, *increase* $h(i)$ *to* $\max_{i \in H^w_{p,r}} h(i)$.

The number of fine partitions is constant and after applying Transformation 5.21, each partition is homogeneous.

Definition 5.22 (Semi-structured packing of fine partitioning)**.** *Suppose we are given a fine partitioning of items I. A packing of items $J \subseteq I$ into a bin is said to be 'division-1 semi-structured' with respect to the fine partitioning iff J doesn't contain items from $B^h_{*,*,*}$, $W^h_{*,*}$, $H^h_{*,*}$ and $D^{*,2}$ and J satisfies Property 5.7.*

A packing of items $J \subseteq I$ into a bin is said to be 'division-2 semi-structured' with respect to the fine partitioning iff J doesn't contain items from $B^w_{*,*,*}$, $W^w_{*,*}$, $H^w_{*,*}$ and $D^{*,1}$ and J satisfies Property 5.8.

Packing of items into bins is called semi-structured iff each bin is either division-1 semi-structured or division-2 semi-structured.

Definition 5.23 (Balanced fine partitioning)**.** *A fine partitioning is said to be* balanced *iff it satisfies all of the following conditions:*

- $\forall p, \forall r, h(W^h_{p,r}) = \delta_{\lg} h(W^h_{p,*})$
- $\forall p, \forall r, w(H^w_{p,r}) = \delta_{\lg} w(H^w_{p,*})$
- $\forall p, \forall q$, *the sets* $\{B^w_{p,q,r} : \forall r\}$ *can be obtained from* $B^w_{p,q,*}$ *by ordering the items in* $B^w_{p,q,*}$ *in non-increasing order of height (breaking ties arbitrarily) and putting the first k items in* $B^w_{p,q,1}$, *the next k items in* $B^w_{p,q,2}$, *and so on, where* $k := \lfloor \delta_{\lg} |B^w_{p,q,*}| \rfloor + 1$.

- $\forall p, \forall q$, *the sets* $\{B^h_{p,q,r} : \forall r\}$ *can be obtained from* $B^h_{p,q,*}$ *by ordering the items in* $B^h_{p,q,*}$ *in non-increasing order of width (breaking ties arbitrarily) and putting the first* k *items in* $B^h_{p,q,1}$*, the next* k *items in* $B^h_{p,q,2}$*, and so on, where* $k := \lfloor \delta_{\lg} |B^h_{p,q,*}| \rfloor + 1$.

We now restate Theorem 5.15 and Lemma 5.27 in terms of fine partitioning.

Lemma 5.29. *Let* I *be a set of items and* $\widehat{I}$ *be the items obtained by weight-rounding* I*. Then there exists a balanced fine partitioning of a slicing of* $\widehat{I}$ *such that after applying Transformation 5.21 to* $\widehat{I}$*, there is a semi-structured* $(5\varepsilon/8)$*-slacked fractional packing of* $\widehat{I}$ *into*

$$\left(1 + \frac{2\varepsilon}{1-\varepsilon}\right)(a\,\mathrm{opt}(I) + b) + 2$$

bins. Here a *and* b *are as defined in Table 5.1 in Theorem 5.15.*

Proof. By Theorem 5.15, we can round up the width of some big and wide non-dense items in I to the nearest multiple of $\varepsilon_1^2/4$ and round up the height of some big and tall non-dense items in I to the nearest multiple of $\varepsilon_1^2/4$ and then pack I into $a\,\mathrm{opt}(I) + b$ ε-slacked bins such that each bin satisfies either Property 5.7 or Property 5.8. Let $\mathcal{B}$ be such a bin packing of I. By Lemmas 5.17, 5.21 and 5.23, $\mathcal{B}$ gives us a $(5\varepsilon/8)$-slacked packing of $\widehat{I}$.

Call the bins in $\mathcal{B}$ that satisfy Property 5.7 division-1 bins. Call the rest of the bins division-2 bins. The items whose width needs to be rounded up to a multiple of $\varepsilon_1^2/4$ are the big and wide items in division-1 bins and the items whose height needs to be rounded up to a multiple of $\varepsilon_1^2/4$ are the big and tall items in division-2 bins. No other items need to have their width or height rounded up in the packing $\mathcal{B}$ produced by Theorem 5.15.

Let $\widehat{I}^w$ and $\widehat{I}^h$ be the items of $\widehat{I}$ in division-1 bins and division-2 bins respectively.

We can compute the coarse partitioning of $\widehat{I}$. Define

$$B^w_p := B_p \cap \widehat{I}^w \qquad W^w_p := W_p \cap \widehat{I}^w \qquad H^w_p := H_p \cap \widehat{I}^w$$
$$B^h_p := B_p \cap \widehat{I}^h \qquad W^h_p := W_p \cap \widehat{I}^h \qquad H^h_p := H_p \cap \widehat{I}^h$$

Define

- $B^w_{p,q} := \{i \in B^w_p : (q-1)\varepsilon_1^2/4 < w(i) \le q\varepsilon_1^2/4\}$.
- $B^h_{p,q} := \{i \in B^h_p : (q-1)\varepsilon_1^2/4 < h(i) \le q\varepsilon_1^2/4\}$.
- $W^w_{p,q} := \{i \in W^w_p : (q-1)\varepsilon_1^2/4 < w(i) \le q\varepsilon_1^2/4\}$.
- $H^h_{p,q} := \{i \in H^h_p : (q-1)\varepsilon_1^2/4 < h(i) \le q\varepsilon_1^2/4\}$.

Define

- $B^w_{p,q,r}$ as the r^{th} linear group of $B^w_{p,q}$ (see Transformation 5.18).
- $B^h_{p,q,r}$ as the r^{th} linear group of $B^h_{p,q}$.
- $W^h_{p,r}$ as the r^{th} linear group of W^h_p.
- $H^w_{p,r}$ as the r^{th} linear group of H^w_p.

This is how we get a fine partitioning of a slicing of $\widehat{I}$.

As per Lemma 5.27, on applying Transformation 5.21 to $\widehat{I}$, the resulting instance can be sliced and packed into $\left(1 + \frac{2\varepsilon}{1-\varepsilon}\right)(a\,\text{opt}(I) + b) + 2$ number of $(5\varepsilon/8)$-slacked bins. □

5.4 Rounding Algorithm

Let I be a set of weight-rounded items. To use Lemma 5.29 to get an approximately-optimal packing of items I, we would like to iterate over all balanced fine partitionings of slicings of I. However, even if we don't consider slicings, doing that will take exponential time, since for each big, wide and tall item, we need to decide whether to designate it as a division-1 item or a division-2 item.

We can get around this problem by iterating over a polynomial-sized set $\mathcal{S}_\Pi$ of fine partitionings such that each balanced fine partitioning $\mathcal{P}$ of a slicing of I is 'close to' a fine partitioning $\widehat{\mathcal{P}}$ in $\mathcal{S}_\Pi$. We will now define what we mean by 'close to'.

Definition 5.24 (Predecessor of a set of items)**.** *Let I_1 and I_2 be sets of items. Interpret each item $i \in I_1$ as a bin whose geometric dimensions are the same as that of i and whose weight capacities are the same as the weights of i. I_2 is said to be a predecessor of I_1 (denoted as $I_2 \preceq I_1$) iff I_2 can be sliced and packed into I_1.*

We will design a polynomial-time algorithm `iterFineParts` that takes as input a weight-rounded set I of items and outputs a set $\mathcal{S}_\Pi$ of pairs such that for each balanced fine partitioning $\mathcal{P}$ of a slicing of I, there exists a pair $(D, \widehat{\mathcal{P}}) \in \mathcal{S}_\Pi$ such that all of these conditions hold:

- $\widehat{\mathcal{P}}$ is a fine partitioning of $I - D$.
- After applying Transformation 5.21 to $\mathcal{P}$ and $\widehat{\mathcal{P}}$, each partition in $\widehat{\mathcal{P}}$ is a predecessor of the corresponding partition in $\mathcal{P}$.
- D is a set of non-dense items (called discarded items) such that $\text{span}(D)$ is small compared to $\text{span}(I)$.

5.4.1 Big Items

We will describe an algorithm, called `partBig`, that takes a coarse partition B_p of big items as input, and outputs multiple fine partitionings of B_p. We can use `partBig` as a subroutine in `iterFineParts`.

To design `partBig`, we will guess the cardinality of sets $B^w_{p,q,r}$ and $B^h_{p,q,r}$ for each q and r. We will then guess the maximum height in $B^w_{p,q,r}$ and the maximum width in $B^h_{p,q,r}$. Then for each guess, we will solve a max-flow problem to check if items in B_p can be assigned to these sets. The details can be inferred from Section 3.3.1 of Prädel's thesis [66], but for the sake of completeness, we give the full details in Sections 5.4.1.1, 5.4.1.2 and 5.4.1.3.

Formally `partBig`(B_p) outputs a set of pairs of the form $(\{\}, \widehat{\mathcal{P}})$, where $\widehat{\mathcal{P}}$ is supposed to be a fine partitioning of B_p. In Section 5.4.1.2, we prove the following important results about `partBig`.

Claim 5.30. `partBig`(B_p) *generates* $O(n^{2|Q|(1+1/\delta_{\lg})}) \leq O(n^{8(d+1)/\varepsilon\varepsilon_1^3})$ *values, where* $n := |B_p|$, *and the running time per value is* $O(n^2/\varepsilon\varepsilon_1)$.

Lemma 5.31. *Let* $\mathcal{P} := \{B^w_{p,q,r} : \forall q, \forall r\} \cup \{B^h_{p,q,r} : \forall q, \forall r\}$ *be a balanced fine partitioning of* B_p. *Then there is an output* $(\{\}, \widehat{\mathcal{P}})$ *of* `partBig`(B_p) *where* $\widehat{\mathcal{P}} := \{\widehat{B}^w_{p,q,r} : \forall q, \forall r\} \cup \{\widehat{B}^h_{p,q,r} : \forall q, \forall r\}$ *such that* $\widehat{\mathcal{P}}$ *is a fine partitioning of* B_p *and after applying Transformation 5.21,*

$$\forall q, \forall r, \widehat{B}^w_{p,q,r} \preceq B^w_{p,q,r} \text{ and } \widehat{B}^h_{p,q,r} \preceq B^h_{p,q,r}.$$

`partBig` for the rotational case is similar to the non-rotational case: When rotations are allowed, assume without loss of generality that $B^h_{*,*,*} = \{\}$. We will guess the cardinality and maximum height in sets $B^w_{p,q,r}$. Then for each guess, we will solve a max-flow problem to check if items in B_p can be assigned to these sets, possibly after rotating some items.

5.4.1.1 A Constrained Partitioning Problem

We will now describe a constrained partitioning problem and its solution using a max-flow algorithm. We will later show how to use an algorithm for this constrained partitioning problem to implement `partBig`.

In this problem, we are given as input a set I of items and a set J of targets. For simplicity and without loss of generality, assume $I = [n]$ and $J = [m]$. For the j^{th} target, we are given a 'desired cardinality' b_j, such that $\sum_{j=1}^{m} b_j = n$. We are also given a set $E \subseteq I \times J$. An item i is said to be feasible for a target j iff $(i, j) \in E$.

We have to either assign each item to exactly one feasible target such that the number of items received by the j^{th} target is b_j, or we have to declare that such an assignment is not possible. An instance of the constrained partitioning problem is given by the tuple (I, J, b, E). Here b is a vector of length m.

We will create a flow network $G' = (V', E')$ for this problem. Create a source vertex s and a sink vertex t. Create a vertex u_i for each item i. Create a vertex v_j for the j^{th} target. Add an edge of capacity 1 from s to each u_i. Add an edge of capacity b_j from each v_j to t. For each $(i, j) \in E$, add an edge of capacity 1 from u_i to v_j. Therefore, we get

$$E' = \{(s, u_i, 1) : \forall i \in I\} \cup \{(u_i, v_j, 1) : \forall (i, j) \in E\} \cup \{(v_j, t, b_j) : \forall j \in J\}.$$

See Fig. 5.5 for an example.

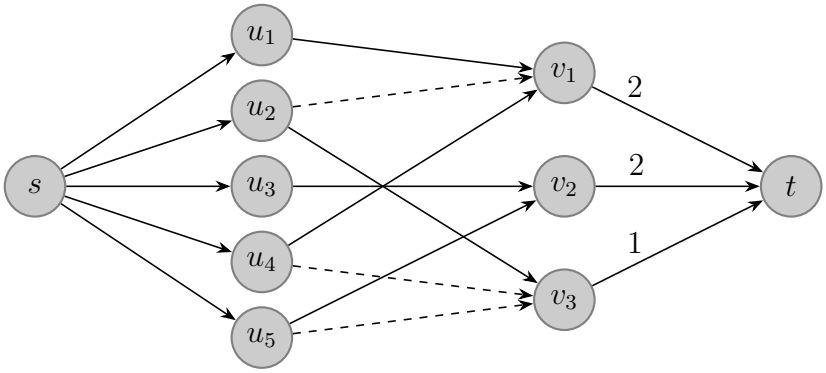


Figure 5.5: Maximum flow for a network with $n = 5$, $m = 3$ and $b = [2, 2, 1]$. Edges are labeled with their capacities and unlabeled edges have capacity 1. Dashed edges have flow 0. Solid edges have flow equal to capacity.

Lemma 5.32. *If we know an integral flow f of value n for the network G', we can use f to get a feasible solution to the constrained partitioning problem.*

Proof. In such a flow, $f(s, u_i) = 1$ for all i and $f(v_j, t) = b_j$ for all j. By the conservation constraint at u_i, there will be exactly one j for each i such that $f(u_i, v_j) = 1$. Assign item i to this j. This gives us a feasible assignment of items to targets. □

Lemma 5.33. *The flow network G' has maximum flow equal to n iff there exists a feasible assignment of items to targets.*

Proof. The cut $(\{s\}, V' - \{s\})$ has value n. By the max-flow-min-cut theorem [25, Theorem 26.6], the max-flow is at most n.

If there is a feasible assignment $\pi : I \mapsto J$, then we can set the flow $f : E' \mapsto \mathbb{R}_{\geq 0}$ as $f(s, u_i) = 1$, $f(u_i, v_{\pi(i)}) = 1$, $f(v_j, t) = b_j$ and flow 0 for the remaining edges. This is a feasible flow of value n, so it must be a max-flow.

Suppose there is a max-flow of value n. Since all capacities are integers, by the max-flow integrality theorem [25, Theorem 26.10], there is an integral max-flow f of value n. Then by Lemma 5.32, there exists a feasible assignment. □

Lemmas 5.32 and 5.33 give us an algorithm for the constrained partitioning problem, which we call `constrPartSolve` (see Algorithm 2).

Algorithm 2 `constrPartSolve`(I, J, b, E)

$V' = \{s, t\} \cup \{u_i : i \in I\} \cup \{v_j : j \in J\}$.
$E' = \{(s, u_i, 1) : \forall i \in I\} \cup \{(u_i, v_j, 1) : \forall (i, j) \in E\} \cup \{(v_j, t, b_j) : \forall j \in J\}$
Get an integral max-flow f for $G' = (V', E')$ using the Ford-Fulkerson algorithm [25].
if f has flow-value $|I|$ **then**
    Let $\pi(i)$ be the unique value of j such that $f(u_i, v_j) = 1$.
    return π
else
    return `null`
end if

Lemma 5.34. `constrPartSolve`$([n], [m], b, E)$ *runs in time* $O(n(n + m + |E|))$.

Proof. The flow network has $n + m + 2$ vertices and $n + m + |E|$ edges. The Ford-Fulkerson algorithm takes time $O(n(n + m + |E|))$. □

5.4.1.2 `partBig` Without Item Rotations

We will now describe `partBig` for the case where item rotations are forbidden. Let B_p be a coarse partition of big items in I. We will guess the cardinality of sets $B^w_{p,q,r}$ and $B^h_{p,q,r}$ for each q and r. We will then guess the maximum height in $B^w_{p,q,r}$ and the maximum width in $B^h_{p,q,r}$ for each q and r. Then for each guess, we will check if items in B_p can be assigned to these sets. Let

$$Q := \mathbb{Z} \cap \left[\frac{4}{\varepsilon_1} + 1, \frac{4}{\varepsilon_1^2}\right] \qquad Q_q := \left\{x \in \mathbb{R} : (q-1)\frac{\varepsilon_1^2}{4} < x \leq q\frac{\varepsilon_1^2}{4}\right\}$$

There are $|Q|$ sets $B^w_{p,q,*}$ and there are $|Q|$ sets $B^h_{p,q,*}$. We will guess the cardinality of all of them. The number of guesses is at most $1 + n^{2|Q|}$. Since the fine partitioning is balanced,

we can find $|B^w_{p,q,r}|$ from $|B^w_{p,q,*}|$ and we can find $|B^h_{p,q,r}|$ from $|B^h_{p,q,*}|$. Let $b^w_{p,q,r}$ and $b^h_{p,q,r}$ be the cardinalities of $B^w_{p,q,r}$ and $B^h_{p,q,r}$, respectively. Let b be the vector containing all $b^*_{p,*,*}$ values.

Next we will guess the maximum height in $B^w_{p,q,r}$ and the maximum width in $B^h_{p,q,r}$. There are a total of $2|Q|/\delta_{\lg} = 8(d+1)(1-\varepsilon_1)/\varepsilon\varepsilon_1^3$ sets, and there are n items in B_p. Therefore, the number of guesses would be at most $n^{8(d+1)/\varepsilon\varepsilon_1^3}$. Let $h^w_{p,q,r}$ be the guess of the maximum height in $B^w_{p,q,r}$ and $w^h_{p,q,r}$ be the guess of the maximum width in $B^h_{p,q,r}$.

After these guesses, we say that item i is feasible for $B^w_{p,q,r}$ iff $w(i) \in Q_q$ and $h(i) \le h^w_{p,q,r}$, and item i is feasible for $B^h_{p,q,r}$ iff $h(i) \in Q_q$ and $w(i) \le w^h_{p,q,r}$. Let E be the pairs of feasible assignments of items to these sets. An item can be feasible for at most $1/\delta_{\lg}$ sets of the form $B^w_{p,q,r}$ and at most $1/\delta_{\lg}$ sets of the form $B^h_{p,q,r}$ (because q is determined by width or height). Therefore, $|E| \le 2n/\delta_{\lg}$.

We can find a feasible assignment of the items B_p to these sets (if one exists) using algorithm

$$\texttt{constrPartSolve}\left(B_p, \left\lceil \frac{8(d+1)(1-\varepsilon_1)}{\varepsilon\varepsilon_1^3} \right\rceil, b, E\right).$$

The time taken to run `constrPartSolve` is $O(n^2/\varepsilon\varepsilon_1)$. We summarize `partBig` in Algorithm 3.

Algorithm 3 `partBig`(B_p):

```
outputs = {}
for each guess b (the cardinalities of the sets) do
    for each guess h^w_{p,q,r} and w^h_{p,q,r} for each q ∈ Q and r ∈ [1/δ_lg] do
        Compute E, the set of pairs of feasible assignments.
        π = constrPartSolve(B_p, ⌈8(d+1)(1−ε_1)/εε_1^3⌉, b, E)
        if π ≠ null then
            Let P̂ be the fine partitioning deduced from π.
            outputs.add(({}, P̂))
        end if
    end for
end for
return outputs
```

Claim 5.30. `partBig`(B_p) *generates* $O(n^{2|Q|(1+1/\delta_{\lg})}) \le O(n^{8(d+1)/\varepsilon\varepsilon_1^3})$ *values, where* $n := |B_p|$, *and the running time per value is* $O(n^2/\varepsilon\varepsilon_1)$.

Lemma 5.31. *Let* $\mathcal{P} := \{B^w_{p,q,r} : \forall q, \forall r\} \cup \{B^h_{p,q,r} : \forall q, \forall r\}$ *be a balanced fine partitioning of* B_p. *Then there is an output* $(\{\}, \widehat{\mathcal{P}})$ *of* `partBig`(B_p) *where* $\widehat{\mathcal{P}} := \{\widehat{B}^w_{p,q,r} : \forall q, \forall r\} \cup \{\widehat{B}^h_{p,q,r} : \forall q, \forall r\}$

such that $\widehat{\mathcal{P}}$ *is a fine partitioning of* B_p *and after applying Transformation 5.21,*

$$\forall q, \forall r, \widehat{B}^w_{p,q,r} \preceq B^w_{p,q,r} \text{ and } \widehat{B}^h_{p,q,r} \preceq B^h_{p,q,r}.$$

Proof. In each iteration, `constrPartSolve` ensures that each item is assigned to exactly one set, so $\widehat{\mathcal{P}}$ is a fine partitioning of B_p. In some iteration, the guesses b, $h^w_{p,q,r}$ and $w^h_{p,q,r}$ will be correct. In that iteration, a feasible assignment of items to the sets exists. Therefore, the output π of `constrPartSolve` will not be `null`.

All items in B^p have the same weights. After applying Transformation 5.21, all items in $B^w_{p,q,r}$ have width $q\varepsilon_1^2/4$ and height $h^w_{p,q,r}$ and all items in $\widehat{B}^w_{p,q,r}$ have width $q\varepsilon_1^2/4$ and height at most $h^w_{p,q,r}$. Therefore, $\widehat{B}^w_{p,q,r} \preceq B^w_{p,q,r}$. Similarly, $\widehat{B}^h_{p,q,r} \preceq B^h_{p,q,r}$. □

5.4.1.3 `partBig` With Item Rotations

We will now describe `partBig` for the case where item rotations are allowed. Let B_p be a coarse partition of big items in I. When rotations are allowed, assume without loss of generality that $B^h_{*,*,*} = \{\}$. We will guess the cardinality and maximum height in sets $B^w_{p,q,r}$. Then for each guess, we will check if items in B_p can be assigned to these sets, possibly after rotating some items.

The number of guesses of $|B^w_{p,q,*}|$ is at most $1 + n^{|Q|}$ Since the fine partitioning is balanced, we can find $|B^w_{p,q,r}|$ from $|B^w_{p,q,*}|$. Let $b^w_{p,q,r}$ be the guess of $|B^w_{p,q,r}|$. Let $h^w_{p,q,r}$ be the guess of the maximum height in $B^w_{p,q,r}$. There are at most $(2n)^{|Q|/\delta_{\lg}} = (2n)^{4(d+1)/\varepsilon\varepsilon_1^3}$ possible guesses for maximum heights.

After these guesses, we say that item i is feasible for $B^w_{p,q,r}$ without rotation iff $w(i) \in Q_q$ and $h(i) \leq h^w_{p,q,r}$ and i is feasible for $B^w_{p,q,r}$ after rotation iff $h(i) \in Q_q$ and $w(i) \leq h^w_{p,q,r}$. Let E_1 be the pairs of feasible assignments of items to these sets without rotation and E_2 be the pairs of feasible assignments of items to these sets after rotation. An item can be feasible for at most $1/\delta_{\lg}$ sets without rotation and feasible for at most $1/\delta_{\lg}$ sets after rotation. Therefore, $|E_1 \cup E_2| \leq |E_1| + |E_2| \leq 2n/\delta_{\lg}$.

We can find a feasible assignment of the items B_p to these sets (if one exists) using algorithm

$$\texttt{constrPartSolve}\left(B_p, \left\lceil \frac{4(d+1)(1-\varepsilon_1)}{\varepsilon\varepsilon_1^3} \right\rceil, b, E_1 \cup E_2\right).$$

The time taken to run `constrPartSolve` is $O(n^2/\varepsilon\varepsilon_1)$. We summarize `partBig` in Algorithm 4.

Claim 5.35. `partBig`(B_p) *generates* $O(n^{|Q|(1+1/\delta_{\lg})}) = O(n^{4(d+1)/\varepsilon\varepsilon_1^3})$ *values, where* $n := |B_p|$*, and the running time per value is* $O(n^2/\varepsilon\varepsilon_1)$.

Algorithm 4 `partBig`(B_p) (with rotations):

`outputs` $= \{\}$
for each guess b (the cardinalities of the sets) **do**
  for each guess $h^w_{p,q,r}$ for each $q \in Q$ and $r \in [1/\delta_{\lg}]$ **do**
    Compute E_1, the set of pairs of feasible assignments without rotations.
    Compute E_2, the set of pairs of feasible assignments after rotations.
    $\pi =$ `constrPartSolve` $\left(B_p, \left\lceil \frac{4(d+1)(1-\varepsilon_1)}{\varepsilon\varepsilon_1^3} \right\rceil, b, E_1 \cup E_2\right)$
    if $\pi \neq$ `null` **then**
      Let $\widehat{P}$ be the fine partitioning deduced from π.
      $\forall i \in B_p$, rotate item i in $\widehat{P}$ iff $(i, \pi(i)) \notin E_1$.
      `outputs.add`$((\{\}, \widehat{P}))$
    end if
  end for
end for
return `outputs`

Lemma 5.36. *Let $\mathcal{P} := \{B^w_{p,q,r} : \forall q, \forall r\}$ be a balanced fine partitioning of B_p. Then there is an output $(\{\}, \widehat{\mathcal{P}})$ of* `partBig`(B_p) *where $\widehat{\mathcal{P}} := \{\widehat{B}^w_{p,q,r} : \forall q, \forall r\}$ such that $\widehat{\mathcal{P}}$ is a fine partitioning of B_p and after applying Transformation 5.21,*

$$\forall q, \forall r, \widehat{B}^w_{p,q,r} \preceq B^w_{p,q,r}.$$

Proof. In each iteration, `constrPartSolve` ensures that each item is assigned to exactly one set. Therefore, $\widehat{\mathcal{P}}$ is a fine partitioning of B_p.

In some iteration, the guesses b and $h^w_{p,q,r}$ will be correct. In that iteration, a feasible assignment of items to the sets exists. Therefore, the output $\widehat{\mathcal{P}}$ of `constrPartSolve` will not be `null`.

All items in B^p have the same weights. After applying Transformation 5.21, all items in $B^w_{p,q,r}$ have width $q\varepsilon_1^2/4$ and height $h^w_{p,q,r}$ and all items in $\widehat{B}^w_{p,q,r}$ have width $q\varepsilon_1^2/4$ and height at most $h^w_{p,q,r}$. Therefore, $\widehat{B}^w_{p,q,r} \preceq B^w_{p,q,r}$. □

5.4.2 Wide and Tall Items

We will describe an algorithm, called `partWide`, that takes a coarse partition W_p of wide items as input, and outputs multiple fine partitionings of W_p. We can use `partWide` as a subroutine in `iterFineParts`.

Let $\mathcal{P} := \{W^w_{p,q} : \forall q\} \cup \{W^h_{p,r} : \forall r\}$ be a balanced fine partitioning of a slicing of W_p. We

want `partWide` to find a fine partitioning $\widehat{\mathcal{P}} := \{\widehat{W}^w_{p,q} : \forall q\} \cup \{\widehat{W}^h_{p,r} : \forall r\}$ of a large subset of W_p such that after applying Transformation 5.21 to $\mathcal{P}$ and $\widehat{\mathcal{P}}$, every fine partition in $\widehat{\mathcal{P}}$ is a predecessor of the corresponding fine partition in $\mathcal{P}$.

For any $J \subseteq W_p$, define $h(J) := \sum_{i \in J} h(i)$ and $w(J) := \max_{i \in J} w(i)$. We will create a rectangular box for each fine partition and then try to pack a large subset of W_p into these boxes. Let $Q := \mathbb{Z} \cap \left[\frac{4}{\varepsilon_1} + 1, \frac{4}{\varepsilon_1^2}\right]$. For each $q \in Q$, let s^w_q be a box of width $q\varepsilon_1^2/4$ and height $h(W^w_{p,q})$. For each $r \in [1/\delta_{\lg}]$, let s^h_r be a box of width $w(W^h_{p,r})$ and height $h(W^h_{p,r})$. Since we don't know $W^h_{p,r}$ and $W^w_{p,q}$, we will guess the value of $w(W^h_{p,r})$ and we will guess very close lower bounds on $h(W^w_{p,q})$ and $h(W^h_{p,r})$. We will then try to pack most of the items from W_p into these boxes.

Let $\widehat{W}^w_{p,q}$ be the items packed into s^w_q and let $\widehat{W}^h_{p,r}$ be the items packed into s^h_r. Then $\widehat{\mathcal{P}} := \{\widehat{W}^w_{p,q} : \forall q\} \cup \{\widehat{W}^h_{p,r} : \forall r\}$ is a fine partitioning of a large subset of W_p. After applying Transformation 5.21 to $\mathcal{P}$ and $\widehat{\mathcal{P}}$, each item in $W^w_{p,q}$ and $\widehat{W}^w_{p,q}$ has width $q\varepsilon_1^2/4$. Since $h(\widehat{W}^w_{p,q}) \leq h(s^w_q) \leq h(W^w_{p,q})$, we get $\widehat{W}^w_{p,q} \preceq W^w_{p,q}$ after Transformation 5.21. We can similarly prove that $\widehat{W}^h_{p,r} \preceq W^h_{p,r}$. Therefore, $\widehat{\mathcal{P}}$ is a suitable fine partitioning.

The details on how to approximately guess the size of boxes and how to pack a large subset of items into boxes can be deduced from Section 3.3.1 of Prädel's thesis [66]. However, for the sake of completeness, we give the details in Sections 5.4.2.1 and 5.4.2.2.

Formally, `partWide`(W_p) outputs a set of pairs of the form $(D, \widehat{\mathcal{P}})$, where items in D are called *discarded* items and $\widehat{\mathcal{P}}$ is supposed to be a fine partitioning of $W_p - D$. In Section 5.4.2.2, we prove the following important results about `partWide`.

Lemma 5.37. *For every output $(D, \widehat{\mathcal{P}})$ of* `partWide`(W_p)*,*

$$h(D) \leq (3\varepsilon_2)\left(\frac{d+1}{\varepsilon\varepsilon_1} + \frac{4}{\varepsilon_1^2} - \frac{4}{\varepsilon_1}\right).$$

Lemma 5.38. *Let $\mathcal{P} := \{W^w_{p,q} : \forall q\} \cup \{W^h_{p,r} : \forall r\}$ be a balanced fine partitioning of a slicing of W_p. Then for some output $(D, \widehat{\mathcal{P}})$ of* `partWide`(W_p)*, $\widehat{\mathcal{P}}$ is a fine partitioning of $W_p - D$ and after applying Transformation 5.21 to $\mathcal{P}$ and $\widehat{\mathcal{P}}$, we get $(\forall q, \widehat{W}^w_{p,q} \preceq W^w_{p,q})$ and $(\forall r, \widehat{W}^h_{p,r} \preceq W^h_{p,r})$.*

Claim 5.39. *Let there be n items in W_p. Let $n_q := 4/\varepsilon_1^2 - 4/\varepsilon_1$. Then* `partWide`$(W_p)$ *outputs at most $\delta_{\lg} n^{n_q+1+1/\delta_{\lg}}$ distinct values. The running time per value is $O(n \log n)$.*

`partWide` can analogously be used for sub-partitioning coarse partitions of tall items. When item rotations are allowed, `partWide`(W_p) gives us $H^w_{p,r}$ instead of $W^h_{p,r}$.

5.4.2.1 Rectangle Stacking Problem

We will describe a problem, called the rectangle stacking problem, and look at an efficient algorithm for this problem, called `greedyStack`. We will later see how to use `greedyStack` to implement `partWide`.

In the rectangle stacking problem, we are given a set S of rectangular boxes and a set I of rectangles. For any rectangle or box i, let $w(i)$ and $h(i)$ be the width and height of i, respectively. For any set T of rectangles, $w(T)$ is defined as $\max_{i \in T} w(i)$, and $h(T)$ is defined as $\sum_{i \in T} h(i)$.

Our aim is to pack a large subset of rectangles from I into the boxes S. In each box s, the rectangles should be stacked, i.e., placed one-over-the-other. This means that if T is the set of rectangles assigned to box s, then $h(T) \leq h(s)$ and $w(T) \leq w(s)$.

We will analyze a simple algorithm for this problem, called `greedyStack`. `greedyStack` first sorts the rectangles and boxes in decreasing order of height. Starting from the first box, it repeatedly packs rectangles into the box till the box overflows, i.e., the rectangles packed inside the box have height more than the height of the box. It then discards the overflowing rectangle and resumes from the next box. If `greedyStack` ever comes across a rectangle whose width is more than the width of the current box, it returns `fail`. If `greedyStack` uses up all boxes but some rectangles haven't yet been packed or discarded, it returns `fail`. Otherwise it returns the tuple $(D, T_1, T_2, \ldots, T_{|S|})$, where D is the set of discarded rectangles and T_j is the set of rectangles packed into box j. See Algorithm 5 for a more precise description of `greedyStack`. See Fig. 5.6 for an example output of `greedyStack`.

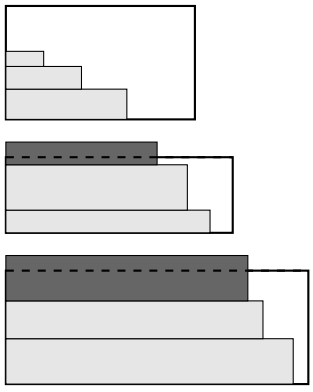


Figure 5.6: Example output of `greedyStack` for 9 rectangles and 3 boxes. The dark rectangles are discarded.

Lemma 5.40. *When* `greedyStack` *doesn't return* `fail`*, it stacks all rectangles in* $I - D$ *into boxes* S *and* $|D| \leq |S|$.

Algorithm 5 `greedyStack`(I, S): Horizontally stack a large subset of rectangles from I into rectangular boxes S. Returns a set $D \subseteq I$ of discarded rectangles and a partitioning $T_1, T_2, \ldots, T_{|S|}$ of $I - D$ such that the rectangles in T_j can be stacked into the j^{th} box.

1: Sort the rectangles I in non-increasing order of width.
2: Sort the boxes S in non-increasing order of width. Let s_k be the k^{th} box.
3: $j = 1$
4: $D = \{\}$ *// set of discarded rectangles*
5: $T_k = \{\}$ for $k \in [|S|]$. *// set of rectangles in s_k*
6: **for** $i \in I$ **do**
7: **if** $j > |S|$ **then**
8: **return** `fail`
9: **else if** $w(i) > w(s_j)$ **then**
10: **return** `fail`
11: **else if** $h(i) > h(s_j) - h(T_j)$ **then**
12: Add rectangle i to set D.
13: j `+=` 1
14: **else**
15: Add rectangle i to T_j. *// Stack rectangle i into box s_j.*
16: **end if**
17: **end for**
18: **return** $(D, T_1, T_2, \ldots, T_{|S|})$.

Proof. We discard a rectangle only when it overflows a box, and when a box overflows, we switch to a new box. Hence, there can be at most $|S|$ discarded rectangles. (In Algorithm 5, we add a rectangle to D whenever we increment j, and we only do that at most $|S|$ times. Therefore, $|D| \leq |S|$.)

If `greedyStack` didn't fail, then each rectangle either got packed or discarded. Therefore, all rectangles in $I - D$ are packed into boxes. (The conditional statements in Algorithm 5 ensure that when `greedyStack` tries to stack a rectangle in line 15 of Algorithm 5, it can actually do so. Since `greedyStack` doesn't return `fail`, every rectangle is either stacked into a box or is added to D.) □

Lemma 5.41. *If a set I of rectangles can be horizontally sliced and stacked into boxes S, then* `greedyStack`(I, S) *does not return* ***`fail`****.*

Proof. We will prove this by contradiction. Let $T := \bigcup_k T_k$. Let d_k be the k^{th} discarded rectangle (i.e., d_k is the rectangle that was added to D when the value of j was equal to k in Algorithm 5). Then $h(d_k) + h(T_k) > h(s_k)$.

Suppose `greedyStack`(I, S) fails because some rectangles were left in the end (i.e., at line

8 in Algorithm 5). Then

$$h(T) + h(D) = \sum_{k=1}^{|S|} (h(T_k) + h(d_k)) > \sum_{k=1}^{|S|} h(s_k) \geq h(I),$$

which is a contradiction.

Suppose $\texttt{greedyStack}(I, S)$ fails because some rectangle i was wider than box j, (i.e., at line 10 in Algorithm 5 because $w(i) > w(s_j)$). Then there cannot be any rectangle in the j^{th} box, because rectangles are sorted in non-increasing order of width and if a rectangle can fit in the j^{th} box, all subsequent rectangles can fit too.

Let $X := \{i' \in I : w(i') > w(i)\} \cup \{i\}$. In a fractional stacking of I into S, the rectangles in X are placed in the first $j-1$ bins. This is because $i' \in X \implies w(i') \geq w(i) > w(s_j)$. Therefore, $h(X) \leq \sum_{k=1}^{j-1} h(s_k)$. On the other hand,

$$h(X - \{i\}) = h(T) + h(D) = \sum_{k=1}^{j-1} (h(T_k) + h(d_k)) > \sum_{k=1}^{j-1} h(s_k).$$

This is a contradiction. Therefore, $\texttt{greedyStack}(I, S)$ cannot fail. □

Claim 5.42. *The running time of* $\texttt{greedyStack}(I, S)$ *is* $O(|I| \log |I| + |S| \log |S|)$.

5.4.2.2 Partitioning Wide Items

Let W_p be a coarse partition of wide items in I. Let $\mathcal{P} := \{W^w_{p,q} : \forall q\} \cup \{W^h_{p,r} : \forall r\}$ be a balanced fine partitioning of a slicing of W_p. Our aim is to find a fine partitioning $\widehat{\mathcal{P}} := \{\widehat{W}^w_{p,q} : \forall q\} \cup \{\widehat{W}^h_{p,r} : \forall r\}$ of a large subset of W_p such that after applying Transformation 5.21 to $\mathcal{P}$ and $\widehat{\mathcal{P}}$, every fine partition in $\widehat{\mathcal{P}}$ is a predecessor of the corresponding fine partition in $\mathcal{P}$.

For any $J \subseteq W_p$, define $h(J) := \sum_{i \in J} h(i)$ and $w(J) := \max_{i \in J} w(i)$. We will create a rectangular box for each fine partition and then try to pack a large subset of W_p into these boxes. Let $Q := \mathbb{Z} \cap \left[\frac{4}{\varepsilon_1} + 1, \frac{4}{\varepsilon_1^2}\right]$. For each $q \in Q$, let s^w_q be a box of width $q\varepsilon_1^2/4$ and height $h(W^w_{p,q})$. For each $r \in [1/\delta_{\lg}]$, let s^h_r be a box of width $w(W^h_{p,r})$ and height $h(W^h_{p,r})$. Since W_p can be sliced and packed into these boxes, we can use $\texttt{greedyStack}$ to pack $|W_p| - O(1)$ items into these boxes (see Lemmas 5.40 and 5.41). Let $\widehat{W}^w_{p,q}$ be the items packed into s^w_q and let $\widehat{W}^h_{p,r}$ be the items packed into s^h_r. Then $\widehat{\mathcal{P}} := \{\widehat{W}^w_{p,q} : \forall q\} \cup \{\widehat{W}^h_{p,r} : \forall r\}$ is a fine partitioning of a large subset of W_p.

After applying Transformation 5.21 to $\mathcal{P}$ and $\widehat{\mathcal{P}}$, each item in $W^w_{p,q}$ and $\widehat{W}^w_{p,q}$ has width $q\varepsilon_1^2/4$. Since $h(\widehat{W}^w_{p,q}) \leq h(s^w_q) = h(W^w_{p,q})$, we get $\widehat{W}^w_{p,q} \preceq W^w_{p,q}$ after Transformation 5.21. We

can similarly prove that $\widehat{W}^h_{p,r} \preceq W^h_{p,r}$. Therefore, $\widehat{\mathcal{P}}$ is a suitable fine partitioning.

Unfortunately, the above algorithm cannot be used, because we don't know the values of $h(W^w_{p,q})$, $h(W^h_{p,r})$ and $w(W^h_{p,r})$, so the boxes cannot be created. We can work around this issue by guessing these values. We will guess $w(W^h_{p,r})$ exactly and guess $h(W^w_{p,q})$ and $h(W^h_{p,q})$ approximately. Guessing a value x approximately means guessing the smallest multiple of ε_2 that is greater than or equal to x. Since our guess $h(s^w_q)$ could be up to ε_2 more than $h(W^w_{p,q})$ and our guess $h(s^h_r)$ could be up to ε_2 more than $h(W^h_{p,r})$, we will also discard items that intersect the top ε_2-sized section of each box. For each guess, we will get a fine partitioning $\widehat{\mathcal{P}}$. One of our guesses will be correct, i.e., for some value of $\widehat{\mathcal{P}}$, we will get $(\forall q, \widehat{W}^w_{p,q} \preceq W^w_{p,q})$ and $(\forall r, \widehat{W}^h_{p,r} \preceq W^h_{p,r})$. We call this algorithm `partWide`, which is described more precisely as Algorithm 6.

Lemma 5.37. *For every output $(D, \widehat{\mathcal{P}})$ of* `partWide`(W_p),

$$h(D) \le (3\varepsilon_2)\left(\frac{d+1}{\varepsilon\varepsilon_1} + \frac{4}{\varepsilon_1^2} - \frac{4}{\varepsilon_1}\right).$$

Proof. Let $D = D_1 \cup D_2$, where D_1 and D_2 are as defined in Algorithm 6.

$$|S^w \cup S^h| \le k + (q_{\max} - q_{\min} + 1) = \frac{d+1}{\varepsilon\varepsilon_1} + \frac{4}{\varepsilon_1^2} - \frac{4}{\varepsilon_1}.$$

$$h(D_1) \le \varepsilon_2 |D_1| \le \varepsilon_2 |S^w \cup S^h|. \qquad \text{(by Lemma 5.40)}$$

$$h(D_2) \le (2\varepsilon_2)|S^w \cup S^h|.$$

Therefore, $h(D_1 \cup D_2) \le (3\varepsilon_2)|S^w \cup S^h|$. □

Lemma 5.38. *Let $\mathcal{P} := \{W^w_{p,q} : \forall q\} \cup \{W^h_{p,r} : \forall r\}$ be a balanced fine partitioning of a slicing of W_p. Then for some output $(D, \widehat{\mathcal{P}})$ of* `partWide`(W_p), *$\widehat{\mathcal{P}}$ is a fine partitioning of $W_p - D$ and after applying Transformation 5.21 to $\mathcal{P}$ and $\widehat{\mathcal{P}}$, we get $(\forall q, \widehat{W}^w_{p,q} \preceq W^w_{p,q})$ and $(\forall r, \widehat{W}^h_{p,r} \preceq W^h_{p,r})$.*

Proof. All items, except the ones in D, are placed into some fine partition. Therefore, $\widehat{\mathcal{P}}$ is a fine partitioning of $W_p - D$.

The first loop guesses $w(W^h_{p,r})$ for each r. Some iteration of the loop will guess correctly. The second loop guesses an integer i_0 such that $h(W^h_{p,*}) \in [k\varepsilon_2(i_0 - 1), k\varepsilon_2 i_0]$ and for each $q \in [q_{\min}, q_{\max}]$ guesses i_q such that $h(W^w_{p,q}) \in [(i_q - 1)\varepsilon_2, i_q\varepsilon_2]$. Some iteration of the loop will guess correctly.

Since $\mathcal{P}$ is balanced, $h(W^h_{p,r}) = \delta_{\lg} h(W^h_{p,*})$. Therefore, for the correct guess of i_0, $h(W^h_{p,r}) \in [\varepsilon_2(i_0 - 1), \varepsilon_2 i_0]$.

Algorithm 6 `partWide`(W_p): Returns a set of pairs of the form $(D, \widehat{\mathcal{P}})$, where $\widehat{\mathcal{P}}$ is a fine-partitioning of $W_p - D$.

`outputs` $= \{\}$
$q_{\min} = 4/\varepsilon_1 + 1$, $q_{\max} = 4/\varepsilon_1^2$.
$k = 1/\delta_{\lg}$
for $[w_1, w_2, \ldots, w_k] \in \{w(i) : i \in W_p\}^k$ **do**
  for $[i_0, i_{q_{\min}}, i_{q_{\min}+1}, \ldots, i_{q_{\max}}] \in \left[\left\lceil \frac{h(W_p)}{k\varepsilon_2} \right\rceil\right] \times \left[\left\lceil \frac{h(W_p)}{\varepsilon_2} \right\rceil\right]^{q_{\max}-q_{\min}+1}$ **do**
    // Create boxes and pack items in them.
    For $q \in [q_{\min}, q_{\max}]$, let s_q^w be a box of width $q\varepsilon_1^2/4$ and height $\varepsilon_2 i_q$.
    Let $S^w = \{s_q^w : q \in [q_{\min}, q_{\max}]\}$.
    For $r \in [k]$, let s_r^h be a box of width w_r and height $\varepsilon_2 i_0$.
    Let $S^h = \{s_r^h : r \in [k]\}$.
    $(D_1, T^w_{q_{\min}}, \ldots, T^w_{q_{\max}}, T^h_1, \ldots, T^h_k) =$ `greedyStack`$(W_p, S^w \cup S^h)$
    if `greedyStack` failed **then**
      `continue`
    end if
    // Clear a strip of height ε_2 in each box.
    $D_2 = \{\}$
    for $q \in [q_{\min}, q_{\max}]$ **do**
      while $h(T_q^w) > h(s_q^w) - \varepsilon_2$ and $|T_q^w| > 0$ **do**
        Move an item from T_q^w to D_2.
      end while
    end for
    for $r \in [k]$ **do**
      while $h(T_r^h) > h(s_r^h) - \varepsilon_2$ and $|T_r^h| > 0$ **do**
        Move an item from T_r^h to D_2.
      end while
    end for
    For all $q \in [q_{\min}, q_{\max}]$, use T_q^w as the fine partition $\widehat{W}^w_{p,q}$.
    For all $r \in [k]$, use T_r^h as the fine partition $\widehat{W}^h_{p,r}$ (or $\widehat{H}^w_{p,r}$ if rotations are allowed).
    `outputs.add`$(D_1 \cup D_2, \{\widehat{W}^w_{p,q} : \forall q\} \cup \{\widehat{W}^h_{p,r} : \forall r\}))$
  end for
end for
return `outputs`

All items in $W^w_{p,q}$ can fit in s_q^w because

$$h(s_q^w) = i_q\varepsilon_2 \geq h(W^w_{p,q}) \text{ and } w(s_q^w) = q\frac{\varepsilon_1^2}{4} \geq \max_{i \in W^w_{p,q}} w(i).$$

All items in $W^h_{p,r}$ can fit in s^h_r because

$$h(s^h_r) = i_0\varepsilon_2 \geq h(W^h_{p,q}) \text{ and } w(s^h_r) = w_r = \max_{i \in W^h_{p,r}} w(i).$$

Therefore, a slicing of W_p can fit in the boxes $S^w \cup S^h$. So, for the correct guesses, Lemma 5.41 implies that $\texttt{greedyStack}(W_p, S^w \cup S^h)$ doesn't fail,

Since we cleared a strip of height ε_2 from each box, $\forall q, h(T^w_q) \leq (i_q - 1)\varepsilon_2 \leq h(W^w_{p,q})$. Wide items can be sliced horizontally, all items in W_p have the same weight class and all items in T^w_q and $W^w_{p,q}$ will have width $q\varepsilon_1^2/4$ after Transformation 5.21. Therefore, after the above transformations, $\widehat{W}^w_{p,q} \preceq W^w_{p,q}$.

Since we cleared a strip of height ε_2 from each box, $\forall r, h(T^h_r) \leq (i_0 - 1)\varepsilon_2 \leq h(W^h_{p,r})$. All items in T^h_r and $W^h_{p,r}$ will have width w_r after Transformation 5.21. Therefore, after the above transformations, $\widehat{W}^h_{p,r} \preceq W^h_{p,r}$. □

Claim 5.39. *Let there be n items in W_p. Let $n_q := 4/\varepsilon_1^2 - 4/\varepsilon_1$. Then $\texttt{partWide}(W_p)$ outputs at most $\delta_{\lg} n^{n_q+1+1/\delta_{\lg}}$ distinct values. The running time per value is $O(n \log n)$.*

5.4.3 Rounding Algorithm

We define an algorithm, called `iterFineParts`, that takes as input a set I of weight-rounded items and returns a set of pairs of the form $(D, \widehat{\mathcal{P}})$, where $D \subseteq I$ is the set of items to discard and $\widehat{\mathcal{P}}$ is a fine partitioning of $I - D$.

`iterFineParts` works by first computing a coarse partitioning of the items. Then for each coarse partition B_p of big items, it calls $\texttt{partBig}(B_p)$, for each coarse partition W_p of wide items, it calls $\texttt{partWide}(W_p)$, and for each coarse partition H_p of tall items, it calls $\texttt{partWide}(H_p)$. It then iterates over all combinations of the outputs of `partBig` and `partWide` and outputs a pair $(D, \widehat{\mathcal{P}})$ for each combination. See Algorithm 7 for a more precise description.

We next use `iterFineParts` to design the algorithm `round`. `round` takes a set I of items. It removes medium items from I using `removeMedium`. It computes a weight-rounding (Transformation 5.17) of $I - I_{\text{med}}$ to get $\widehat{I}$. Then for each output $(D, \widehat{\mathcal{P}})$ of $\texttt{iterFineParts}(\widehat{I})$, it computes $\widetilde{I}$ by applying Transformation 5.21 to $\widehat{I} - D$ with respect to $\widehat{\mathcal{P}}$ and outputs $(\widetilde{I}, D \cup I_{\text{med}})$. See Algorithm 8 for a more precise description of `round`.

Assume that in an output $(\widetilde{I}, D)$ of $\texttt{round}(I, \varepsilon)$, the knowledge of the associated fine partitioning is implicitly present in $\widetilde{I}$.

Algorithm 7 `iterFineParts`(I): I is a set of weight-rounded items. Returns a set of pairs of the form $(D, \widehat{\mathcal{P}})$, where D is a subset of items to discard and $\widehat{\mathcal{P}}$ is a fine partitioning of $I - D$.

`outputs` $= \{\}$
$\{B_p : \forall p\} \cup \{W_p : \forall p\} \cup \{H_p : \forall p\} \cup$ (small and dense partitions) $=$ coarse-partition(I)
`iters` $= \prod_{p=1}^{n_{\text{bwc}}}$ `partBig`$(B_p) \times \prod_{p=1}^{n_{\text{wwc}}}$ `partWide`$(W_p) \times \prod_{p=1}^{n_{\text{wwc}}}$ `partWide`(H_p)
for $\mathcal{L} \in$ `iters` **do** *// $\mathcal{L}$ is a list of pairs*
    $\widehat{\mathcal{P}} =$ (small and dense partitions)
    $D = \{\}$
    for $(D_j, \widehat{\mathcal{P}}_j) \in \mathcal{L}$ **do**
        $D = D \cup D_j$
        Include partitions of $\widehat{\mathcal{P}}_j$ into $\widehat{\mathcal{P}}$.
    end for
    `outputs.add`$((D, \widehat{\mathcal{P}}))$
end for
return `outputs`

Lemma 5.43 (Polynomial-time)**.** *The total number of outputs of* `round`(I) *is* $O(n^\gamma)$*, where there are* n *items in* I *and*

$$\gamma := n_{\text{bwc}} \frac{8(d+1)}{\varepsilon\varepsilon_1^3} + 2n_{\text{wwc}} \left(\frac{4}{\varepsilon_1^2} + \frac{d+1}{\varepsilon\varepsilon_1} \right).$$

The time for each output is at most $O(n^2/\varepsilon\varepsilon_1)$*.*

Proof. Follows from Claims 5.30 and 5.39. □

Lemma 5.44 (Low discard)**.** *Let* $(\widetilde{I}, D')$ *be an output of* `round`(I, ε)*. Then*

$$\begin{aligned} \operatorname{span}(D') &\le \varepsilon \operatorname{span}(I) + \frac{6\varepsilon_2 n_{\text{wwc}}}{\varepsilon_1^2} \left(\frac{d+1}{\varepsilon\varepsilon_1} + \frac{4}{\varepsilon_1^2} - \frac{4}{\varepsilon_1} \right) \\ &\le \varepsilon \operatorname{span}(I) + 6(d+5) \frac{\varepsilon_2 n_{\text{wwc}}}{\varepsilon_1^4}. \end{aligned}$$

Proof. Let $D' = D \cup I_{\text{med}}$. By Lemma 5.1, $\operatorname{span}(I_{\text{med}}) \le \varepsilon \operatorname{span}(I)$. Let D_1 be the wide items in D and D_2 be the tall items in D. Since all items in D come from `partWide`, D only contains non-dense items and $D = D_1 \cup D_2$. Let

$$\lambda := n_{\text{wwc}}(3\varepsilon_2) \left(\frac{d+1}{\varepsilon\varepsilon_1} + \frac{4}{\varepsilon_1^2} - \frac{4}{\varepsilon_1} \right).$$

Algorithm 8 `round`(I, ε): Returns a set of pairs of the form $(\widetilde{I}, D')$, where D' is a subset of items to discard and $\widetilde{I}$ is a rounding of $I - D'$.

`outputs` $= \{\}$
$\delta_0 := \min\left(\frac{1}{4d+1}, \frac{2\varepsilon}{3}\right)$
$(I_{\text{med}}, \varepsilon_2, \varepsilon_1) =$ `removeMedium`$(I, \varepsilon, f, \delta_0)$ *// f will be described later*
// Assume ε and ε_1 are passed as parameters to all subroutines and transformations.
Let $\widehat{I}$ be the weight-rounding (Transformation 5.17) of $I - I_{\text{med}}$.
for $(D, \widehat{\mathcal{P}}) \in$ `iterFineParts`$(\widehat{I})$ **do**
    Let $\widetilde{I}$ be the instance obtained by applying Transformation 5.21 to $\widehat{I} - D$ based on the fine partitioning $\widehat{\mathcal{P}}$.
    `outputs.add`$((\widetilde{I}, D \cup I_{\text{med}}))$.
end for
return `outputs`

By Lemma 5.37, $h(D_1) \le \lambda$ and $w(D_2) \le \lambda$.

$$\text{span}(D_1) = \sum_{i \in D_1} \max(a(i), v_{\max}(i)) \le \sum_{i \in D_1} \max\left(h(i), \frac{h(i)}{\varepsilon_1^2}\right) \le \sum_{i \in D_1} \frac{h(i)}{\varepsilon_1^2} \le \frac{\lambda}{\varepsilon_1^2}.$$

Similarly, $\text{span}(D_2) \le \lambda/\varepsilon_1^2$. □

Lemma 5.45 (Homogeneity)**.** *Let $(\widetilde{I}, D)$ be an output of* `round`(I, ε)*. Then the number of types of items in $\widetilde{I}$ is at most a constant:*

$$\frac{8(d+1)n_{\text{bwc}}}{\varepsilon\varepsilon_1^3} + 2n_{\text{wwc}}\left(\frac{d+1}{\varepsilon\varepsilon_1} + \frac{4}{\varepsilon_1^2}\right) + n_{\text{swc}} + 2(n_{\text{lwc}} + n_{\text{hwc}}).$$

Proof. After applying Transformation 5.21 to $\widehat{I} - D$, all non-dense items in each fine partition have:

- the same weight class.
- the same width, if the fine partition contains only wide or big items.
- the same height, if the fine partition contains only tall or big items.

From the definition of fine partitioning, the number of different partitions of non-dense items is at most

$$\frac{8n_{\text{bwc}}}{\varepsilon_1^2\delta_{\text{lg}}} + 2n_{\text{wwc}}\left(\frac{1}{\delta_{\text{lg}}} + \frac{4}{\varepsilon_1^2}\right) + n_{\text{swc}}.$$

In each division, there are n_{hwc} distinct heavy dense items and n_{lwc} weight classes in light dense items by Lemmas 5.22 and 5.24. □

Table 5.2: Upper bound on the number of different types of items

	no. of types
Division-1 big items ($B^w_{*,*,*}$)	$4(d+1)n_{\text{bwc}}/\varepsilon\varepsilon_1^3$
Division-2 big items ($B^h_{*,*,*}$)	$4(d+1)n_{\text{bwc}}/\varepsilon\varepsilon_1^3$
Division-1 wide non-dense items ($W^w_{*,*}$)	$4n_{\text{wwc}}/\varepsilon_1^2$
Division-2 wide non-dense items ($W^h_{*,*}$)	$(d+1)n_{\text{wwc}}/\varepsilon\varepsilon_1$
Division-1 tall non-dense items ($H^w_{*,*}$)	$(d+1)n_{\text{wwc}}/\varepsilon\varepsilon_1$
Division-2 tall non-dense items ($H^h_{*,*}$)	$4n_{\text{wwc}}/\varepsilon_1^2$
Small non-dense items (S_*)	n_{swc}
Division-1 heavy dense items ($D^{h,1}$)	n_{hwc}
Division-2 heavy dense items ($D^{h,2}$)	n_{hwc}
Division-1 light dense items ($D^{l,1}$)	n_{lwc}
Division-2 light dense items ($D^{l,2}$)	n_{lwc}

Lemma 5.46. *Let $\mathcal{P}$ be a balanced fine partitioning of a slicing of I. Then there is some output $(D, \widehat{\mathcal{P}})$ of* `iterFineParts`(I) *(Algorithm 7) such that $\widehat{\mathcal{P}}$ is a fine partitioning of $I - D$ and after Transformation 5.21, each partition in $\widehat{\mathcal{P}}$ is a predecessor of the corresponding partition in $\mathcal{P}$.*

Proof. Follows from Lemmas 5.31 and 5.38 □

Theorem 5.47. *There is an output $(\widetilde{I}, D \cup I_{\text{med}})$ of* `round`(I, ε) *such that $\widetilde{I}$ can be fractionally packed into at most $\left(1 + \frac{2\varepsilon}{1-\varepsilon}\right)(a\,\text{opt}(I) + b) + 2$ semi-structured $(5\varepsilon/8)$-slacked bins. Here values of a and b are as per Table 5.1.*

Proof. Let $m := \left(1 + \frac{2\varepsilon}{1-\varepsilon}\right)(a\,\text{opt}(I) + b) + 2$. Lemma 5.29 guarantees the existence of a balanced fine partitioning $\mathcal{P}$ of a slicing of $\widehat{I}$ such that after applying Transformation 5.21, there is a $(5\varepsilon/8)$-slacked fractional packing of $\widehat{I}$ into m bins that is semi-structured relative to $\mathcal{P}$.

By Lemma 5.46, we get that there is a fine partitioning $\widehat{\mathcal{P}}$ of $\widehat{I} - D$ such that after applying Transformation 5.21, each partition of $\widehat{\mathcal{P}}$ is a predecessor of the corresponding partition of $\mathcal{P}$. Therefore, we can pack each item in $\widetilde{I}$ into the place of some items in the packing of $\widehat{I} - D$. Therefore, there is a $(5\varepsilon/8)$-slacked fractional packing of $\widetilde{I}$ into at most m bins that is semi-structured relative to $\widehat{\mathcal{P}}$. □

5.5 Existence of Compartmental Packing

Definition 5.25 (Compartmental packing)**.** *Consider a semi-structured packing of items I into m bins, of which m_1 bins are division-1 bins and $m - m_1$ are division-2 bins.*

A compartment *is defined to be a rectangular region in a bin such that every item either lies completely inside the region or completely outside the region. Furthermore, a compartment doesn't contain big items, and a compartment doesn't contain both wide and tall items.*

In a division-1 bin, a compartment is called a dense compartment *iff it is the region $S^{(R')}$ and it contains a dense item (recall that $S^{(R')} := [1 - \varepsilon_1/2, 1] \times [0, 1]$). In a division-1 bin, a compartment is called a* sparse compartment *iff it satisfies all of these properties:*

- *The compartment doesn't contain dense items.*
- *The compartment contains at least 1 wide item or 1 tall item. If it contains wide items, it is called a wide compartment, and if it contains tall items, it is called a tall compartment.*
- *The x-coordinate of the left edge of the compartment is a multiple of $\varepsilon_1^2/4$.*
- *The compartment's width is a multiple of $\varepsilon_1^2/4$, and if the compartment is tall, its width is exactly $\varepsilon_1^2/4$.*
- *The compartment's height is rounded, i.e., if the compartment is wide, its height is a multiple of a constant $\varepsilon_{\text{cont}} := \varepsilon\varepsilon_1^5/12$ (note that $\varepsilon_{\text{cont}}^{-1} \in \mathbb{Z}$), and if the compartment is tall, its height is a sum of the heights of at most $1/\varepsilon_1 - 1$ of the items inside the compartment.*

A division-1 bin is said to be compartmental *iff we can create non-overlapping dense and sparse compartments in the bin such that all wide items, tall items and dense items are packed into compartments.*

We can analogously define compartmental packing for division-2 bins by swapping the coordinate axes. A semi-structured bin packing of items is called compartmental if each bin in the packing is compartmental.

Lemma 5.48. *Let there be a rectangular bin $B := [0, 1]^2$. Let there be a set I of rectangles packed inside the bin. Then there is a polynomial-time algorithm which can decompose the empty space in the bin $(B - I)$ into at most $3|I| + 1$ rectangles by making horizontal cuts only.*

Proof. Extend the top and bottom edges of each rectangle leftwards and rightwards till they hit another rectangle or the bin boundary. This partitions the empty space into rectangles R. See Fig. 5.7 for an example.

For each rectangle $i \in I$, the top edge of i is the bottom edge of a rectangle in R, the bottom edge of i is the bottom edge of two rectangles in R. Apart from possibly the rectangle in R

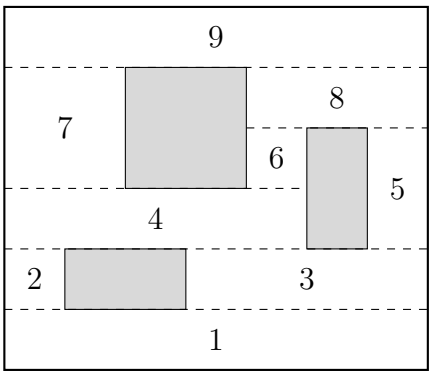


Figure 5.7: Using horizontal cuts to partition the empty space around the 3 rectangles into 9 rectangular regions.

whose bottom edge is at the bottom of the bin, the bottom edge of every rectangle in R is either the bottom or top edge of a rectangle in I. Therefore, $|R| \leq 3|I| + 1$. □

Lemma 5.49. *If there exists a semi-structured μ-slacked packing of items I into m bins, then there exists a compartmental μ-slacked fractional packing of I into $(1 + 2\varepsilon/(1-\mu))\, m + 2$ bins.*

Proof. Consider a division-1 bin. By Property 5.7(b), we get that all dense items (if any) lie in dense compartments. Next, using the method of Section 3.2.3 in [66] (Rounding the Other Side > Containers for the wide and long rectangles), we can slice items and create non-overlapping compartments in the bin without moving any item such that all tall and wide items are packed into compartments. Their method works by first constructing tall compartments and then using the algorithm of Lemma 5.48 to partition the space outside tall compartments and big items into wide compartments. The resulting packing is compartmental, except that compartments' heights are not rounded. We will now show how to round the heights of compartments.

Let there be n_t tall compartments in the bin and n_{big} big items in the bin. Define

$$n_{\text{tcont}} := \frac{4}{\varepsilon_1^2}\left(\frac{1}{\varepsilon_1} - 1\right) \leq \frac{4}{\varepsilon_1^3} \qquad\qquad n_{\text{wcont}} := \frac{12}{\varepsilon_1^2}\left(\frac{1}{\varepsilon_1} - 1\right) + 1 \leq \frac{12}{\varepsilon_1^3}$$

In the bin, there are $4/\varepsilon_1^2$ slots of width $\varepsilon_1^2/4$ and height 1. Consider one such slot. Let there be k big items that intersect that slot (k can be 0). The height of each big item and each sparse tall compartment is more than ε_1. Therefore, the number of tall sparse compartments in that slot is at most $1/\varepsilon_1 - 1 - k$. Each big item spans at least $4/\varepsilon_1 + 1$ slots, and reduces by 1 the number of tall compartments in the slots it spans. Hence, the number of tall sparse

compartments is at most

$$n_t \le \frac{4}{\varepsilon_1^2}\left(\frac{1}{\varepsilon_1}-1\right) - n_{\text{big}}\left(\frac{4}{\varepsilon_1}+1\right) \le \frac{4}{\varepsilon_1^2}\left(\frac{1}{\varepsilon_1}-1\right) - n_{\text{big}} = n_{\text{tcont}} - n_{\text{big}}.$$

By Lemma 5.48, the number of wide compartments is at most $3(n_t + n_{\text{big}}) + 1 \le n_{\text{wcont}}$.

Since small items can be sliced in both dimensions, we can treat them like a liquid. For each tall sparse compartment C in the bin, let the tall items in C sink down in this liquid. Then shift down the top edge of C to the top edge of the topmost tall item in C (so some small items will no longer be inside C). Then the height of C will be the sum of the heights of at most $1/\varepsilon_1 - 1$ tall items inside C (since tall items have height $> \varepsilon_1$ and C has height at most 1).

For each wide compartment C in the bin, unpack a horizontal slice of height $h(C) \bmod \varepsilon_{\text{cont}}$ from C (this may require slicing items) and move down the top edge of C by $h(C) \bmod \varepsilon_{\text{cont}}$. This rounds down $h(C)$ to a multiple of $\varepsilon_{\text{cont}}$.

Apply the above transformation to all division-1 bins and an analogous transformation to all division-2 bins. This gives us a μ-slacked compartmental packing into m bins. However, we unpacked some items from wide containers in division-1 bins and tall containers in division-2 bins. We need to repack these items.

Let there be m_1 division-1 bins. We removed a slice of height less than $\varepsilon_{\text{cont}}$ from each wide compartment in division-1 bins. Let S be the set of all such slices from division-1 bins. There are at most n_{wcont} wide compartments, so $h(S) \le \varepsilon_{\text{cont}} n_{\text{wcont}} m_1$. For each slice $i \in S$, define

$$\text{span}'(i) := \max\left(h(i), \min\left(\frac{v_{\max}(i)}{1-\mu}, 1\right)\right).$$

For each slice $i \in S$,

$$\text{span}'(i) \le \max\left(h(i), \frac{v_{\max}(i)}{1-\mu}\right) \le \max\left(h(i), \frac{h(i)}{\varepsilon_1^2(1-\mu)}\right) \le \frac{h(i)}{\varepsilon_1^2(1-\mu)}.$$

Therefore,

$$\sum_{i\in S} \text{span}'(i) \le \frac{h(S)}{\varepsilon_1^2(1-\mu)} \le \frac{\varepsilon_{\text{cont}} n_{\text{wcont}}}{\varepsilon_1^2(1-\mu)} m_1.$$

Interpret each slice $i \in S$ as a 1D item of size $\text{span}'(i)$ and pack the slices one-over-the-other touching the left edge of bins using Next-Fit. By Lemma 5.25, we can pack them all into $1 + 2\sum_{i\in S} \text{span}'(i)$ bins that are μ-slacked. Since $\varepsilon_{\text{cont}} = \varepsilon\varepsilon_1^5/12 \le \varepsilon\varepsilon_1^2/n_{\text{wcont}}$, the number of

bins used to pack S is at most

$$\frac{2\varepsilon_{\text{cont}} n_{\text{wcont}}}{(1-\mu)\varepsilon_1^2} m_1 + 1 \le \frac{2\varepsilon}{1-\mu} m_1 + 1.$$

These bins are division-1 compartmental; they have just one wide compartment of width 1 and height 1.

Similarly, we can pack unpacked items from division-2 bins into at most $\frac{2\varepsilon}{1-\mu}(m - m_1) + 1$ division-2 compartmental μ-slacked bins. □

Theorem 5.50. *There is an output $(\widetilde{I}, D)$ of* `round`(I, ε) *such that $\widetilde{I}$ has a compartmental $(5\varepsilon/8)$-slacked fractional packing into at most $(1 + 2\varepsilon/(1-\varepsilon))^2 (a \operatorname{opt}(I) + b) + 4/(1-\varepsilon)$ bins. Here a and b are as per Table 5.1.*

Proof. By Theorem 5.47, there is an output $(\widetilde{I}, D)$ of `round`(I, ε) such that $\widetilde{I}$ can be packed into $m := (1 + 2\varepsilon/(1-\varepsilon))(a \operatorname{opt}(I) + b) + 2$ semi-structured $(5\varepsilon/8)$-slacked bins.

By Lemma 5.49, the number of compartmental $(5\varepsilon/8)$-slacked bins needed to fractionally pack $\widetilde{I}$ is at most $(1 + 2\varepsilon/(1-\varepsilon))\, m + 2 \le (1 + 2\varepsilon/(1-\varepsilon))^2 (a \operatorname{opt}(I) + b) + 4/(1-\varepsilon)$. □

Table 5.3: Upper bound on the number of distinct widths and heights for compartments of different types

Compartment type	no. of widths	no. of heights
Division-1 wide	$\frac{4}{\varepsilon_1^2}$	$\frac{12}{\varepsilon\varepsilon_1^5}$
Division-1 tall	1	$\left(\frac{(d+1)n_{\text{wwc}}}{\varepsilon\varepsilon_1} + 1\right)^{1/\varepsilon_1 - 1}$
Division-2 wide	$\left(\frac{(d+1)n_{\text{wwc}}}{\varepsilon\varepsilon_1} + 1\right)^{1/\varepsilon_1 - 1}$	1
Division-2 tall	$\frac{12}{\varepsilon\varepsilon_1^5}$	$\frac{4}{\varepsilon_1^2}$

Since division-1 tall items have $(d+1)n_{\text{wwc}}/\varepsilon\varepsilon_1$ possible heights, the number of possible heights of division-1 tall sparse compartments is $((d+1)n_{\text{wwc}}/\varepsilon\varepsilon_1 + 1)^{1/\varepsilon_1 - 1}$. This is a huge number of possible heights, and it is possible to reduce it by partitioning tall compartments into weight classes and using linear grouping. We will not perform this improvement here.

5.6 Packing Algorithm

Let I be a subset of the rounded items. Formally, let $(\widetilde{I}', D) \in \texttt{round}(I', \varepsilon)$ and $I \subseteq \widetilde{I}'$.

We will first present a polynomial-time algorithm $\texttt{fpack}(I, m)$ that takes as input I and an integer m and either outputs a fractional packing of I into m compartmental μ-slacked bins (where $\mu \leq \varepsilon$) or claims that fractionally packing I into m compartmental μ-slacked bins is impossible.

We can use $\texttt{fpack}(I, m)$ to find the optimal compartmental μ-slacked fractional packing of I by using binary search on m. With slight abuse of notation, let $\texttt{fpack}(I)$ denote this algorithm.

Then, we will present an algorithm that finds a μ-slacked (non-fractional) packing of I by using $\texttt{fpack}(I)$ as a subroutine. Note that we're interested in getting a non-fractional μ-slacked packing of I, but that packing need not be compartmental.

5.6.1 Guessing Bin Configurations

A μ-slacked bin J can have one of 4 possible slack types:

1. Normal: $\forall k \in [d], v_k(J) \leq 1 - \mu$.
2. Big single: $|J| = 1$ and J contains a big item and $\forall k \in [d], v_k(J) \in (1 - \mu, 1]$.
3. Dense single: $|J| = 1$ and J contains a dense item and $\forall k \in [d], v_k(J) \in (1 - \mu, 1]$.
4. Dense double: $|J| = 2$ and J contains two dense items and $\forall k \in [d], v_k(J) \in (1 - \mu, 1]$ and $\forall i \in J, v_{\max}(i) \leq 1/2$.

Note that these slack types are disjoint, i.e., a bin cannot have more than one slack types.

A configuration of a bin is defined to be all of this information: (i) The division type, (ii) The slack type, (iii) Whether the bin has a dense compartment, (iv) A packing of big items, heavy items and compartments into the bin.

We will now enumerate all possible configurations that a bin can have.

For a division-1 bin, there are $4(d+1)n_{\text{bwc}}/\varepsilon\varepsilon_1^3$ different types of big items, n_{hwc} different types of heavy items, $48/\varepsilon\varepsilon_1^7$ different types of wide compartments and $((d+1)n_{\text{wwc}}/\varepsilon\varepsilon_1+1)^{1/\varepsilon_1-1}$ different types of tall compartments. A bin can pack less than $1/\varepsilon_1^2$ big items, less than $1/\varepsilon_1$ heavy items at most n_{wcont} wide compartments and at most n_{tcont} tall compartments. Therefore, by iterating over

$$n'_{\text{nconfs}} := \left(\frac{4(d+1)n_{\text{bwc}}}{\varepsilon\varepsilon_1^3} + 1\right)^{1/\varepsilon_1^2-1} \left(\frac{48}{\varepsilon\varepsilon_1^7} + 1\right)^{n_{\text{wcont}}} \left(\left(\frac{(d+1)n_{\text{wwc}}}{\varepsilon\varepsilon_1} + 1\right)^{1/\varepsilon_1-1} + 1\right)^{n_{\text{tcont}}}$$

values (a large constant), we can guess the set of big items, heavy items and compartments in a division-1 bin of normal slack type that does not have a dense compartment. For a bin that has a dense compartment, the number configurations to consider is $n'_{\text{nconfs}}(n_{\text{hwc}}+1)^{1/\varepsilon_1-1}$.

For a bin of big-single slack type, there are at most $4(d+1)n_{\text{bwc}}/\varepsilon\varepsilon_1^3$ configurations. For a bin of dense-single slack type, there are at most n_{hwc} configurations. For a bin of dense-double slack type, there are at most n_{hwc}^2 configurations. Double the number of configurations to also account for division-2 items. Therefore, the total number of configurations is at most

$$n_{\text{nconfs}} := 2\left(n'_{\text{nconfs}}\left(1+(n_{\text{hwc}}+1)^{1/\varepsilon_1-1}\right)+\frac{4(d+1)n_{\text{bwc}}}{\varepsilon\varepsilon_1^3}+n_{\text{hwc}}+n_{\text{hwc}}^2\right).$$

There can be at most $n_{\text{wcont}}+n_{\text{tcont}}$ items and compartments in a bin (see the proof of Lemma 5.49). Since the x-coordinate of these items and compartments is a multiple of $\varepsilon_1^2/4$, we can use brute-force to check if they can be packed into a bin. For each item, guess its x-coordinate and its 'layer' number. This will take time $O\left((4(n_{\text{wcont}}+n_{\text{tcont}})/\varepsilon_1^2)^{n_{\text{wcont}}+n_{\text{tcont}}}\right)$.

For m bins, we can have at most $\binom{m+n_{\text{nconfs}}-1}{n_{\text{nconfs}}-1} \in O(m^{n_{\text{nconfs}}-1})$ possible combinations of configurations. Now for each combination, we will check if the remaining items can fit into the bins.

5.6.2 Fractionally Packing Wide, Tall, Small and Light items

We will use a linear program to fractionally pack wide, tall, small and light items. Note that bins of non-normal slack type can only have big and heavy items, which have already been packed, so we won't consider them any further.

Definition 5.26 (Length and Breadth)**.** *For an item i, let the length $\ell(i)$ be the longer geometric dimension and the breadth $b(i)$ be the shorter geometric dimension (so for a wide item i, $\ell(i) := w(i)$ and $b(i) := h(i)$, and for a tall item i, $\ell(i) := h(i)$ and $b(i) := w(i)$). Similarly define ℓ and b for wide and tall compartments.*

In any packing of items in a wide compartment, move a line horizontally upwards, as if scanning the compartment. The line, at each point, will intersect some wide items. The set of such wide items is called a 1D configuration. See Fig. 5.8 for an example. Similarly define 1D configuration for items in tall compartments. Any fractional packing of items inside a compartment can be described by mentioning the breadths of all 1D configurations in the compartment. The number of types of items is a constant and there can be at most $1/\varepsilon_1 - 1$ items in a 1D configuration. Therefore, the number of 1D configurations is a constant.

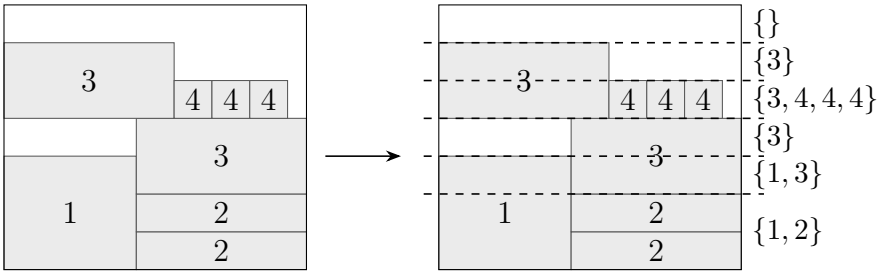


Figure 5.8: A compartment with wide items from 4 different fine partitions. This compartment has 5 distinct 1D configurations.

- Let M_1 and M_2 be the set of division-1 bins and division-2 bins respectively of normal slack type. Let $M := M_1 \cup M_2$. Let $M_D \subseteq M$ be the set of bins that have a dense compartment.
- Let the set of wide and tall non-dense item types be L. For $t \in L$, let $b(t)$ be the sum of breadths of items of type t and each item has length $\ell(t)$.
- Let J_i be the set of compartments in bin i.
- For compartment j, let $\mathcal{C}_j$ be the set of feasible 1D configurations and let $b(j)$ be the breadth of the compartment.
- Let ℓ_C be the sum of lengths of items in 1D configuration C.
- Let $n_{t,C}$ be the number of items of type t in 1D configuration C.
- Let $\alpha_{k,C}$ be the weight-to-breadth ratio of 1D configuration C in the k^{th} vector dimension.
- Let $\beta_{k,p}$ be the weight-to-area ratio of weight class p for small non-dense items.
- Let $\gamma_{k,p} := v_k(i)/v_{\max}(i)$ for a light dense item i in weight class p.
- Let B_i be the set of big items in bin i.
- Let H_i be the set of heavy items in bin i.
- Let D_i be 1 for bin i if it contains a dense compartment and 0 otherwise.

We will fractionally pack items using a linear program that only has constraints and has no objective function. We call it the fractional packing feasibility program FP. It has variables x, y and z, where

- $x_{j,C}$ is the breadth of 1D configuration C in compartment j.
- $y_{i,p}$ is the area of small non-dense items of weight class p in bin i.
- $z_{i,p}$ is the $v_{\max}$ of light dense items of weight class p in bin i. When $i \notin M_D$, $z_{i,p}$ is the constant 0 instead of being a variable.

Feasibility program FP:

$$\sum_{C \in \mathcal{C}_j} x_{j,C} = b(j) \qquad \forall i \in M, \forall j \in J_i$$

(compartment breadth constraint)

$$\sum_{j \in J_i} \sum_{C \in \mathcal{C}_j} \ell_C x_{j,C} + \sum_{p=1}^{n_{\text{swc}}} y_{i,p} \leq 1 - \frac{\varepsilon_1}{2} D_i - a(B_i)$$

$\forall i \in M$

(bin area constraint)

$$\sum_{j \in J_i} \sum_{C \in \mathcal{C}_j} \alpha_{k,C} x_{j,C} + \sum_{p=1}^{n_{\text{wwc}}} \beta_{k,p} y_{i,p} + \sum_{p=1}^{n_{\text{lwc}}} \gamma_{k,p} z_{i,p} \leq 1 - \mu - v_k(B_i) - v_k(H_i)$$

$\forall i \in M, \forall k \in [d]$

(bin weight constraint)

$$\sum_{i \in M} \sum_{j \in J_i} \sum_{\substack{C \in \mathcal{C}_j \\ C \ni t}} n_{t,C} x_{j,C} = b(t) \quad \forall t \in L$$

(conservation of wide and tall items)

$$\sum_{i \in M} y_{i,p} = a(S_p) \qquad \forall p \in [n_{\text{swc}}]$$

(conservation of small items)

$$\sum_{i \in M_1} z_{i,p} = v_{\max}(D_p^{l,w}) \qquad \forall p \in [n_{\text{lwc}}]$$

(conservation of division-1 light items)

$$\sum_{i \in M_2} z_{i,p} = v_{\max}(D_p^{l,h}) \qquad \forall p \in [n_{\text{lwc}}]$$

(conservation of division-2 light items)

$$x_{j,C} \geq 0 \qquad \forall i \in M, \forall j \in J_i, \forall C \in \mathcal{C}_j$$
$$y_{i,p} \geq 0 \qquad \forall i \in M, \forall p \in [n_{\text{swc}}]$$
$$z_{i,p} \geq 0 \qquad \forall i \in M_D, \forall p \in [n_{\text{lwc}}]$$

(non-negativity constraints)

The number of constraints in FP (other than the non-negativity constraints) is at most

$$\begin{aligned} n_c &:= m(n_{\text{wcont}} + n_{\text{tcont}} + d + 1) + 2n_{\text{wwc}} \left(\frac{4}{\varepsilon_1^2} + \frac{d+1}{\varepsilon\varepsilon_1} \right) + n_{\text{swc}} + 2n_{\text{lwc}} \\ &\leq \left(\frac{16}{\varepsilon_1^3} + d \right) m + 2n_{\text{wwc}} \left(\frac{4}{\varepsilon_1^2} + \frac{d+1}{\varepsilon\varepsilon_1} + 1 \right) && (2n_{\text{lwc}} \leq n_{\text{wwc}} \text{ and } n_{\text{swc}} \leq n_{\text{wwc}}) \\ &\leq \left(\frac{16}{\varepsilon_1^3} + d \right) m + 2(d+6) \frac{n_{\text{wwc}}}{\varepsilon_1^2}. \end{aligned}$$

The number of variables and constraints in FP are linear in m. Therefore, FP can be solved in time polynomial in m. Furthermore, if FP is feasible, we can obtain an extreme-point solution to FP.

Therefore, $\texttt{fpack}(I, m)$ guesses all combinations of configurations of m bins and for each such combination of configurations solves the feasibility program to check if the remaining items can be packed into the bins according to the bin configurations. Furthermore, $\texttt{fpack}(I, m)$ runs in time polynomial in m. $\texttt{fpack}(I)$ makes at most $O(\log n)$ calls to $\texttt{fpack}(I, m)$ with $m \leq n$. Therefore, $\texttt{fpack}(I)$ runs time polynomial in n.

5.6.3 Getting Containers from a Fractional Packing Solution

Suppose $\texttt{fpack}(I)$ outputs a fractional packing that uses m bins. In each compartment j, we create a slot of breadth $x_{j,C}$ for each 1D configuration C. Since we're given an extreme-point solution to FP, the number of slots is at most the number of constraints n_c by rank lemma. In each slot, we create $n_{t,C}$ containers of type $t \in L$ having length $\ell(t)$ and breadth $x_{j,C}$. See Fig. 5.9 for an example.

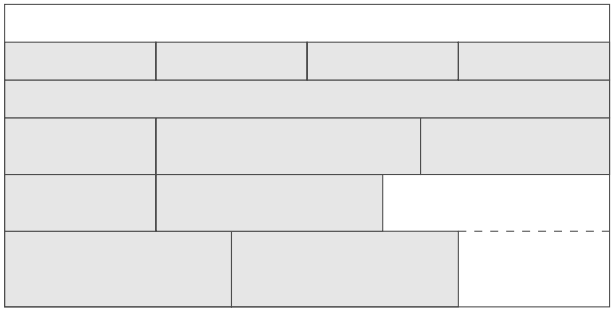


Figure 5.9: A compartment with 6 slots and 12 containers.

Now we (non-fractionally) pack a large subset of wide and tall items into containers and we pack a large subset of small non-dense and light dense items outside containers. In each wide container, items will be stacked one-over-the-other. In each tall container, items will be stacked

side-by-side. We pack the remaining unpacked items into a small number of new bins. This will give us a non-fractional packing that uses close to m bins.

5.6.4 Packing Light Dense Items

For light dense items, for each bin i and each weight class $p \in [n_{\text{lwc}}]$, keep adding items of the same division as the bin and from weight class p to the bin till the total $v_{\max}$ of the items exceeds $z_{i,p}$. Then discard the last item that was added.

As per the conservation constraints for light dense items, all items will either be packed or discarded. The $v_{\max}$ of items from weight class p that are packed into bin i is at most $z_{i,p}$.

The number of discarded items is at most the number of z-variables in FP, which is at most mn_{lwc}. Let D be the set of discarded items. Then $\text{span}(D) = v_{\max}(D) \leq (\varepsilon_2 n_{\text{lwc}})m$. We choose $\varepsilon_2 \leq \varepsilon/n_{\text{lwc}}$. Since $\varepsilon_2 \leq \varepsilon_1^2(1-\varepsilon)$, each item's weight is at most $1-\varepsilon$. Therefore, we can use Next-Fit to pack D into $2\,\text{span}(D)/(1-\mu) + 2 \leq 2\varepsilon m/(1-\mu) + 2$ number of μ-slacked bins (by scaling up each item's weight by $1/(1-\mu)$ before packing and scaling it back down after packing), where tall and small dense items are packed separately from wide dense items.

The time taken to pack these items is $O(|D_*^{l,*}|)$.

5.6.5 Packing Wide and Tall Non-Dense Items

For each item type t, iteratively pack items of type t into a container of type t till the total breadth of items in the container exceeds $x_{j,C}$. Then discard the last item and move to a new container and repeat. As per the conservation constraints for wide and tall items, all items will either be packed or discarded.

Treat the items discarded from each slot C as a single (composite) item of breadth ε_2, length ℓ_C and weight $\varepsilon_2/\varepsilon_1^2$ in each dimension. We will pack these composite items into bins, where wide and tall items are packed separately.

Let D be the set of all such discarded items. Then $|D| \leq n_c$. For a composite item i, let

$$\text{span}'(i) := \max\left(b(i), \min\left(\frac{v_{\max}(i)}{1-\mu}, 1\right)\right).$$

Treat each composite item as a 1D item of size $\text{span}'(i)$. Then by Lemma 5.25, we can use Next-Fit to pack these items into $2\,\text{span}'(D) + 2$ μ-slacked bins, where wide and tall items are

packed separately.

$$\operatorname{span}'(i) \le \max\left(b(i), \frac{v_{\max}(i)}{1-\mu}\right) \le \max\left(\varepsilon_2, \frac{\varepsilon_2}{\varepsilon_1^2(1-\mu)}\right) \le \frac{\varepsilon_2}{\varepsilon_1^2(1-\mu)}.$$

Therefore, the number of bins needed is

$$2\operatorname{span}'(D) + 2 \le 2\frac{\varepsilon_2}{\varepsilon_1^2(1-\mu)}n_c + 2 \le \frac{2\varepsilon_2(16+d\varepsilon_1^3)}{\varepsilon_1^5(1-\mu)}m + \frac{4(d+6)}{1-\mu}\frac{\varepsilon_2 n_{\text{wwc}}}{\varepsilon_1^4} + 2.$$

Therefore, we choose $\varepsilon_2 \le \varepsilon\varepsilon_1^5/(16+d\varepsilon_1^3)$ so that the number of new bins needed is at most

$$\frac{2\varepsilon}{1-\mu}m + 2 + \frac{4(d+6)}{1-\mu}\frac{\varepsilon_2 n_{\text{wwc}}}{\varepsilon_1^4}.$$

The time taken to pack these items is $O(|L|)$.

5.6.6 Packing Small Non-Dense Items

In each bin, there are at most $n_{\text{wcont}} + n_{\text{tcont}}$ compartments and big items (see the proof of Lemma 5.49). By Lemma 5.48, the free space outside compartments can be partitioned into

$$3(n_{\text{wcont}} + n_{\text{tcont}}) + 1 \le 3\left(\frac{16}{\varepsilon_1^2}\left(\frac{1}{\varepsilon_1} - 1\right) + 1\right) + 1 \le \frac{48}{\varepsilon_1^3}$$

rectangular regions. Suppose there are p_i slots in bin i for which $x_{j,C} > 0$. The sum of p_i over all bins is equal to n_c, the number of constraints in FP. Each slot may give us a free rectangular region in the compartment. Therefore, the free space in bin i can be partitioned into $p_i + 48/\varepsilon_1^3$ rectangular regions. Let R_i denote the set of these free rectangular regions.

Let M be the set of bins of normal slack type. Let $m := |M|$. Then

$$\sum_{i\in M} |R_i| \le n_c + \frac{48}{\varepsilon_1^3}m \le \left(\frac{64}{\varepsilon_1^3} + d\right)m.$$

This observation forms the basis of our algorithm `packSmall` (Algorithm 9) for packing small non-dense items.

Lemma 5.51. *Algorithm* `packSmall`(I, M) *doesn't fail at line 21.*

Proof. Since all items have area at most ε_2^2,

$$a(T) < a(r) - 2\varepsilon_2 + \varepsilon_2^2 \le a(r) - (w(r) + h(r))\varepsilon_2 + \varepsilon_2^2 = (w(r) - \varepsilon_2)(h(r) - \varepsilon_2).$$

Algorithm 9 `packSmall`(I, M, y): Here I is a set of items and M is the fractional packing output by `fpack`(I). (x, y, z) is a feasible solution to FP output by `fpack`(I).

Let S_p be the p^{th} coarse partition of small non-dense items in I.
$D_1 = D_2 = \{\}$ *// sets of items to discard*
for each bin $i \in M$ of normal slack type **do**
    Let R_i be the set of free rectangular regions in bin i.
    // Select a set of items to pack
    $\widehat{S} = \{\}$
    for $p \in [n_{\text{swc}}]$ **do**
        $S_{i,p} = \{\}$
        while $a(S_{i,p}) < y_{i,p}$ and $|S_p| > 0$ **do**
            Move an item from S_p to $S_{i,p}$.
        end while
        if $a(S_{i,p}) > y_{i,p}$ **then**
            Move the last item in $S_{i,p}$ to D_1.
        end if
        $\widehat{S} = \widehat{S} \cup S_{i,p}$
    end for
    // Pack those items into R_i
    while $|R_i| > 0$ and $|\widehat{S}| > 0$ **do**
        Remove a rectangle r from R_i.
        Let T be the smallest prefix of $\widehat{S}$ such that $a(T) \geq a(r) - 2\varepsilon_2$ or $T = \widehat{S}$
        Pack T into r using NFDH. *// We will prove that this is possible*
        $\widehat{S}$ `-=` T
    end while
    $D_2 = D_2 \cup \widehat{S}$
end for
return $D_1 \cup D_2$ *// discarded items*

Therefore, by Lemma 3.4, we get that T can be packed into r. □

Lemma 5.52. *In* `packSmall`(I, M)*, every small non-dense item is either packed or discarded.*

Proof by contradiction. Assume $\exists p \in [n_{\text{swc}}]$ such that there are items in S_p that are neither packed nor discarded. Therefore, for each bin i, at Algorithm 9, $a(S_{i,p}) \geq y_{i,p}$. Therefore, the total area of all items from S_p that are either packed or discarded is at least

$$\sum_{i \in M} y_{i,p} = a(S_p), \qquad \text{(by conservation constraint in FP)}$$

which means that all items have been packed or discarded, which is a contradiction. □

Lemma 5.53. *Let* $D_1 \cup D_2 =$ `packSmall`(I, M)*. Then* $a(D_1 \cup D_2) \leq \varepsilon_2 \left(n_{\text{swc}} + \frac{128}{\varepsilon_1^3} + 2d\right) m$.

Proof. During $\texttt{packSmall}(I, M)$, for each bin, $a(D_1)$ increases by at most $\varepsilon_2 n_{\text{swc}}$. Therefore, $a(D_1) \le \varepsilon_2 n_{\text{swc}} m$.

We know that $a(\widehat{S}) \le \sum_{p=1}^{n_{\text{swc}}} y_{i,p} \le a(R_i)$. The first inequality follows from the way we chose $\widehat{S}$. The second inequality follows from the area constraint in FP.

Case 1: We used up all items in $\widehat{S}$ during bin i:
We didn't discard any items during bin i.

Case 2: We used up all rectangles in R_i during bin i:
Then the used area is at least $a(R_i) - 2\varepsilon_2|R_i|$. Therefore, the items discarded during bin i have area at most $a(\widehat{S}) - a(R_i) + 2\varepsilon_2|R_i| \le 2\varepsilon_2|R_i|$. Therefore,

$$a(D_2) \le 2\varepsilon_2 \sum_{i \in M} |R_i| \le 2\varepsilon_2 \left(\frac{64}{\varepsilon_1^3} + d\right) m. \qquad \square$$

Lemma 5.54. *Let $D := \texttt{packSmall}(I, M)$. Then we can pack D into $2\varepsilon m/(1-\mu) + 1$ number of μ-slacked bins, where $\mu \le \varepsilon$.*

Proof. Since $\varepsilon_2^2 \le (1-\varepsilon)\varepsilon_1^2$, we get that $\forall i \in D, v_{\max}(i) \le 1 - \varepsilon \le 1 - \mu$. Let $\text{span}'(i) = a(i)/\varepsilon_1^2(1-\mu)$. Then by interpreting each $i \in D$ as a 1D item of size $\text{span}'(i)$, we can pack them into $2\,\text{span}'(D) + 1$ bins using Next-Fit. In each bin J, $v_{\max}(J) \le a(J)/\varepsilon_1^2 = (1-\mu)\,\text{span}'(J) \le 1 - \mu$. Also, $a(J) \le \varepsilon_1^2(1-\mu)\,\text{span}'(J) \le (1-\varepsilon_2)^2$. Therefore, the bin is μ-slacked and by Lemma 3.4, we can pack items J in the bin using NFDH.

We choose $\varepsilon_2 \le \varepsilon\varepsilon_1^2/\left(n_{\text{swc}} + 128/\varepsilon_1^3 + 2d\right)$. Therefore, the number of bins needed is at most

$$2\,\text{span}'(D) + 1 \le \frac{2a(D)}{\varepsilon_1^2(1-\mu)} + 1 \le \frac{2\varepsilon}{1-\mu} m + 1. \qquad \square$$

The time taken to pack small items is $O(n_S \log n_S)$, where n_S is the number of small items, because we need to sort items by height for NFDH.

5.6.7 The Algorithm and its Approximation Factor

We give an algorithm $\texttt{ipack}_\mu$ (Algorithm 10) for packing a subset I of rounded items.

Theorem 5.55. *Let $(\widetilde{I}', D) \in \texttt{round}(I', \varepsilon)$ and $I \subseteq \widetilde{I}'$. Let there be m bins in the optimal μ-slacked compartmental fractional packing of I, where $\mu \le \varepsilon$. Then $\texttt{ipack}_\mu(I)$ runs in polynomial time and outputs a μ-slacked packing of I into*

$$\left(1 + \frac{6\varepsilon}{1-\mu}\right) m + 5 + \frac{4(d+6)}{1-\mu} \frac{\varepsilon_2 n_{\text{wwc}}}{\varepsilon_1^4}$$

Algorithm 10 $\texttt{ipack}_{\mu}(I)$: Computes a non-fractional μ-slacked packing of I. Here $\mu \leq \varepsilon$ and $I \subseteq \widetilde{I}'$ and $(D, \widetilde{I}') \in \texttt{round}(I')$ for some set I' of items.

1: Let $(J, x, y, z) := \texttt{fpack}_{\mu}(I)$. Here J is a fractional packing of I into m bins and (x, y, z) is a feasible solution to the feasibility program FP.
2: Create containers inside compartments in J using x as per Section 5.6.3.
3: Pack light dense items into dense compartments using z as per Section 5.6.4.
4: Pack wide and tall non-dense items into containers as per Section 5.6.5.
5: $\texttt{packSmall}(I, J, y)$ *// Pack small non-dense items outside containers.*

bins, where each bin satisfies either Property 5.7(b) or Property 5.8(b).

Proof. $\texttt{fpack}(I)$ finds the optimal μ-slacked compartmental fractional packing of I in polynomial time. Given the output of $\texttt{fpack}(I)$, $\texttt{ipack}$ can, in $O(n \log n)$ time, compute a packing of I into $(1 + 6\varepsilon/(1-\mu))\, m + 5 + 4(d+6)\varepsilon_2 n_{\text{wwc}}/\varepsilon_1^4(1-\mu)$ number of μ-slacked bins.

Each bin satisfies either Property 5.7(b) or Property 5.8(b). This is because the m bins output by the fractional packing are compartmental, and the extra bins either contain only dense tall and small items or only dense wide items or only non-dense items. □

We choose $\varepsilon_2 := \lceil \max\left(n_{\text{lwc}}/\varepsilon, (16+d^3)/\varepsilon\varepsilon_1^5, (128 + \varepsilon_1^3(n_{\text{swc}} + 2d))/\varepsilon\varepsilon_1^5\right) \rceil^{-1}$. Therefore, $\varepsilon_2^{-1} \in O(\varepsilon_1^{-5} + \varepsilon^{-d}\varepsilon_1^{-(2d+2)})$. So, the parameter f in $\texttt{removeMedium}(I, \varepsilon, f, \delta_0)$ is

$$f(x) = \left\lceil \max\left(\frac{n_{\text{lwc}}}{\varepsilon}, \frac{16+d^3}{\varepsilon x^5}, \frac{128 + x^3((8/x^2\varepsilon)^d + 2d)}{\varepsilon x^5} \right) \right\rceil^{-1}.$$

Theorem 5.56. *Let I be a (2, d) bin packing instance. For some $(\widetilde{I}, D) \in \texttt{round}(I, \varepsilon)$, if we pack D using* simplePack *and pack $\widetilde{I}$ using* ipack*, we get a packing of I into at most*

$$((1+17\varepsilon)a + 6\varepsilon(d+1))\,\text{opt}(I) + O(1) + \frac{4(10d+51)}{1-\varepsilon}\frac{\varepsilon_2 n_{\text{wwc}}}{\varepsilon_1^4}$$

bins. Here a is the constant defined in Table 5.1 in Section 5.3.2.

Proof. By Theorem 5.50, $\exists(\widetilde{I}, D) \in \texttt{round}(I, \varepsilon)$ such that $\widetilde{I}$ has a compartmental $(5\varepsilon/8)$-slacked fractional packing into at most $m := (1 + 2\varepsilon/(1-\varepsilon))^2\, a\,\text{opt}(I) + O(1)$ bins.

By Theorem 4.5 and Lemmas 4.1 and 5.44, we get

$$\begin{aligned} |\texttt{simplePack}(D)| &\leq 6\,\text{span}(D) + 3 \leq 6\left(\varepsilon\,\text{span}(I) + 6(d+5)\frac{\varepsilon_2 n_{\text{wwc}}}{\varepsilon_1^4}\right) + 3 \\ &\leq 6\varepsilon(d+1)\,\text{opt}(I) + \left(3 + 36(d+5)\frac{\varepsilon_2 n_{\text{wwc}}}{\varepsilon_1^4}\right). \end{aligned}$$

Let $J := \texttt{ipack}_\mu(\widetilde{I})$, where $\mu := 5\varepsilon/8$. Then the bins in J are $(5\varepsilon/8)$-slacked. The number of bins in J is at most

$$\begin{aligned}
&\left(1+\frac{6\varepsilon}{1-\mu}\right)m + 5 + \frac{4(d+6)}{1-\mu}\frac{\varepsilon_2 n_{\text{wwc}}}{\varepsilon_1^4} \\
&\leq \left(1+\frac{2\varepsilon}{1-\varepsilon}\right)^2\left(1+\frac{6\varepsilon}{1-\varepsilon}\right)a\,\text{opt}(I) + O(1) + \frac{4(d+6)}{1-\varepsilon}\frac{\varepsilon_2 n_{\text{wwc}}}{\varepsilon_1^4} \\
&\leq (1+17\varepsilon)a\,\text{opt}(I) + O(1) + \frac{4(d+6)}{1-\varepsilon}\frac{\varepsilon_2 n_{\text{wwc}}}{\varepsilon_1^4}. \qquad (\varepsilon \leq 1/8)
\end{aligned}$$

To get a packing of $I - D$ from a packing of $\widetilde{I}$, we need to revert the transformations done by `round`. The only transformation in `round` where we rounded down instead of rounding up is Transformation 5.14. As per Lemma 5.20, it is possible to undo Transformation 5.14 because the bins are $(5\varepsilon/8)$-slacked and each bin satisfies either Property 5.7(b) or Property 5.8(b). □

By Theorem 5.56, we get that for every $\varepsilon' > 0$, there is a $(a + \varepsilon')$-asymptotic-approx algorithm for $(2, d)$ bin packing. We can get a better approximation factor by using the Round-and-Approx Framework, and we call the resulting algorithm `cbPack`.

5.7 Using the Round-and-Approx Framework

To use the Round-and-Approx (R&A) framework, we must show how to implement `round`, `complexPack`, `unround` and `solveConfigLP`, and we must prove the structural theorem.

1. `solveConfigLP`(I): Using the algorithm of [53] for $(2, d)$ KS and the LP algorithm of [72], we get a $2(1+\varepsilon)$-approximate solution to configLP(I). Therefore, $\mu = 2(1+\varepsilon)$.

2. `round`: We can use Algorithm 8 as `round`. By Lemma 5.43, `round` runs in polynomial time. By Lemma 5.44, `round` has low discard. By Lemma 5.45, `round` partitions items into a constant number of classes.

3. **Structural theorem**: Theorem 5.50. We call a packing structured iff it is compartmental and $(5\varepsilon/8)$-slacked. Here $\rho = a(1 + 2\varepsilon/(1-\varepsilon))^2$.

4. `complexPack`$(\widetilde{S})$: Use $\texttt{ipack}_\mu$ as `complexPack`, where $\mu = 5\varepsilon/8$. By Theorem 5.55, $\alpha = 1 + 6\varepsilon/(1-\varepsilon)$.

5. `unround`$(\widetilde{J})$: Since the output of `ipack` is $(5\varepsilon/8)$-slacked and each bin satisfies either Property 5.7(b) or Property 5.8(b), we can use Lemma 5.20 to undo Transformation 5.14. The other transformations round up, so they are trivial to undo. Therefore, $\gamma = 1$.

The only remaining requirement of R&A is proving the bounded expansion lemma.

Lemma 5.57 (Bounded expansion)**.** *Let* $(\widetilde{I}, D) \in \texttt{round}(I, \varepsilon)$*. Let* $\widetilde{K} \subseteq \widetilde{I}$ *be a fine partition of* $\widetilde{I}$*. Let* $C \subseteq I$ *be a set of items that fit into a bin and let* $\widetilde{C}$ *be the corresponding rounded items. Then* $\operatorname{span}(\widetilde{K} \cap \widetilde{C})$ *is upper-bounded by* $1/\varepsilon_1 + 1/4$*.*

Proof. To prove this, it is sufficient to prove that $\forall i \in I - D$, if item i gets rounded to item $\tilde{\imath}$, then $\operatorname{span}(\tilde{\imath})/\operatorname{span}(i)$ is upper-bounded by $1/\varepsilon_1 + 1/4$.

1. **Big items**: We will consider division-1 big items. The analysis for division-2 big items is analogous.

$$
\begin{aligned}
& w(\tilde{\imath}) \le w(i) + \varepsilon_1^2/4 \wedge h(\tilde{\imath}) \le 1 && \text{(by Transformation 5.21)} \\
& \implies \frac{w(\tilde{\imath})}{w(i)} \le 1 + \frac{\varepsilon_1}{4} \wedge \frac{h(\tilde{\imath})}{h(i)} \le \frac{1}{\varepsilon_1} \\
& \implies \frac{a(\tilde{\imath})}{\operatorname{span}(i)} \le \frac{a(\tilde{\imath})}{a(i)} \le \frac{w(\tilde{\imath})}{w(i)} \frac{h(\tilde{\imath})}{h(i)} \le \frac{1}{\varepsilon_1} + \frac{1}{4}
\end{aligned}
$$

$$
\begin{aligned}
\frac{v_{\max}(\tilde{\imath})}{\operatorname{span}(i)} &\le \frac{v_{\max}(i) + \varepsilon_1^2 \varepsilon/8}{\max(a(i), v_{\max}(i))} && \text{(by Transformation 5.13)} \\
&\le \min\left(\frac{v_{\max}(i)}{\varepsilon_1^2} + \frac{\varepsilon}{8}, 1 + \frac{\varepsilon_1^2 \varepsilon}{8 v_{\max}(i)}\right) \\
&\le 1 + \min\left(\frac{v_{\max}(i)}{\varepsilon_1^2}, \frac{\varepsilon}{8} \frac{\varepsilon_1^2}{v_{\max}(i)}\right) \\
&\le 1 + \sqrt{\frac{\varepsilon}{8}} \le \frac{3}{2} \le \frac{1}{\varepsilon_1} + \frac{1}{4} && (\varepsilon_1 \le 2/3)
\end{aligned}
$$

2. **Non-dense wide items**:

$$
\frac{a(\tilde{\imath})}{\operatorname{span}(i)} \le \frac{a(\tilde{\imath})}{a(i)} = \frac{w(\tilde{\imath})}{w(i)} \le \frac{1}{\varepsilon_1}
$$

$$\begin{aligned}
\frac{v_{\max}(\tilde{i})}{\operatorname{span}(i)} &\leq \frac{v_{\max}(i) + h(i)(\varepsilon_1\varepsilon/8)}{\max(h(i)\varepsilon_1, v_{\max}(i))} && \text{(by Transformation 5.13)} \\
&\leq \min\left(\frac{v_{\max}(i)}{\varepsilon_1 h(i)} + \frac{\varepsilon}{8}, 1 + \frac{\varepsilon}{8}\frac{\varepsilon_1 h(i)}{v_{\max}(i)}\right) \\
&\leq 1 + \min\left(\frac{v_{\max}(i)}{\varepsilon_1 h(i)}, \frac{\varepsilon}{8}\frac{\varepsilon_1 h(i)}{v_{\max}(i)}\right) \\
&\leq 1 + \sqrt{\frac{\varepsilon}{8}} \leq \frac{3}{2} \leq \frac{1}{\varepsilon_1} && (\varepsilon_1 \leq 2/3)
\end{aligned}$$

3. **Non-dense tall items**: Similar to the non-dense wide items case.

4. **Non-dense small items**: $a(\tilde{i}) = a(i)$.

$$\begin{aligned}
\frac{v_{\max}(\tilde{i})}{\operatorname{span}(i)} &\leq \frac{v_{\max}(i) + a(i)(\varepsilon/8)}{\max(a(i), v_{\max}(i))} && \text{(by Transformation 5.13)} \\
&\leq \min\left(\frac{v_{\max}(i)}{a(i)} + \frac{\varepsilon}{8}, 1 + \frac{\varepsilon}{8}\frac{a(i)}{v_{\max}(i)}\right) \\
&\leq 1 + \min\left(\frac{v_{\max}(\tilde{i})}{a(i)}, \frac{\varepsilon}{8}\frac{a(i)}{v_{\max}(i)}\right) \\
&\leq 1 + \sqrt{\frac{\varepsilon}{8}} \leq \frac{3}{2} \leq \frac{1}{\varepsilon_1} && (\varepsilon_1 \leq 2/3)
\end{aligned}$$

5. **Heavy dense items**: (See Transformation 5.15)

$$\frac{\operatorname{span}(\tilde{i})}{\operatorname{span}(i)} \leq \frac{v_{\max}(\tilde{i})}{v_{\max}(i)} \leq 1 + \frac{\varepsilon_1\varepsilon}{8v_{\max}(i)} \leq 1 + \frac{\varepsilon}{8} \leq \frac{1}{\varepsilon_1}$$

6. **Light dense items**: (See Transformation 5.16)

$$\frac{\operatorname{span}(\tilde{i})}{\operatorname{span}(i)} \leq \frac{v_{\max}(\tilde{i})}{v_{\max}(i)} \leq 1 + \frac{\varepsilon}{8} \leq \frac{1}{\varepsilon_1}$$

□

The asymptotic approximation factor given by the Round-and-Approx framework is

$$\mu\left(1 + \frac{\alpha\rho\gamma}{\mu}\right) + \Theta(1)\varepsilon = 2\left(1 + \ln\left(\frac{a}{2}\right)\right) + \Theta(1)\varepsilon$$

Using Table 5.1, for $d = 1$, we get $2.919065+\varepsilon'$ when item rotations are forbidden and $2.810930+\varepsilon'$ when item rotations are allowed.

Chapter 6

Harmonic Algorithms for dD Geometric Bin Packing

In this chapter, we focus on the d-dimensional multiple-choice geometric bin packing problem (dMCBP). This problem generalizes both the non-rotational and rotational versions of dD GBP. See Section 1.2.2 for a detailed introduction to this problem and its significance.

In dMCBP, we are given a set $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ as input, where for each j, I_j is a set of items, henceforth called an *itemset*. We have to pick exactly one item from each itemset and pack those items into the minimum number of bins.

Preliminaries

For convenience, in this chapter only, we will denote non-rotational dD GBP by dBP and non-rotational dD strip packing by dSP. In dBP and dMCBP, we assume without loss of generality that bins are dD cubes of side length 1.

We now give an equivalent definition of dMCBP for notational convenience. Let $\mathcal{I}$ be a set of itemsets. Let K be a set of items that contains exactly one item from each itemset in $\mathcal{I}$. Formally, for each itemset $I \in \mathcal{I}$, $|K \cap I| = 1$. Then K is called an *assortment* of $\mathcal{I}$. Let $\Psi(\mathcal{I})$ denote the set of all assortments of $\mathcal{I}$. In dMCBP, given an input instance $\mathcal{I}$, we have to select an assortment $K \in \Psi(\mathcal{I})$ and output a bin packing of K, such that the number of bins used is minimized. Therefore, $\mathrm{opt}_{d\mathrm{MCBP}}(\mathcal{I}) = \min_{K \in \Psi(\mathcal{I})} \mathrm{opt}_{d\mathrm{BP}}(K)$.

Define $\mathrm{flat}(\mathcal{I})$ as the union of all itemsets in $\mathcal{I}$. Then $n := |\mathcal{I}|$ is the number of itemsets in $\mathcal{I}$ and $N := |\mathrm{flat}(\mathcal{I})|$ is the total number of items across all itemsets of $\mathcal{I}$.

Overview of the Chapter

- In Section 6.1, we describe ideas from $\texttt{HDH}_k$ [18] that help us devise harmonic-based algorithms for dMCBP. For two of these ideas, $\texttt{HDH-unit-pack}_k$ and weighting functions, we give more details in Sections 6.5 and 6.8, respectively.
- In Section 6.2, we show a simple $O(N+n\log n)$-time algorithm for dMCBP, called $\texttt{fullh}_k$, having an AAR of T_k^d, where n is the number of itemsets and N is total number of items across all the n itemsets.
- We present an algorithm for dMCBP, called $\texttt{HGaP}_k$, having an AAR of $T_k^{d-1}(1+\varepsilon)$ and having a running time of $N^{O(1/\varepsilon^2)}n^{(1/\varepsilon)^{O(1/\varepsilon)}}$. We give an overview of $\texttt{HGaP}_k$ in Section 6.3 and give its details in Section 6.4.
- In Section 6.6, we define the dD multiple-choice strip packing problem (dMCSP) and extend Caprara's $\texttt{HDH}_k$ algorithm [18] to dMCSP. The algorithm has an AAR of T_k^{d-1} and runs in time $O(N+n\log n)$.
- In Section 6.7, we define the dD multiple-choice geometric knapsack problem (dMCKS), and for any $0<\varepsilon<1$, we show an $O(N\log N+Nn/\varepsilon)$-time algorithm that is $3^d(1+\varepsilon)$-approximate.
- Caprara [18] showed that no shelf-based algorithm for 2BP or 2SP can get an AAR better than $T_\infty\approx 1.69103$, and his $\texttt{HDH}_k$ algorithm achieves an AAR of T_k^{d-1} for dBP and dSP. In Section 6.9, we extend that result to show that no shelf-based algorithm for dBP or dSP can get an AAR better than T_∞^{d-1}.

6.1 Important Ideas from the $\texttt{HDH}_k$ Algorithm

In this section, we will describe some important ideas behind the $\texttt{HDH}_k$ algorithm for dBP by Caprara [18]. These ideas are the building blocks for our algorithms for dMCBP.

6.1.1 Weighting Functions

Fekete and Schepers [32] present a useful approach for obtaining lower bounds on the optimal solution to bin packing problems. Their approach is based on *weighting functions*.

Definition 6.1. *$g : [0,1] \mapsto [0,1]$ is a weighting function iff for all $m \in \mathbb{Z}_{>0}$ and $x \in [0,1]^m$,*

$$\sum_{i=1}^{m} x_i \leq 1 \implies \sum_{i=1}^{m} g(x_i) \leq 1$$

(Weighting functions are also called dual feasible functions *(DFFs)).*

Theorem 6.1. *Let I be a set of dD items that can be packed into a bin. Let $g_1, g_2, \ldots, g_d$ be weighting functions. For $i \in I$, define $g(i)$ as the item whose length is $g_j(\ell_j(i))$ in the j^{th} dimension. Then $\{g(i) : i \in I\}$ can be packed into a dD bin (without rotating the items).*

Theorem 6.1 is proved in Section 6.8.

6.1.2 The Harmonic Function

To obtain a lower-bound on $\text{opt}_{d\text{BP}}(I)$ using Theorem 6.1, Caprara [18] defined a function f_k. For an integer constant $k \geq 3$, $f_k : [0,1] \mapsto [0,1]$ is defined as

$$f_k(x) := \begin{cases} \frac{1}{q} & x \in \left(\frac{1}{q+1}, \frac{1}{q}\right] \text{ for } q \in [k-1] \\ \frac{k}{k-2}x & x \leq \frac{1}{k} \end{cases}.$$

f_k was originally defined and studied by Lee and Lee [56] for their online algorithm for 1BP, except that they used $k/(k-1)$ instead of $k/(k-2)$. Define $\text{type}_k : [0,1] \mapsto [k]$ as

$$\text{type}_k(x) := \begin{cases} q & x \in \left(\frac{1}{q+1}, \frac{1}{q}\right] \text{ for } q \in [k-1] \\ k & x \leq \frac{1}{k} \end{cases}.$$

Define T_k to be the smallest positive constant such that $H_k(x) := f_k(x)/T_k$ is a weighting function. We call H_k the *harmonic weighting function*. We can efficiently compute T_k as a function of k using ideas from [56, 71]. Table 6.1 lists the values of T_k for the first few k. It can also be proven that T_k is a decreasing function of k and $T_\infty := \lim_{k\to\infty} T_k \approx 1.6910302$.

Table 6.1: Values of T_k.

k	3	4	5	6	7	∞
T_k	3	2	$11/6 = 1.8\overline{3}$	$7/4 = 1.75$	$26/15 = 1.7\overline{3}$	≈ 1.6910302

For a dD cuboid i, define $f_k(i)$ to be the cuboid whose length is $f_k(\ell_j(i))$ in the j^{th} dimension. For a set I of dD cuboids, let $f_k(I) := \{f_k(i) : i \in I\}$. Similarly define $H_k(i)$ and $H_k(I)$. Define

type(i) to be a d-dimensional vector whose j^{th} component is type$_k(\ell_j(i))$. Note that there can be at most k^d different values of type(i). Sometimes, for the sake of convenience, we may express type(i) as an integer in $[k^d]$.

Theorem 6.2. *For a set of I of dD items,* $\text{vol}(f_k(I)) \le T_k^d \operatorname{opt}_{d\text{BP}}(I)$.

Proof. Let $m := \operatorname{opt}_{d\text{BP}}(I)$. Let J_j be the items in the j^{th} bin in the optimal bin packing of I. By Theorem 6.1 and because H_k is a weighting function, $H_k(J_j)$ fits in a bin. Therefore,

$$\text{vol}(f_k(I)) = \sum_{j=1}^{m} T_k^d \,\text{vol}(H_k(J_j)) \le \sum_{j=1}^{m} T_k^d = T_k^d \operatorname{opt}_{d\text{BP}}(I). \qquad \square$$

6.1.3 The HDH-unit-pack$_k$ Subroutine

From the HDH$_k$ algorithm by Caprara [18], we extracted out a useful subroutine, which we call HDH-unit-pack$_k$, that satisfies the following useful property:

Property 6.2. *The algorithm* HDH-unit-pack$_k^{[t]}(I)$ *takes a* sequence *I of dD items such that all items have type t and* $\text{vol}(f_k(I - \{\text{last}(I)\})) < 1$ *(here* last(I) *is the last item in sequence I). It returns a packing of I into a single dD bin in $O(n \log n)$ time, where $n := |I|$.*

The design of HDH-unit-pack$_k$ and its correctness can be inferred from Lemma 4.1 in [18]. We use HDH-unit-pack$_k$ as a black-box subroutine in our algorithms, i.e., we only rely on Property 6.2; we don't need to know anything else about HDH-unit-pack$_k$. Nevertheless, for the sake of completeness, in Section 6.5, we give a complete description of HDH-unit-pack$_k$ and prove its correctness.

6.2 Fast and Simple Algorithm for dMCBP (fullh$_k$)

We will now describe an algorithm for dBP called the *full-harmonic algorithm* (fullh$_k$). We will then extend it to dMCBP.

fullh$_k$ works by first partitioning the items based on their type vector (type vector is defined in Section 6.1.2). Then for each partition, it repeatedly picks the smallest prefix J such that $\text{vol}(f_k(J)) \ge 1$ and packs J into a dD bin using HDH-unit-pack$_k$. See Algorithm 11 for a more precise description of fullh$_k$. Note that fullh$_k(I)$ has a running time of $O(|I| \log |I|)$.

Theorem 6.3. *The number of bins used by* fullh$_k(I)$ *is less than $Q + \text{vol}(f_k(I))$, where Q is the number of distinct* types *of items (so $Q \le k^d$).*

Algorithm 11 $\texttt{fullh}_k(I)$: Returns a bin packing of dD items I.

Let P be an empty list.
for each type t **do**
  $I^{[t]} = \{i \in I : \text{type}(i) = t\}$.
  while $|I^{[t]}| > 0$ **do**
    Find J, the smallest prefix of $I^{[t]}$ such that $J = I^{[t]}$ or $\text{vol}(f_k(J))) \geq 1$.
    $B = \texttt{HDH-unit-pack}_k^{[t]}(J)$. *// B is a packing of J into a dD bin.*
    Append B to the list P.
    Remove J from $I^{[t]}$.
  end while
end for
return the list P of bins.

Proof. Let $I^{[t]}$ be the items in I of type t. Suppose $\texttt{fullh}_k(I)$ uses $m^{[t]}$ bins to pack $I^{[t]}$. For each type t, the first $m^{[t]} - 1$ bins have vol $\cdot f_k$ at least 1, so $\text{vol}(f_k(I^{[t]})) > m^{[t]} - 1$. Therefore, total number of bins used is $\sum_{t=1}^{Q} m^{[t]} < \sum_{t=1}^{Q}(1 + \text{vol}(f_k(I^{[t]}))) = Q + \text{vol}(f_k(I))$. □

Lemma 6.4 (Corollary to Theorems 6.2 and 6.3)**.** $\texttt{fullh}_k(I)$ *uses less than* $Q + T_k^d \,\text{opt}_{d\text{BP}}(I)$ *bins, where* Q *is the number of distinct item* types.

Theorem 6.5. *Let $\mathcal{I}$ be a dMCBP instance. Let $\widehat{K} := \{\text{argmin}_{i \in I} \text{vol}(f_k(i)) : I \in \mathcal{I}\}$, i.e., $\widehat{K}$ is the assortment obtained by picking from each itemset the item i having the minimum value of* $\text{vol}(f_k(i))$*. Then the number of bins used by* $\texttt{fullh}_k(\widehat{K})$ *is less than* $Q + T_k^d \,\text{opt}_{d\text{MCBP}}(\mathcal{I})$*, where Q is the number of distinct* types *of items in* $\text{flat}(\mathcal{I})$ *(so $Q \leq k^d$).*

Proof. For any assortment K, $\text{vol}(f_k(\widehat{K})) \leq \text{vol}(f_k(K))$. Let K^* be the assortment in an optimal packing of $\mathcal{I}$. By Theorems 6.2 and 6.3, the number of bins used by $\texttt{fullh}_k(\widehat{K})$ is less than

$$Q + \text{vol}(f_k(\widehat{K})) \leq Q + \text{vol}(f_k(K^*)) \leq Q + T_k^d \,\text{opt}_{d\text{BP}}(K^*) = Q + T_k^d \,\text{opt}_{d\text{MCBP}}(\mathcal{I}). \qquad \square$$

Let $N := |\,\text{flat}(\mathcal{I})|$ and $n := |\mathcal{I}|$. We can find $\widehat{K}$ in $O(N)$ time and compute $\texttt{fullh}_k(\widehat{K})$ in $O(n \log n)$ time. This gives us an $O(N + n \log n)$-time algorithm for dMCBP having AAR T_k^d.

6.3 Better Algorithm for dMCBP ($\texttt{HGaP}_k$)

Here we will describe a $T_k^{d-1}(1+\varepsilon)$-asymptotic-approximate algorithm for dMCBP that is based on $\texttt{HDH}_k$ and Lueker and de la Vega's APTAS for 1BP [26]. We call our algorithm *Harmonic Guess-and-Pack* ($\texttt{HGaP}_k$). This improves upon $\texttt{fullh}_k$ that has AAR T_k^d.

Definition 6.3. *For a dD item i, let $h(i) := \ell_d(i)$, $w(i) := \prod_{j=1}^{d-1} f_k(\ell_j(i))$ and $a(i) := w(i)h(i)$. Let $\texttt{round}(i)$ be a rectangle of height $h(i)$ and width $w(i)$. For a set X of dD items, define $w(X) := \sum_{i \in X} w(i)$ and $\texttt{round}(X) := \{\texttt{round}(i) : i \in X\}$.*

For any $\varepsilon > 0$, the algorithm $\texttt{HGaP}_k(\mathcal{I}, \varepsilon)$ returns a bin packing of $\mathcal{I}$, where $\mathcal{I}$ is a set of dD itemsets. $\texttt{HGaP}_k$ first converts $\mathcal{I}$ to a set $\widehat{\mathcal{I}}$ of 2D itemsets. It then computes P_{best}, which is a *structured* bin packing of $\widehat{\mathcal{I}}$ (we formally define *structured* later). Finally, it uses the algorithm `inflate` to convert P_{best} into a bin packing of the dD itemsets $\mathcal{I}$, where $|\texttt{inflate}(P_{\text{best}})|$ is very close to $|P_{\text{best}}|$. See Algorithm 12 for a more precise description. This approach of converting items to 2D, packing them, and then converting back to dD is very useful, because most of our analysis is about how to compute a structured 2D packing, and a packing of 2D items is easier to visualize and reason about than a packing of dD items.

Algorithm 12 $\texttt{HGaP}_k(\mathcal{I}, \varepsilon)$: Returns a bin packing of dD itemsets $\mathcal{I}$, where $\varepsilon \in (0, 1)$.

```
Let δ := ε/(2 + ε).
Ī = {round(I) : I ∈ I}
Initialize P_best to null.
for P ∈ guessShelves(Ī, δ) do
    P̄ = chooseAndPack(Ī, P, δ)
    if P̄ is not null and (P_best is null or |P̄| ≤ |P_best|) then
        P_best = P̄
    end if
end for
return inflate(P_best)
```

A bin packing is said to be *shelf-based* if items are packed into *shelves* and the shelves are packed into bins, where a shelf is a rectangle of width 1. See Fig. 6.1 for an example. A structured bin packing is a shelf-based bin packing where the heights of the shelves satisfy some additional properties (we describe these properties later). The algorithm `guessShelves` repeatedly guesses the number and heights of shelves and computes a structured packing P of those shelves into bins. Then for each packing P, the algorithm $\texttt{chooseAndPack}(\widehat{\mathcal{I}}, P, \delta)$ *tries to* pack an assortment of $\widehat{\mathcal{I}}$ into the shelves in P plus maybe one additional shelf. If `chooseAndPack` succeeds, call the resulting bin packing $\overline{P}$. Otherwise, `chooseAndPack` returns `null`. P_{best} is the value of $\overline{P}$ with the minimum number of bins across all guesses by `guessShelves`.

We prove that $\texttt{HGaP}_k$ is $T_k^{d-1}(1+\varepsilon)$-asymptotic-approximate by showing that for some $P^* \in \texttt{guessShelves}(\widehat{\mathcal{I}}, \delta)$, we have $|P^*| \lessapprox T_k^{d-1}\,\text{opt}(\mathcal{I})(1+\varepsilon)$ and $\texttt{chooseAndPack}(\widehat{\mathcal{I}}, P^*, \delta)$ is not `null`.

We will now precisely define *structured* bin packing and state the main theorems on $\texttt{HGaP}_k$.

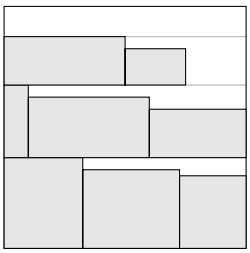


Figure 6.1: An example of shelf-based packing with 3 shelves.

6.3.1 Structured Packing

Definition 6.4 (Slicing)**.** *Slicing a 1D item i is the operation of replacing it by items i_1 and i_2 such that* $\text{size}(i_1) + \text{size}(i_2) = \text{size}(i)$.

Slicing a rectangle i using a vertical cut is the operation of replacing i by two rectangles i_1 and i_2 where $h(i) = h(i_1) = h(i_2)$ and $w(i) = w(i_1) + w(i_2)$. Slicing i using a horizontal cut is the operation of replacing i by two rectangles i_1 and i_2 where $w(i) = w(i_1) = w(i_2)$ and $h(i) = h(i_1) + h(i_2)$.

Definition 6.5 (Shelf-based δ-fractional packing)**.** *Let $\delta \in (0, 1)$ be a constant. Let K be a set of rectangular items. Items in $K_L := \{i \in K : h(i) > \delta\}$ are said to be 'δ-large' and items in $K_S := K - K_L$ are said to be 'δ-small'. A δ-fractional bin packing of K is defined to be a packing of K into bins where items in K_L can be sliced (recursively) using vertical cuts only, and items in K_S can be sliced (recursively) using both horizontal and vertical cuts.*

A shelf *is a rectangle of width 1 into which we can pack items such that the bottom edge of each item in the shelf touches the bottom edge of the shelf. A shelf can itself be packed into a bin. A δ-fractional bin packing of K is said to be* shelf-based *iff (all slices of) all items in K_L are packed into shelves, the shelves are packed into the bins, and items in K_S are packed outside the shelves (and inside the bins). Packing of items into a shelf S is said to be* tight *iff the top edge of some item (or slice) in S touches the top edge of S.*

Definition 6.6 (Structured packing)**.** *Let K be a set of rectangles and let P be a packing of empty shelves into bins. Let H be the set of heights of shelves in P (note that H is not a multiset, i.e., we only consider distinct heights of shelves). Then P is said to be* structured *for (K, δ) iff $|H| \leq \lceil 1/\delta^2 \rceil$ and each element in H is the height of some δ-large item in K.*

A shelf-based δ-fractional packing of K is said to be structured *iff the shelves in the packing are structured for (K, δ). Define $\text{sopt}_\delta(K)$ to be the number of bins in the optimal structured δ-fractional packing of K.*

$\texttt{HGaP}_k$ relies on the following key structural theorem. We formally prove it in Section 6.4.2 and give an outline of the proof here.

Theorem 6.6 (Structural theorem)**.** *Let I be a set of dD items. Let $\delta \in (0,1)$ be a constant. Then* $\text{sopt}_\delta(\texttt{round}(I)) < T_k^{d-1}(1+\delta)\,\text{opt}_{d\text{BP}}(I) + \lceil 1/\delta^2 \rceil + 1 + \delta$.

Proof outline. Let $\widehat{I} := \texttt{round}(I)$. Let $\widehat{I}_L$ and $\widehat{I}_S$ be the δ-large and δ-small items in $\widehat{I}$, respectively.

We give a simple greedy algorithm to pack $\widehat{I}_L$ into shelves. Let J be the shelves output by this algorithm. We can treat J as a 1BP instance, and $\widehat{I}_S$ as a sliceable 1D item of size $a(\widehat{I}_S)$. We prove that an optimal 1D bin packing of $J \cup \widehat{I}_S$ gives us an optimal shelf-based δ-fractional packing of $\widehat{I}$.

We use linear grouping by Lueker and Vega [26]. We partition J into linear groups of size $\lfloor \delta\,\text{size}(J) \rfloor + 1$ each. Let h_j be the height of the first 1D item in the j^{th} group. Let $J^{(\text{hi})}$ be the 1BP instance obtained by rounding up the height of each item in the j^{th} group to h_j for all j. Then $J^{(\text{hi})}$ contains at most $\lceil 1/\delta^2 \rceil$ distinct sizes, so the optimal packing of $J^{(\text{hi})} \cup \widehat{I}_S$ gives us a structured δ-fractional packing of $\widehat{I}$. Therefore, $\text{sopt}_\delta(\widehat{I}) \le \text{opt}(J^{(\text{hi})} \cup \widehat{I}_S)$. Let $J^{(\text{lo})}$ be the 1BP instance obtained by rounding down the height of each item in the j^{th} group to h_{j+1} for all j. We prove that $J^{(\text{lo})}$ contains at most $\lceil 1/\delta^2 \rceil - 1$ distinct sizes and that $\text{opt}(J^{(\text{hi})} \cup \widehat{I}_S) < \text{opt}(J^{(\text{lo})} \cup \widehat{I}_S) + \delta a(\widehat{I}_L) + (1+\delta)$.

We model packing $J^{(\text{lo})} \cup \widehat{I}_S$ as a linear program, denoted by $\text{LP}(\widehat{I})$, that has at most $\lceil 1/\delta^2 \rceil^{1/\delta}$ variables and $\lceil 1/\delta^2 \rceil$ non-trivial constraints. The optimum extreme point solution to $\text{LP}(\widehat{I})$, therefore, has at most $\lceil 1/\delta^2 \rceil$ positive entries, so $\text{opt}(J^{(\text{lo})} \cup \widehat{I}_S) \le \text{opt}(\text{LP}(\widehat{I})) + \lceil 1/\delta^2 \rceil$.

We use techniques from Caprara [18] to obtain a monotonic weighting function η from the optimal solution to the dual of $\text{LP}(\widehat{I})$. For each item $i \in I$, we define $p(i) := w(i)\eta(h(i))$ and prove that $p(I) \ge \text{opt}(\text{LP}(\widehat{I}))$. By Theorem 6.1, we get that $p(I) \le T_k^{d-1}\,\text{opt}_{d\text{BP}}(I)$ and $a(\widehat{I}_L) \le T_k^{d-1}\,\text{opt}_{d\text{BP}}(I)$. Combining the above facts gives us an upper-bound on $\text{sopt}_\delta(\widehat{I})$ in terms of $\text{opt}_{d\text{BP}}(I)$. □

6.3.2 Subroutines

6.3.2.1 `guessShelves`

The algorithm $\texttt{guessShelves}(\widehat{\mathcal{I}}, \delta)$ takes a set $\widehat{\mathcal{I}}$ of 2D itemsets and a constant $\delta \in (0,1)$ as input. We will design `guessShelves` so that it satisfies the following theorem.

Theorem 6.7. $\texttt{guessShelves}(\widehat{\mathcal{I}}, \delta)$ *returns all possible packings of empty shelves into at most* $|\widehat{\mathcal{I}}|$ *bins such that each packing is structured for* $(\text{flat}(\widehat{\mathcal{I}}), \delta)$. $\texttt{guessShelves}(\widehat{\mathcal{I}}, \delta)$ *returns at most*

$T := (N^{\lceil 1/\delta^2 \rceil} + 1)(n+1)^R$ *packings, where* $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$, $n := |\widehat{\mathcal{I}}|$, *and* $R := \binom{\lceil 1/\delta^2 \rceil + \lceil 1/\delta \rceil - 1}{\lceil 1/\delta \rceil - 1} \leq (1 + \lceil 1/\delta^2 \rceil)^{1/\delta}$. *Its running time is* $O(T)$.

`guessShelves` works by first guessing at most $\lceil 1/\delta^2 \rceil$ distinct heights of shelves. It then enumerates all configurations, i.e., different ways in which shelves can be packed into a bin. It then guesses the configurations in a bin packing of the shelves. `guessShelves` can be easily implemented using standard techniques. For the sake of completeness, we give a more precise description of `guessShelves` and prove Theorem 6.7 in Section 6.4.3.

6.3.2.2 `chooseAndPack`

`chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ takes as input a set $\widehat{\mathcal{I}}$ of 2D itemsets, a constant $\delta \in (0, 1)$, and a bin packing P of empty shelves that is structured for $(\operatorname{flat}(\widehat{\mathcal{I}}), \delta)$. It tries to pack an assortment of $\widehat{\mathcal{I}}$ into the shelves in P.

`chooseAndPack` works by rounding up the width of all δ-large items in $\widehat{\mathcal{I}}$ to a multiple of $1/n$. This would increase the number of shelves required by 1, so it adds another empty shelf. It then uses dynamic programming to pack an assortment into the shelves, such that the area of the chosen δ-small items is minimum. This is done by maintaining a dynamic programming table that keeps track of the number of itemsets considered so far and the remaining space in shelves of each type. If it is not possible to pack the items into the shelves, then `chooseAndPack` outputs `null`. In Section 6.4.4, we give the details of this algorithm and formally prove the following theorems:

Theorem 6.8. *If there exists an assortment* $\widehat{K}$ *of* $\widehat{\mathcal{I}}$ *having a structured* δ*-fractional bin packing* P*, then* `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ *does not output* ***`null`****.*

Theorem 6.9. *If the output of* `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ *is not* ***`null`****, then the output* $\overline{P}$ *is a shelf-based* δ*-fractional packing of some assortment of* $\widehat{\mathcal{I}}$ *such that* $|\overline{P}| \leq |P| + 1$ *and the distinct shelf heights in* $\overline{P}$ *are the same as that in* P*.*

Theorem 6.10. `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ *runs in* $O(Nn^{2\lceil 1/\delta^2 \rceil})$ *time. Here* $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$, $n := |\widehat{\mathcal{I}}|$.

6.3.2.3 `inflate`

For a set I of dD items, `inflate` is an algorithm that converts a shelf-based packing of `round`(I) into a packing of I having roughly the same number of bins.

For a dD item i, btype(i) (called *base type*) is defined to be a $(d-1)$-dimensional vector whose j^{th} component is $\text{type}_k(\ell_j(i))$. Roughly, $\texttt{inflate}(P)$ works as follows: It first slightly modifies the packing P so that items of different base types are in different shelves and δ-small items are no longer sliced using horizontal cuts. Then it converts each 2D shelf to a dD shelf of the same height using $\texttt{HDH-unit-pack}_k$ (a dD shelf is a cuboid where the first $d-1$ dimensions are equal to 1).

In Section 6.4.5, we formally describe `inflate` and prove the following theorem about it.

Theorem 6.11. *Let I be a set of dD items having Q distinct base types. Let P be a shelf-based δ-fractional packing of $\texttt{round}(I)$ where shelves have t distinct heights. Then $\texttt{inflate}(P)$ returns a packing of I into less than $|P|/(1-\delta)+t(Q-1)+1+\delta Q/(1-\delta)$ bins in $O(n \log n)$ time, where $n := |I|$.*

Now that we have mentioned the guarantees of all the subroutines used by $\texttt{HGaP}_k$, we can prove the correctness and running time of $\texttt{HGaP}_k$.

6.3.3 Correctness and Running Time of $\texttt{HGaP}_k$

Theorem 6.12. *The number of bins used by $\texttt{HGaP}_k(\mathcal{I}, \varepsilon)$ to pack $\mathcal{I}$ is less than*

$$T_k^{d-1}(1+\varepsilon)\,\text{opt}_{d\text{MCBP}}(\mathcal{I}) + \left\lceil \left(\frac{2}{\varepsilon}+1\right)^2 \right\rceil \left(Q+\frac{\varepsilon}{2}\right) + 3 + (Q+3)\frac{\varepsilon}{2}.$$

Here Q is the number of distinct base types in $\text{flat}(\mathcal{I})$.

Proof. Let K^* be the assortment in an optimal bin packing of $\mathcal{I}$. Let $\widehat{K}^* = \texttt{round}(K^*)$. Let P^* be the optimal structured δ-fractional bin packing of $\widehat{K}^*$. Then $|P^*| = \text{sopt}_\delta(\widehat{K}^*)$ by the definition of sopt. By Theorem 6.7, $P^* \in \texttt{guessShelves}(\widehat{\mathcal{I}}, \delta)$. Let $\overline{P}^* = \texttt{chooseAndPack}(\widehat{\mathcal{I}}, P^*, \delta)$. By Theorem 6.8, $\overline{P}^*$ is not `null`. By Theorem 6.9, P_{best} is structured for $(\text{flat}(\widehat{\mathcal{I}}), \delta)$ and $|P_{\text{best}}| \le |\overline{P}^*| \le \text{sopt}_\delta(\widehat{K}^*) + 1$.

By Theorem 6.11, we get that

$$|\,\texttt{inflate}(P_{\text{best}})| < \frac{\text{sopt}_\delta(\widehat{K}^*)}{1-\delta} + \left\lceil \frac{1}{\delta^2} \right\rceil (Q-1) + 1 + \frac{\delta Q + 1}{1-\delta}.$$

By Theorem 6.6 (structural theorem) and using $\text{opt}_{d\text{BP}}(K^*) = \text{opt}_{d\text{MCBP}}(\mathcal{I})$, we get

$$\text{sopt}_\delta(\widehat{K}^*) < T_k^{d-1}(1+\delta)\,\text{opt}_{d\text{MCBP}}(\mathcal{I}) + \lceil 1/\delta^2 \rceil + 1 + \delta.$$

Therefore, $|\texttt{inflate}(P_{\text{best}})|$ is less than

$$T_k^{d-1}\frac{1+\delta}{1-\delta}\operatorname{opt}_{d\text{MCBP}}(\mathcal{I}) + \left\lceil\frac{1}{\delta^2}\right\rceil\left(Q+\frac{\delta}{1-\delta}\right)+3+\frac{\delta(3+Q)}{1-\delta}$$
$$= T_k^{d-1}(1+\varepsilon)\operatorname{opt}_{d\text{MCBP}}(\mathcal{I}) + \left\lceil\left(\frac{2}{\varepsilon}+1\right)^2\right\rceil\left(Q+\frac{\varepsilon}{2}\right)+3+(Q+3)\frac{\varepsilon}{2}. \qquad \square$$

Theorem 6.13. *$\texttt{HGaP}_k(\mathcal{I},\varepsilon)$ runs in time $O(N^{1+\lceil 1/\delta^2\rceil}n^{R+2\lceil 1/\delta^2\rceil})$, where $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$, $n := |\widehat{\mathcal{I}}|$, $\delta := \varepsilon/(2+\varepsilon)$ and $R := \binom{\lceil 1/\delta^2\rceil+\lceil 1/\delta\rceil-1}{\lceil 1/\delta\rceil-1} \le (1+\lceil 1/\delta^2\rceil)^{1/\delta}$.*

Proof. Follows from Theorems 6.7, 6.10 and 6.11. $\square$

Section 6.4.6 gives hints on improving the running time of $\texttt{HGaP}_k$.

6.4 Details of the $\texttt{HGaP}_k$ Algorithm

This section gives details of the subroutines used by $\texttt{HGaP}_k$. It also proves the theorems claimed in Section 6.3.

6.4.1 Preliminaries

Definition 6.7. *Let I_1 and I_2 be sets of 1D items. Then I_1 is defined to be a predecessor of I_2 (denoted as $I_1 \preceq I_2$) iff there exists a one-to-one mapping $\pi : I_1 \mapsto I_2$ such that $\forall i \in I_1, i \le \pi(i)$.*

Observation 6.14. *Let $I_1 \preceq I_2$ where π is the corresponding mapping. Then we can obtain a packing of I_1 from a packing of I_2, by packing each item $i \in I_1$ in the place of $\pi(i)$. Hence,* $\operatorname{opt}(I_1) \le \operatorname{opt}(I_2)$.

Definition 6.8 (Canonical shelving)**.** *Let I be a set of rectangles. Order the items in I in non-increasing order of height (break ties arbitrarily but deterministically) and greedily pack them into tight shelves, slicing items using vertical cuts if necessary. The set of shelves thus obtained is called the* canonical shelving *of I, and is denoted by* can-shelv(I). *(The canonical shelving is unique because ties are broken deterministically.)*

See Fig. 6.2 for an example of canonical shelving.

Suppose a set I of rectangular items is packed into a set J of shelves. Then we can interpret J as a 1BP instance where the height of each shelf is the size of the corresponding 1D item.

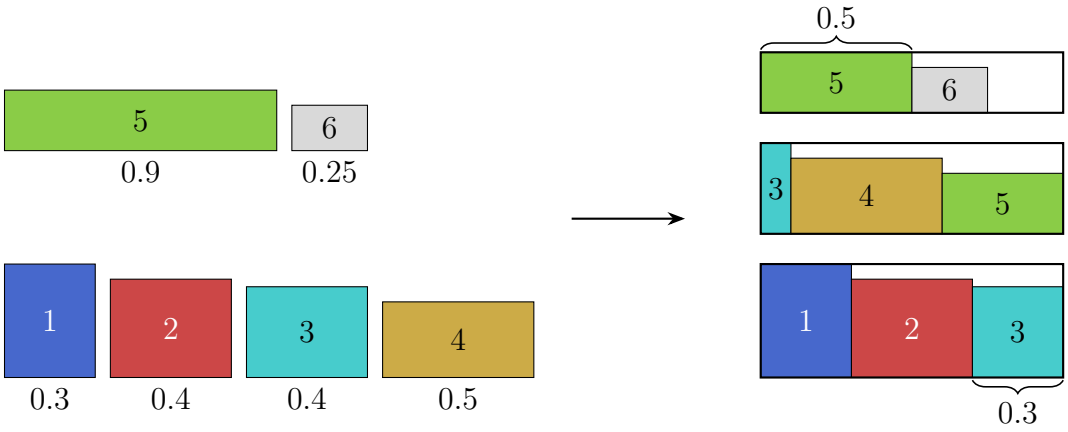


Figure 6.2: Six items and their canonical shelving into three tight shelves of width 1. The items are numbered by decreasing order of height. Each item has its width mentioned below it. Item 3 was sliced into two items of widths 0.3 and 0.1. Item 5 was sliced into two items of widths 0.4 and 0.5.

We will now prove that the canonical shelving is optimal, i.e., any shelf-based bin packing of items can be obtained by first computing the canonical shelving and then packing the shelves into bins like a 1BP instance.

Lemma 6.15. *Let I be a set of rectangles packed inside shelves J. Let $J^* := \text{can-shelv}(I)$. Then $J^* \preceq J$.*

Proof. We say that a shelf is full if the total width of items in a shelf is 1. Arrange the shelves J in non-increasing order of height, and arrange the items I in non-increasing order of height. Then try to pack I into J using the following greedy algorithm: For each item i, pack the largest possible slice of i into the first non-full shelf and pack the remaining slice (if any) in the next shelf. If this greedy algorithm succeeds, then within each shelf of J, there is a shelf of J^*, so $J^* \preceq J$. We will now prove that this greedy algorithm always succeeds.

For the sake of proof by contradiction, assume that the greedy algorithm failed, i.e., for an item (or slice) i there was a non-full shelf S but $h(i) > h(S)$. Let I' be the items (and slices) packed before i and J' be the shelves before S. Therefore, $w(I') = |J'|$.

All items in I' have height at least $h(i)$, so all shelves in J' have height at least $h(i)$. All shelves after J' have height less than $h(i)$. Therefore, J' is exactly the set of shelves of height at least $h(i)$.

In the packing P, $I' \cup \{i\}$ can only be packed into shelves of height at least $h(i)$, so $w(I') + w(i) \leq |J'|$. But this contradicts $w(I') = |J'|$. Therefore, the greedy algorithm cannot fail. □

Lemma 6.16. *Consider the inequality $x_1 + x_2 + \ldots + x_n \le s$, where for each $j \in [n]$, $x_j \in \mathbb{Z}_{\ge 0}$. Let N be the number of solutions to this inequality. Then $N = \binom{s+n}{n} \le (s+1)^n$.*

Proof. The proof of $N = \binom{s+n}{n}$ is a standard result in combinatorics.

To prove $N \le (s+1)^n$, note that we can choose each $x_j \in \{0, 1, \ldots, s\}$ independently. □

6.4.2 Structural Theorem

Let I be a set of dD items. Let $\widehat{I} := \texttt{round}(I)$. Let $\delta \in (0, 1)$ be a constant. Let $\widehat{I}_L := \{i \in \widehat{I} : h(i) > \delta\}$ and $\widehat{I}_S := \widehat{I} - \widehat{I}_L$. Let $J := \text{can-shelv}(\widehat{I}_L)$. Let $m := |J|$, i.e., J contains m shelves. We can interpret $\widehat{I}_S$ as a single sliceable 1D item of size $a(\widehat{I}_S)$.

We will show the existence of a structured δ-fractional packing of $\widehat{I}$ into at most $T_k^{d-1}(1+\delta)\,\text{opt}_{d\text{BP}}(I) + \lceil 1/\delta^2 \rceil + 1 + \delta$ bins. This would prove Theorem 6.6.

Definition 6.9 (Linear grouping [26])**.** *Arrange the 1D items J in non-increasing order of size and number them from 1 to m. Let $q := \lfloor \delta \,\text{size}(J) \rfloor + 1$. Let J_1 be the first q items, J_2 be the next q items, and so on. J_j is called the j^{th}* linear group *of J. This gives us $t := \lceil m/q \rceil$ linear groups. Note that the last group, J_t, may have less than q items.*

Let h_j be the size of the first item in J_j. Let $h_{t+1} := 0$. For $j \in [t-1]$, let $J_j^{(\text{lo})}$ be the items obtained by decreasing the height of items in J_j to h_{j+1}. For $j \in [t]$, let $J_j^{(\text{hi})}$ be the items obtained by increasing the height of items in J_j to h_j.

Let $J^{(\text{lo})} := \bigcup_{j=1}^{t-1} J_j^{(\text{lo})}$ and $J^{(\text{hi})} := \bigcup_{j=1}^{t} J_j^{(\text{hi})}$. We call $J^{(\text{lo})}$ a down-rounding of J and $J^{(\text{hi})}$ an up-rounding of J.

Lemma 6.17. $t \le \lceil 1/\delta^2 \rceil$.

Proof. Since each shelf in J has height more than δ, $\text{size}(J) > |J|\delta$.

$$t := \left\lceil \frac{|J|}{\lfloor \delta \,\text{size}(J) \rfloor + 1} \right\rceil \le \left\lceil \frac{\text{size}(J)/\delta}{\delta \,\text{size}(J)} \right\rceil = \left\lceil \frac{1}{\delta^2} \right\rceil. \qquad \square$$

Lemma 6.18. $J^{(\text{lo})} \preceq J \preceq J^{(\text{hi})} \preceq J^{(\text{lo})} \cup J_1^{(\text{hi})}$.

Proof. It is trivial to see that $J^{(\text{lo})} \preceq J \preceq J^{(\text{hi})}$. For $j \in [t-1]$, all (1D) items in both $J_j^{(\text{lo})}$ and $J_{j+1}^{(\text{hi})}$ have height h_{j+1}, and $|J_{j+1}| \le q = |J_j|$. Therefore, $J_{j+1}^{(\text{hi})} \preceq J_j^{(\text{lo})}$ and hence

$$J^{(\text{hi})} = J_1^{(\text{hi})} \cup \bigcup_{j=1}^{t-1} J_{j+1}^{(\text{hi})} \preceq J_1^{(\text{hi})} \cup \bigcup_{j=1}^{t-1} J_j^{(\text{lo})} = J_1^{(\text{hi})} \cup J^{(\text{lo})}. \qquad \square$$

Lemma 6.19. $\text{size}(J) < 1 + a(\widehat{I}_L)$.

Proof. In the canonical shelving of $\widehat{I}_L$, let S_j be the j^{th} shelf. Let $h(S_j)$ be the height of S_j. Let $a(S_j)$ be the total area of the items in S_j. Since the shelves are tight, items in S_j have height at least $h(S_{j+1})$. So, $a(S_j) \geq h(S_{j+1})$ and

$$\text{size}(J) = \sum_{j=1}^{|J|} h(S_j) \leq 1 + \sum_{j=1}^{|J|-1} h(S_{j+1}) \leq 1 + \sum_{j=1}^{|J|-1} a(S_j) < 1 + a(\widehat{I}_L). \qquad \square$$

Lemma 6.20. $\text{sopt}_\delta(\widehat{I}) < \text{opt}(J^{(\text{lo})} \cup \widehat{I}_S) + \delta a(\widehat{I}_L) + (1 + \delta)$.

Proof. By the definition of can-shelv, $\widehat{I}_L$ can be packed into J. By Lemma 6.18, $J \preceq J^{(\text{hi})}$, so $\widehat{I}_L$ can be packed into $J^{(\text{hi})}$. By Lemma 6.17, the number of distinct sizes in $J^{(\text{hi})}$ is at most $\lceil 1/\delta^2 \rceil$. So, the optimal 1D bin packing of $J^{(\text{hi})} \cup \widehat{I}_S$ will give us a structured δ-fractional bin packing of $\widehat{I}$. Hence, $\text{sopt}_\delta(\widehat{I}) \leq \text{opt}(J^{(\text{hi})} \cup \widehat{I}_S)$.

By Lemma 6.18 and Observation 6.14 we get

$$\text{opt}(J^{(\text{hi})} \cup \widehat{I}_S) \leq \text{opt}(J^{(\text{lo})} \cup J_1^{(\text{hi})} \cup \widehat{I}_S) \leq \text{opt}(J^{(\text{lo})} \cup \widehat{I}_S) + \text{opt}(J_1^{(\text{hi})}).$$

By Lemma 6.19,

$$\text{opt}(J_1^{(\text{hi})}) \leq |J_1^{(\text{hi})}| \leq q \leq 1 + \delta\,\text{size}(J) < 1 + \delta(1 + a(\widehat{I}_L)). \qquad \square$$

6.4.2.1 LP for Packing $J^{(\text{lo})} \cup \widehat{I}_S$

We will formulate an integer linear program for bin packing $J^{(\text{lo})} \cup \widehat{I}_S$.

Let $C \in \mathbb{Z}_{\geq 0}^{t-1}$ such that $h_C := \sum_{j=1}^{t-1} C_j h_{j+1} \leq 1$. Then C is called a configuration. C represents a set of 1D items that can be packed into a bin and where C_j items are from $J_j^{(\text{lo})}$. Let $\mathcal{C}$ be the set of all configurations. We can pack at most $\lceil 1/\delta \rceil - 1$ items into a bin because $h_t > \delta$. By Lemma 6.16, we get $|\mathcal{C}| \leq \binom{\lceil 1/\delta \rceil - 1 + t - 1}{t-1} \leq \lceil 1/\delta^2 \rceil^{1/\delta}$.

Let x_C be the number of bins packed according to configuration C. Bin packing $J^{(\text{lo})} \cup \widehat{I}_S$ is equivalent to finding the optimal integer solution to the following linear program, which we

denote as $\mathrm{LP}(\widehat{I})$.

$$\begin{aligned}
\min_{x\in\mathbb{R}^{|\mathcal{C}|}} \quad & \sum_{C\in\mathcal{C}} x_C \\
\text{where} \quad & \sum_{C\in\mathcal{C}} C_j x_C \geq q && \forall j \in [t-1] \\
& \sum_{C\in\mathcal{C}} (1-h_C) x_C \geq a(\widehat{I}_S) \\
& x_C \geq 0 && \forall C \in \mathcal{C}
\end{aligned}$$

Here the first set of constraints say that for each $j \in [t-1]$, all of the $q := \lfloor \delta \,\mathrm{size}(J) \rfloor + 1$ shelves $J_j^{(\mathrm{lo})}$ should be covered by the configurations in x. The second constraint says that we should be able to pack $a(\widehat{I}_S)$ into the non-shelf space in the bins.

Lemma 6.21. $\mathrm{opt}(J^{(\mathrm{lo})} \cup \widehat{I}_S) \leq \mathrm{opt}(\mathrm{LP}(\widehat{I})) + t$.

Proof. Let x^* be an optimal extreme-point solution to $\mathrm{LP}(\widehat{I})$. By rank-lemma, x^* has at most t non-zero entries. Let $\widehat{x}$ be a vector where $\widehat{x}_C := \lceil x^*_C \rceil$. Then $\widehat{x}$ is an integral solution to $\mathrm{LP}(\widehat{I})$ and $\sum_C \widehat{x}_C < t + \sum_C x^*_C = \mathrm{opt}(\mathrm{LP}(\widehat{I})) + t$. □

The dual of $\mathrm{LP}(\widehat{I})$, denoted by $\mathrm{DLP}(\widehat{I})$, is

$$\begin{aligned}
\max_{y\in\mathbb{R}^{t-1}, z\in\mathbb{R}} \quad & a(\widehat{I}_S) z + q \sum_{j=1}^{t-1} y_j \\
\text{where} \quad & \sum_{j=1}^{t-1} C_j y_j + (1-h_C) z \leq 1 \quad \forall C \in \mathcal{C} \\
& z \geq 0 \text{ and } y_j \geq 0 \quad \forall j \in [t-1]
\end{aligned}$$

6.4.2.2 Weighting Function for a Feasible Solution to $\mathrm{DLP}(\widehat{I})$

We will now see how to obtain a monotonic weighting function $\eta : [0,1] \mapsto [0,1]$ from a feasible solution to $\mathrm{DLP}(\widehat{I})$. To do this, we adapt techniques from Caprara's analysis of $\mathtt{HDH}_k$ [18]. Such a weighting function will help us upper-bound $\mathrm{opt}(\mathrm{LP}(\widehat{I}))$ in terms of $\mathrm{opt}_{d\mathrm{BP}}(I)$.

We first describe a transformation that helps us convert any feasible solution of $\mathrm{DLP}(\widehat{I})$ to a feasible solution that is *monotonic*. We then show how to obtain a weighting function from this monotonic solution.

Transformation 6.10. *Let (y,z) be a feasible solution to $\mathrm{DLP}(\widehat{I})$. Let $s \in [t-1]$. Define $y_t := 0$ and $h_{t+1} := 0$. Then change y_s to $\max(y_s, y_{s+1} + (h_{s+1} - h_{s+2})z)$.*

Lemma 6.22. *Let (y, z) be a feasible solution to* DLP$(\widehat{I})$*. Let $(\widehat{y}, z)$ be the new solution obtained by applying Transformation 6.10 with parameter $s \in [t-1]$. Then $(\widehat{y}, z)$ is feasible for* DLP$(\widehat{I})$*.*

Proof. For a configuration C, let $f(C, y, z) := C^T y + (1 - h_C)z$, where $C^T y := \sum_{j=1}^{t-1} C_j y_j$. Since (y, z) is feasible for DLP$(\widehat{I})$, $f(C, y, z) \leq 1$. As per Transformation 6.10,

$$\widehat{y}_j := \begin{cases} \max(y_s, y_{s+1} + (h_{s+1} - h_{s+2})z) & j = s \\ y_j & j \neq s \end{cases}.$$

If $y_s \geq y_{s+1} + (h_{s+1} - h_{s+2})z$, then $\widehat{y} = y$, so $(\widehat{y}, z)$ would be feasible for DLP$(\widehat{I})$. So now assume that $y_s < y_{s+1} + (h_{s+1} - h_{s+2})z$.

Let C be a configuration. Define $C_t := 0$. Let

$$\widehat{C}_j := \begin{cases} 0 & j = s \\ C_s + C_{s+1} & j = s+1 \\ C_j & \text{otherwise} \end{cases}.$$

Then, $C^T\widehat{y} - \widehat{C}^T y = C_s\widehat{y}_s + C_{s+1}\widehat{y}_{s+1} - \widehat{C}_s y_s - \widehat{C}_{s+1} y_{s+1} = C_s(h_{s+1} - h_{s+2})z$.

Also, $h_{\widehat{C}} - h_C = \widehat{C}_s h_{s+1} + \widehat{C}_{s+1} h_{s+2} - C_s h_{s+1} - C_{s+1} h_{s+2} = -C_s(h_{s+1} - h_{s+2})$.

Since $h_{\widehat{C}} \leq h_C \leq 1$, $\widehat{C}$ is a configuration.

$$\begin{aligned} f(C, \widehat{y}, z) &= C^T\widehat{y} + (1 - h_C)z \\ &= (\widehat{C}^T y + C_s(h_{s+1} - h_{s+2})z) + (1 - h_{\widehat{C}} - C_s(h_{s+1} - h_{s+2}))z \\ &= f(\widehat{C}, y, z) \leq 1. \end{aligned}$$

Therefore, $(\widehat{y}, z)$ is feasible for DLP$(\widehat{I})$. □

Definition 6.11. *Let (y, z) be a feasible solution to* DLP$(\widehat{I})$*. Let*

$$\widehat{y}_j := \begin{cases} \max(y_{t-1}, zh_t) & j = t-1 \\ \max(y_j, \widehat{y}_{j+1} + (h_{j+1} - h_{j+2})z) & j < t-1 \end{cases}.$$

Then $(\widehat{y}, z)$ is called the monotonization of (y, z).

Lemma 6.23. *Let (y, z) be a feasible solution to* DLP$(\widehat{I})$*. Let $(\widehat{y}, z)$ be the monotonization of (y, z). Then $(\widehat{y}, z)$ is a feasible solution to* DLP$(\widehat{I})$*.*

Proof. $(\widehat{y}, z)$ can be obtained by multiple applications of Transformation 6.10: first with $s = t-1$, then with $s = t-2$, and so on till $s = 1$. Then by Lemma 6.22, $(\widehat{y}, z)$ is feasible for $\mathrm{DLP}(\widehat{I})$. □

Let (y^*, z^*) be an optimal solution to $\mathrm{DLP}(\widehat{I})$. Let $(\widehat{y}, z^*)$ be the monotonization of (y^*, z^*). Then define the function $\eta : [0,1] \mapsto [0,1]$ as

$$\eta(x) := \begin{cases} \widehat{y}_1 & \text{if } x \in [h_2, 1] \\ \widehat{y}_j & \text{if } x \in [h_{j+1}, h_j), \text{ for } 2 \le j \le t-1 \\ xz^* & \text{if } x < h_t \end{cases}.$$

Lemma 6.24. *η is a monotonic weighting function.*

Proof. η is monotonic by the definition of monotonization.

Let $X \subseteq (0,1]$ be a finite set such that $\mathrm{sum}(X) \le 1$. Let $X_0 := X \cap [0, h_t)$, let $X_1 := X \cap [h_2, 1]$ and for $2 \le j \le t-1$, let $X_j := X \cap [h_{j+1}, h_j)$. Let $C \in \mathbb{Z}_{\ge 0}^{t-1}$ such that $C_j := |X_j|$. Let $h_C := \sum_{j=1}^{t-1} C_j h_{j+1}$.

$$\begin{aligned} 1 \ge \mathrm{sum}(X) &= \mathrm{sum}(X_0) + \sum_{j=1}^{t-1} \mathrm{sum}(X_j) \\ &\ge \mathrm{sum}(X_0) + \sum_{j=1}^{t-1} C_j h_{j+1} && \text{(for } j \ge 1\text{, each element in } X_j \text{ is at least } h_{j+1}\text{)} \\ &= \mathrm{sum}(X_0) + h_C. \end{aligned}$$

Since $h_C \le 1 - \mathrm{sum}(X_0) \le 1$, C is a configuration. Therefore,

$$\begin{aligned} \sum_{x \in X} \eta(x) = \sum_{j=0}^{t-1} \sum_{x \in X_j} \eta(x) &= z^* \,\mathrm{sum}(X_0) + \sum_{j=1}^{t-1} C_j \widehat{y}_j && \text{(by definition of } \eta\text{)} \\ &\le (1 - h_C) z^* + C^T \widehat{y} && (h_C \le 1 - \mathrm{sum}(X_0)) \\ &\le 1. && (C \text{ is a configuration and } (\widehat{y}, z^*) \text{ is feasible for } \mathrm{DLP}(\widehat{I}) \text{ by Lemma 6.23}) \end{aligned}$$

□

Lemma 6.25. *For $i \in I$, let $p(i) := \eta(h(i))w(i)$. Then $\mathrm{opt}(\mathrm{LP}(\widehat{I})) \le p(I) \le T_k^{d-1} \,\mathrm{opt}_{d\mathrm{BP}}(I)$.*

Proof. Let (y^*, z^*) be an optimal solution to $\mathrm{DLP}(\widehat{I})$. Let $(\widehat{y}, z^*)$ be the monotonization of (y^*, z^*).

In the canonical shelving of I, suppose a rectangular item i (or a slice thereof) lies in shelf S where $S \in J_j$. Then $h(i) \in [h_{j+1}, h_j]$, where $h_{t+1} := 0$. This is because shelves in $J := \text{can-shelv}(\widehat{I})$ are tight. If $j = 1$, then $\eta(h(i)) = \widehat{y}_1 \geq y_1^*$. If $2 \leq j \leq t-1$, then $\eta(h(i)) \in \{\widehat{y}_{j-1}, \widehat{y}_j\} \geq \widehat{y}_j \geq y_j^*$. We are guaranteed that for $j \in [t-1]$, and each shelf $S \in J_j$, $w(S) = 1$.

$$\begin{aligned}
p(I) &= \sum_{j=1}^{t} \sum_{S \in J_j} \sum_{i \in S} \eta(h(i)) w(i) + \sum_{i \in \widehat{I}_S} \eta(h(i)) w(i) && \text{(by definition of } p\text{)} \\
&\geq \sum_{j=1}^{t-1} \sum_{S \in J_j} \sum_{i \in S} y_j^* w(i) + \sum_{i \in \widehat{I}_S} (h(i) z^*) w(i) && \text{(by definition of } \eta\text{)} \\
&= \sum_{j=1}^{t-1} y_j^* q + a(\widehat{I}_S) z^* && \text{(since } w(J_j) = q \text{ for } j \in [t-1]\text{)} \\
&= \operatorname{opt}(\mathrm{DLP}(\widehat{I})). && \text{(}(y^*, z^*) \text{ is optimal for } \mathrm{DLP}(\widehat{I})\text{)}
\end{aligned}$$

By strong duality of linear programs, $\operatorname{opt}(\mathrm{LP}(\widehat{I})) = \operatorname{opt}(\mathrm{DLP}(\widehat{I})) \leq p(I)$.

Since η and H_k are weighting functions (by Lemma 6.24), we get that $p(I) \leq T_k^{d-1} \operatorname{opt}_{d\mathrm{BP}}(I)$ by Theorem 6.1. □

Theorem 6.6 (Structural theorem)**.** *Let I be a set of dD items. Let $\delta \in (0, 1)$ be a constant. Then* $\operatorname{sopt}_\delta(\texttt{round}(I)) < T_k^{d-1}(1+\delta) \operatorname{opt}_{d\mathrm{BP}}(I) + \lceil 1/\delta^2 \rceil + 1 + \delta$.

Proof.

$$a(\widehat{I}_L) \leq a(\widehat{I}) = \sum_{i \in I} \left(\ell_d(i) \prod_{j=1}^{d-1} f_k(\ell_j(i)) \right) \leq T_k^{d-1} \operatorname{opt}_{d\mathrm{BP}}(I). \qquad \text{(by Theorem 6.1)}$$

$$\begin{aligned}
\operatorname{sopt}_\delta(\widehat{I}) &< \operatorname{opt}(J^{(\mathrm{lo})} \cup \widehat{I}_S) + \delta a(\widehat{I}_L) + (1+\delta) && \text{(by Lemma 6.20)} \\
&\leq \operatorname{opt}(\mathrm{LP}(\widehat{I})) + \left\lceil \frac{1}{\delta^2} \right\rceil + \delta T_k^{d-1} \operatorname{opt}_{d\mathrm{BP}}(I) + (1+\delta) && \text{(by Lemmas 6.17 and 6.21)} \\
&\leq T_k^{d-1}(1+\delta) \operatorname{opt}_{d\mathrm{BP}}(I) + \left\lceil \frac{1}{\delta^2} \right\rceil + 1 + \delta. && \text{(by Lemma 6.25)}
\end{aligned}$$

□

6.4.3 Guessing Shelves and Bins

We want `guessShelves`$(\widehat{\mathcal{I}}, \delta)$ to return all possible packings of empty shelves into at most $n := |\widehat{\mathcal{I}}|$ bins such that each packing is structured for $(\text{flat}(\widehat{\mathcal{I}}), \delta)$.

Let $H = \{h(i) : i \in \text{flat}(\widehat{\mathcal{I}})\}$. Let $N := |\text{flat}(\widehat{\mathcal{I}})|$. `guessShelves`$(\widehat{\mathcal{I}}, \delta)$ starts by picking the distinct heights of shelves by iterating over all subsets of H of size at most $\lceil 1/\delta^2 \rceil$. The number of such subsets is at most $N^{\lceil 1/\delta^2 \rceil} + 1$. Let $\widetilde{H} := \{h_1, h_2, \ldots, h_t\}$ be one such guess, where $t \leq \lceil 1/\delta^2 \rceil$. Without loss of generality, assume $h_1 > h_2 > \ldots > h_t > \delta$.

Next, `guessShelves` needs to decide the number of shelves of each height and a packing of those shelves into bins. Let $C \in \mathbb{Z}^t_{\geq 0}$ such that $h_C := \sum_{j=1}^{t-1} C_j h_j \leq 1$. Then C is called a configuration. C represents a set of shelves that can be packed into a bin and where C_j shelves have height h_j. Let $\mathcal{C}$ be the set of all configurations. We can pack at most $\lceil 1/\delta \rceil - 1$ items into a bin because $h_t > \delta$. By Lemma 6.16, we get

$$|\mathcal{C}| \leq \binom{\lceil 1/\delta \rceil - 1 + t}{t} \leq \binom{\lceil 1/\delta \rceil - 1 + \lceil 1/\delta^2 \rceil}{\lceil 1/\delta \rceil - 1} \leq \left(\left\lceil \frac{1}{\delta^2} \right\rceil + 1 \right)^{1/\delta}.$$

There can be at most n bins, and `guessShelves` has to decide the configuration of each bin. By Lemma 6.16, the number of ways of doing this is at most $\binom{|\mathcal{C}|+n}{|\mathcal{C}|} \leq (n+1)^{|\mathcal{C}|}$. Therefore, `guessShelves` computes all configurations and then iterates over all $\binom{|\mathcal{C}|+n}{|\mathcal{C}|}$ combinations of these configs.

This completes the description of `guessShelves` and proves Theorem 6.7.

6.4.4 `chooseAndPack`

`chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ takes as input a set $\widehat{\mathcal{I}}$ of 2D itemsets, a packing P of empty shelves into bins and constant $\delta \in (0, 1)$. It tries to pack $\widehat{\mathcal{I}}$ into P and one additional shelf. Before we design `chooseAndPack`, let us see how to handle a special case, i.e., where $\widehat{\mathcal{I}}$ is *simple*.

Definition 6.12. *A set $\widehat{\mathcal{I}}$ of 2D itemsets is* δ-simple *iff the width of each δ-large item in* $\text{flat}(\widehat{\mathcal{I}})$ *is a multiple of* $1/|\widehat{\mathcal{I}}|$.

Let P be a bin packing of empty shelves. Let $h_1 > h_2 > \ldots > h_t$ be the distinct heights of the shelves in P, where $h_t > \delta$. We will use dynamic programming to either pack a simple instance $\widehat{\mathcal{I}}$ into P or claim that no assortment of $\widehat{\mathcal{I}}$ can be packed into P. Call this algorithm `simpleChooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$.

Let $\widehat{\mathcal{I}} := \{I_1, I_2, \ldots, I_n\}$. For $j \in \{0, 1, \ldots, n\}$, define $\widehat{\mathcal{I}}_j := \{I_1, I_2, \ldots, I_j\}$, i.e., $\widehat{\mathcal{I}}_j$ contains the first j itemsets from $\widehat{\mathcal{I}}$. Let $\vec{u} := [u_1, u_2, \ldots, u_t] \in \{0, 1, \ldots, n^2\}^t$ be a vector. Let $\Phi(j, \vec{u})$ be

the set of all assortments of $\widehat{\mathcal{I}}_j$ that can be packed into t shelves, where the r^{th} shelf has height h_r and width u_r/n. For a set K of items, define smallArea(K) as the total area of δ-small items in K. Define $g(j, \vec{u}) := \min_{K \in \Phi(j,\vec{u})} \text{smallArea}(K)$. If $\Phi(j, \vec{u}) = \emptyset$, then we let $g(j, \vec{u}) = \infty$.

We will show how to compute $g(j, \vec{u})$ for all $j \in \{0, 1, \ldots, n\}$ and all $\vec{u} \in \{0, 1, \ldots, n^2\}^t$ using dynamic programming. Let there be n_r shelves in P having height h_r. Then for $j = n$ and $u_r = n_r n$, $\widehat{\mathcal{I}}$ can be packed into P iff $g(j, \vec{u})$ is at most the area of non-shelf space in P.

Note that in any solution K corresponding to $g(j, \vec{u})$, we can assume without loss of generality that the item i from $K \cap I_j$ is placed in the smallest shelves possible. This is because we can always swap i with the slices of items in those shelves. This observation gives us the following recurrence relation for $g(j, \vec{u})$:

$$g(j, \vec{u}) = \begin{cases} \infty & \text{if } u_j < 0 \text{ for some } j \in [t] \\ 0 & \text{if } n = 0 \text{ and } u_j \geq 0 \text{ for all } j \in [t] \\ \min_{i \in I_j} \begin{pmatrix} \text{smallArea}(\{i\}) \\ + g(j-1, \text{reduce}(\vec{u}, i)) \end{pmatrix} & \text{if } n > 0 \text{ and } u_j \geq 0 \text{ for all } j \in [t] \end{cases} \tag{6.1}$$

Here reduce($\vec{u}, i$) is a vector obtained as follows: If i is δ-small, then reduce($\vec{u}, i$) := $\vec{u}$. Otherwise, initialize x to $w(i)$. Let p_i be the largest integer r such that $h(i) \leq h_r$. For r varying from p_i to 2, subtract $\min(x, u_j)$ from x and u_j. Then subtract x from u_1. The new value of $\vec{u}$ is defined to be the output of reduce($\vec{u}, i$).

The recurrence relation allows us to compute $g(j, \vec{u})$ for all j and $\vec{u}$ using dynamic programming in time $O(Nn^{2t})$ time, where $N := |\text{flat}(\widehat{\mathcal{I}})|$. With a bit more work, we can also compute the corresponding assortment K, if one exists. Therefore, `simpleChooseAndPack`($\widehat{\mathcal{I}}, P, \delta$) computes a packing of $\widehat{\mathcal{I}}$ into P if one exists, or returns `null` if no assortment of $\widehat{\mathcal{I}}$ can be packed into P.

Now we will look at the case where $\widehat{\mathcal{I}}$ is not δ-simple. Let $\widehat{\mathcal{I}}'$ be the instance obtained by rounding up the width of each δ-large item in $\widehat{\mathcal{I}}$ to a multiple of $1/n$, where $n := |\widehat{\mathcal{I}}|$. Let $\overline{P}$ be the bin packing obtained by adding another bin to P containing a single shelf of height h_1. `chooseAndPack`($\widehat{\mathcal{I}}, P, \delta$) computes $\widehat{\mathcal{I}}'$ and $\overline{P}$, and then returns the output of `simpleChooseAndPack`($\widehat{\mathcal{I}}', \overline{P}, \delta$).

Theorem 6.9. *If the output of* `chooseAndPack`($\widehat{\mathcal{I}}, P, \delta$) *is not* `null`*, then the output* $\overline{P}$ *is a shelf-based* δ*-fractional packing of some assortment of* $\widehat{\mathcal{I}}$ *such that* $|\overline{P}| \leq |P| + 1$ *and the distinct shelf heights in* $\overline{P}$ *are the same as that in* P*.*

Proof. Follows from the definition of `simpleChooseAndPack`. □

Theorem 6.8. *If there exists an assortment $\widehat{K}$ of $\widehat{\mathcal{I}}$ having a structured δ-fractional bin packing P, then* `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ *does not output* `null`.

Proof. Let $\widehat{K}'$ be the items obtained by rounding up the width of each item in $\widehat{K}$ to a multiple of $1/n$. Then $\widehat{K}'$ is an assortment of $\widehat{\mathcal{I}}'$. We will show that $\widehat{K}'$ fits into $\overline{P}$, so `simpleChooseAndPack`$(\widehat{\mathcal{I}}', \overline{P}, \delta)$ will not output `null`.

Slice each item $i \in \widehat{K}'$ into two pieces using a vertical cut such that one piece has width equal to the original width of i in $\widehat{K}$, and the other piece has width less than $1/n$. This splits $\widehat{K}'$ into sets $\widehat{K}$ and T. T contains at most n items, each of width less than $1/n$. Therefore, we can pack $\widehat{K}$ into P and we can pack T into the newly-created shelf of height h_1. Therefore, $\widehat{K}'$ can be packed into $\overline{P}$, so `simpleChooseAndPack`$(\widehat{\mathcal{I}}', \overline{P}, \delta)$ won't output `null`. □

Theorem 6.10. `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ *runs in* $O(Nn^{2\lceil 1/\delta^2 \rceil})$ *time. Here* $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$, $n := |\widehat{\mathcal{I}}|$.

Proof. The running time of `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ is dominated by computing $g(j, \vec{u})$ for all j and $\vec{u}$, which takes $O(Nn^{2t})$ time. Since P is structured for $(\widehat{\mathcal{I}}, \delta)$, the number of distinct shelves in P, which is t, is at most $\lceil 1/\delta^2 \rceil$. □

6.4.5 `inflate`

Let I be a set of dD items. Let P be a shelf-based δ-fractional bin packing of $\widehat{I} :=$ `round`(I) into m bins. Let there be t distinct heights of shelves in P: $h_1 > h_2 > \ldots > h_t > \delta$. We want to design an algorithm `inflate`(P) that returns a packing of I into approximately $|P|$ bins.

Define $\widehat{I}_L := \{i \in \widehat{I} : h(i) > \delta\}$ and $\widehat{I}_S := \widehat{I} - \widehat{I}_L$. Let there be Q distinct base types in I (so $Q \leq k^{d-1}$).

6.4.5.1 Separating Base Types

We will now impose an additional constraint over P: items in each shelf must have the same btype. This will be helpful later, when we will try to compute a packing of dD items I.

Separating base types of $\widehat{I}_S$ is easy, since we can slice them in both directions. An analogy is to think of a mixture of multiple immiscible liquids of different densities settling into equilibrium.

Let there be n_j shelves of height h_j. Let $\widehat{I}_j$ be the items packed into shelves of height h_j. Therefore, $w(\widehat{I}_j) \leq n_j$. Let $\widehat{I}_{j,q} \subseteq \widehat{I}_j$ be the items of base type $q \in [Q]$.

For each q, pack $\widehat{I}_{j,q}$ into $\lceil w(\widehat{I}_{j,q}) \rceil$ shelves of height h_j (slicing items if needed). For these newly-created shelves, define the btype of the shelf to be the btype of the items in it. Let the

number of newly-created shelves of height h_j be n'_j. Then

$$n'_j = \sum_{q=1}^{Q} \lceil w(\widehat{I}_{j,q}) \rceil < \sum_{q=1}^{Q} w(\widehat{I}_{j,q}) + Q \leq n_j + Q \implies n'_j \leq n_j + Q - 1.$$

n_j of these shelves can be packed into existing bins in place of the old shelves. The remaining $n'_j - n_j \leq Q - 1$ shelves can be packed on the base of new bins.

Therefore, by using at most $t(Q-1)$ new bins, we can ensure that for every shelf, all items in that shelf have the same btype. These new bins don't contain any items from $\widehat{I}_S$. Call this new bin packing P'.

6.4.5.2 Forbidding Horizontal Slicing

We will now use P' to compute a shelf-based bin packing P'' of $\widehat{I}$ where items in $\widehat{I}$ can be sliced using vertical cuts only.

Let $\widehat{I}_{q,S}$ be the items in $\widehat{I}_S$ of base type q. Pack items $\widehat{I}_{q,S}$ into shelves using can-shelv. Suppose can-shelv used m_q shelves to pack $\widehat{I}_{q,S}$. For $j \in [m_q]$, let $h_{q,j}$ be the height of the j^{th} shelf. Let $H_q := \sum_{j=1}^{m_q} h_{q,j}$ and $H := \sum_{q=1}^{Q} H_q$. Since for $j \in [m_q - 1]$, all items in the j^{th} shelf have height at least $h_{q,j+1}$,

$$a(\widehat{I}_{q,S}) > \sum_{j=1}^{m_q - 1} h_{q,j+1} \geq H_q - h_{q,1} \geq H_q - \delta.$$

Therefore, $H < a(\widehat{I}_S) + Q\delta$. Let $\widehat{J}_S$ be the set of these newly-created shelves.

Use Next-Fit to pack $\widehat{J}_S$ into the space used by $\widehat{I}_S$ in P'. $\widehat{I}_S$ uses at most m bins in P' (recall that $m := |P|$). A height of less than δ will remain unpacked in each of those bins. The total height occupied by $\widehat{I}_S$ in P' is $a(\widehat{I}_S)$. Therefore, Next-Fit will pack a height of more than $a(\widehat{I}_S) - \delta m$.

Some shelves in $\widehat{J}_S$ may still be unpacked. Their total height will be less than $H - (a(\widehat{I}_S) - \delta m) < \delta(Q + m)$. We will pack these shelves into new bins using Next-Fit. The number of new bins used is at most $\lceil \delta(Q+m)/(1-\delta) \rceil$. Call this bin packing P''. The number of bins in P'' is at most $m' := m + t(Q-1) + \lceil \delta(Q+m)/(1-\delta) \rceil$.

6.4.5.3 Shelf-Based dD packing

We will now show how to convert the packing P'' of $\widehat{I}$ that uses m' bins into a packing of I that uses m' dD bins.

First, we repack the items into the shelves. For each $q \in [Q]$, let $\widehat{J}_q$ be the set of shelves in P'' of btype q. Let $\widehat{I}^{[q]}$ be the items packed into $\widehat{J}_q$. Compute $\widehat{J}^*_q := \text{can-shelv}(\widehat{I}^{[q]})$ and pack the shelves $\widehat{J}^*_q$ into $\widehat{J}_q$. This is possible by Lemma 6.15.

This repacking gives us an ordering of shelves in $\widehat{J}_q$. Number the shelves from 1 onwards. All items have at most 2 slices. If an item has 2 slices, and one slice is packed into shelf number p, then the other slice is packed into shelf number $p+1$. The slice in shelf p is called the leading slice. Every shelf has at most one leading slice.

Let S_j be the j^{th} shelf of $\widehat{J}_q$. Let R_j be the set of unsliced items in S_j and the item whose leading slice is in S_j. Order the items in R_j arbitrarily, except that the sliced item, if any, should be last. Then $w(R_j - \text{last}(R_j)) < 1$. So, we can use $\texttt{HDH-unit-pack}^{[q]}_k(R_j)$ to pack R_j into a $(d-1)$D bin. This $(d-1)$D bin gives us a dD shelf whose height is the same as that of S_j. On repeating this process for all shelves in $\widehat{J}_q$ and for all $q \in [Q]$, we get a packing of I into shelves. Since each dD shelf corresponds to a shelf in P'' of the same height, we can pack these dD shelves into bins in the same way as P''. This gives us a bin packing of I into m' bins.

6.4.5.4 The Algorithm

Sections 6.4.5.1, 6.4.5.2 and 6.4.5.3 describe how to convert a shelf-based δ-fractional packing P of $\widehat{I}$ having t distinct shelf heights into a shelf-based dD bin packing of I. We call this conversion algorithm `inflate`.

It is easy to see that the time taken by `inflate` is $O(|I| \log |I|)$.

If P has m bins, then the number of bins in $\texttt{inflate}(P)$ is at most

$$m + t(Q-1) + \left\lceil \frac{\delta(Q+m)}{1-\delta} \right\rceil < \frac{m}{1-\delta} + t(Q-1) + 1 + \frac{\delta Q}{1-\delta}.$$

This proves Theorem 6.11.

6.4.6 Improving Running Time

For simplicity of presentation, we left out some opportunities for improving the running time of $\texttt{HGaP}_k$. Here we briefly describe a way of speeding up $\texttt{HGaP}_k$ which reduces its running time from $O(N^{1+\lceil 1/\delta^2 \rceil} n^{R+2\lceil 1/\delta^2 \rceil})$ to $O(N^{1+\lceil 1/\delta^2 \rceil} n^{2\lceil 1/\delta^2 \rceil})$. Here $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$, $n := |\widehat{\mathcal{I}}|$, $\delta := \varepsilon/(2+\varepsilon)$ and $R := \binom{\lceil 1/\delta^2 \rceil + \lceil 1/\delta \rceil - 1}{\lceil 1/\delta \rceil - 1} \leq (1 + \lceil 1/\delta^2 \rceil)^{1/\delta}$.

In `guessShelves`, we guess two things simultaneously: (i) the number and heights of shelves (ii) the packing of the shelves into bins. This allows us to guess the optimal structured δ-fractional packing. But we don't need that; an approximate structured packing would do.

Therefore, we only guess the number and heights of shelves. We guess at most $N^{\lceil 1/\delta^2 \rceil} + 1$ distinct heights of shelves, and by Lemma 6.16, we guess at most $(n+1)^{\lceil 1/\delta^2 \rceil}$ vectors of shelf-height frequencies. Then we can use Lueker and De La Vega's $O(n \log n)$-time APTAS for 1BP [26] to pack the shelves into bins.

Also, once we guess the distinct heights of shelves, we don't need to run `chooseAndPack` afresh for every packing of empty shelves. We can reuse the dynamic programming table.

The running time is, therefore,

$$O\left(N^{\lceil 1/\delta^2 \rceil}\left(n^{\lceil 1/\delta^2 \rceil} n \log n + N n^{2\lceil 1/\delta^2 \rceil}\right)\right) = O(N^{1+\lceil 1/\delta^2 \rceil} n^{2\lceil 1/\delta^2 \rceil}).$$

6.5 HDH-unit-pack$_k$

This section gives a precise description of HDH-unit-pack$_k$ (see Section 6.1.3) and proves its correctness.

6.5.1 Shelf-Based Packing

A packing of 2D items in a bin (or strip) is said to be *shelf-based* iff the bin can be decomposed into regions, called shelves, using horizontal cuts, and the bottom edge of each item touches the bottom edge of some shelf. See Fig. 6.3 for an example. Next-Fit Decreasing Height (NFDH) and First-Fit Decreasing Height (FFDH) [24] are well-known shelf-based algorithms for 2BP and 2SP.

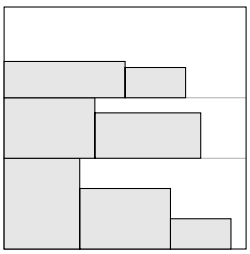


Figure 6.3: An example of shelf-based packing for $d = 2$ with 3 shelves.

The definition of shelf-based packing can be extended to dD for $d \geq 1$. For $d = 1$, every packing is said to be a shelf-based packing. For $d \geq 2$, for a dD cuboid, there are two faces of the cuboid that are perpendicular to the d^{th} dimension. The face with the smaller d^{th} coordinate is called the *base* of the cuboid. A packing of dD items into a bin is shelf-based iff the dD bin can

be split into dD shelves using hyperplanes perpendicular to the d^{th} dimension, and the base of each item is placed on the base of some shelf.

A packing of dD items into a bin is *recursive-shelf-based* iff the packing is shelf-based and the packing of the bases of items on the base of each shelf is a $(d-1)$D recursive-shelf-based bin packing. (For $d = 1$, every packing is said to be recursive-shelf-based.)

This helps us reduce dBP to $(d-1)$BP, $(d-1)$BP to $(d-2)$BP, and so on. The algorithm $\texttt{HDH}_k$ by Caprara [18] outputs a recursive-shelf-based packing by using this strategy.

6.5.2 Description and Analysis of $\texttt{HDH-unit-pack}_k$

For a dD item i, define $i^{(j)}$ as the jD item obtained by ignoring all dimensions of i other than the first j. For a set I of dD items, let $I^{(j)} := \{i^{(j)} : i \in I\}$.

$\texttt{HDH-unit-pack}_k$ takes as input a set I of dD items, where all items in I have the same type vector and $\text{vol}(f_k(I - \{\text{last}(I)\})) < 1$. $\texttt{HDH-unit-pack}_k(I)$ works recursively on d. When $d = 1$, it simply returns I. When $d > 1$, it first sorts I in decreasing order of d^{th} dimension if $\text{type}_k(\ell_d(i)) = k$ for each item $i \in I$. It then repeatedly picks the smallest prefix J of I such that $\text{vol}(f_k(J^{(d-1)})) \geq 1$, and packs J into a dD shelf. It packs all those shelves into a dD bin and returns that packing. See Algorithm 13 for a more precise description.

Define $\widetilde{f_k}(i)$ to be the cuboid $\tilde{\imath}$ where $\ell_j(\tilde{\imath}) := f_k(\ell_j(i))$ for $j \in [d-1]$ and $\ell_d(\tilde{\imath}) := \ell_d(i)$. Define $\widetilde{f_k}(I) := \{\widetilde{f_k}(i) : i \in I\}$.

Theorem 6.26 (Correctness)**.** *For a set I of dD cuboidal items, if* $\text{vol}(f_k(I - \{\text{last}(I)\})) < 1$, *then* $\texttt{HDH-unit-pack}_k(I)$ *can pack I in a dD bin.*

Proof. Let us prove this by induction on d. Let $\mathcal{P}(d)$ be this proposition: For every sequence I of dD items, if $\text{vol}(f_k(I)) < 1 + \text{vol}(f_k(\text{last}(I)))$, then $\texttt{HDH-unit-pack}_k(I)$ can pack I into a dD bin.

Base case: Let I be a sequence of 1D items such that $\text{vol}(f_k(I)) < 1 + \text{vol}(f_k(\text{last}(I)))$. Suppose $t_1 \neq k$. Then for all $i \in I, \text{vol}(i) \leq 1/t_1 = \text{vol}(f_k(i))$. Therefore,

$$\begin{aligned}
&\text{vol}(f_k(I)) < 1 + \text{vol}(f_k(\text{last}(I))) \\
&\implies \frac{|I|}{t_1} < 1 + \frac{1}{t_1} \\
&\implies |I| < t_1 + 1 \implies |I| \leq t_1 \\
&\implies \text{vol}(I) \leq \frac{|I|}{t_1} \leq 1.
\end{aligned}$$

Algorithm 13 $\texttt{HDH-unit-pack}_k^{[t]}(I)$: For any $d \geq 1$, returns a recursive-shelf-based packing of I into a dD bin, where I is a sequence of dD cuboidal items and $\text{vol}(f_k(I - \{\text{last}(I)\})) < 1$. Here $\text{last}(I)$ is the last item in sequence I. Also, all items in I have the same type t, i.e., $\forall i \in I, \text{type}(i) = t$.

1: **if** d == 1 **then** *// when items are 1D*
2: **return** I. *// Theorem 6.26 proves that they fit in a bin.*
3: **end if**
4: **if** t_d == k **then** *// when length in d^{th} dimension is small*
5: Sort I in decreasing order of d^{th} dimension.
6: **end if** *// otherwise don't disturb ordering of items.*
7: Let P be an empty list.
8: **while** $|I| > 0$ **do**
9: Find J, the smallest prefix of I such that $J = I$ or $\text{vol}(f_k(J^{(d-1)})) \geq 1$.
10: Let t' be a $(d-1)$-dimensional vector obtained by removing the d^{th} entry from t.
11: $S = \texttt{HDH-unit-pack}_k^{[t']}(J^{(d-1)})$ *// S is a dD shelf containing items J.*
12: Append S to the list P.
13: Remove J from I.
14: **end while**
15: Return the shelf packing P. *// Theorem 6.26 proves that the sum of heights of shelves doesn't exceed 1, so this is a valid packing.*

Since $\text{vol}(I) \leq 1$, I fits in a bin.

Suppose $t_1 = k$. Then for all $i \in I, \text{vol}(f_k(i)) = \frac{k}{k-2}\,\text{vol}(i)$. Therefore,

$$\begin{aligned}
\text{vol}(I) &= \frac{k-2}{k}\,\text{vol}(f_k(I)) \\
&< \frac{k-2}{k}\,(1 + \text{vol}(f_k(\text{last}(I)))) \\
&= \frac{k-2}{k} + \text{vol}(\text{last}(I)) \\
&\leq \frac{k-2}{k} + \frac{1}{k} < 1.
\end{aligned}$$

Since $\text{vol}(I) \leq 1$, I fits in a bin. Therefore, $\mathcal{P}(1)$ holds.

Inductive step:

Let $d \geq 2$ and assume $\mathcal{P}(d-1)$ holds. Let I be a sequence of dD items such that $\text{vol}(f_k(I)) < 1 + \text{vol}(f_k(\text{last}(I)))$. $\mathcal{P}(d-1)$ implies that $\texttt{HDH-unit-pack}_k(I)$ doesn't fail at line 11. Let $s := \text{last}(I)$.

For $i \in I$, define $w(i) := \prod_{j=1}^{d-1} f_k(\ell_j(i))$ and for $X \subseteq I$, define $w(X) := \sum_{i \in X} w(i)$. Let there be p shelves in the list P. Let S_j be the j^{th} shelf that was added to P. Given the way

each prefix is chosen in line 9,

$$\forall j \leq p-1, w(S_j) \geq 1 \tag{6.2}$$

Define $\ell_d(S_j) := \max_{i \in S_j} \ell_d(i)$ to be the height of shelf S_j. Let H be the total height of the shelves, i.e. $H := \sum_{j=1}^{p} \ell_d(S_j)$. Then we need to prove that the shelves fit in the bin, i.e. $H \leq 1$.

Case 1: Suppose $t_d \neq k$.
Then $\forall i \in I, \ell_d(i) \leq 1/t_d = f_k(\ell_d(i))$. Therefore,

$$1 > \text{vol}(f_k(I-s)) = \frac{w(I-s)}{t_d} \implies w(I-s) < t_d.$$

Since ordering of items is not disturbed, $s \in S_p$. Therefore,

$$\begin{aligned} t_d > w(I-s) &= \sum_{j=1}^{p-1} w(S_j) + w(S_p - s) \geq p-1 && \text{(by (6.2))} \\ \implies p < t_d + 1 &\implies p \leq t_d \\ \implies H &= \sum_{j=1}^{p} \ell_d(S_j) \leq \frac{p}{t_d} \leq 1. && (\forall i \in I, \ell_d(i) \leq 1/t_d) \end{aligned}$$

Since $H \leq 1$, the shelves fit in a dD bin.

Case 2: Suppose $t_d = k$.
Then $\forall i \in I, f_k(\ell_d(i)) = \frac{k}{k-2}\ell_d(i)$. Therefore,

$$\begin{aligned} \text{vol}(\widetilde{f}_k(I)) &= \sum_{i \in I} w(i)\ell_d(i) = \frac{k-2}{k} \sum_{i \in I} w(i) f_k(\ell_d(i)) = \frac{k-2}{k} \text{vol}(f_k(I)) \\ &< \frac{k-2}{k}(1 + \text{vol}(f_k(s))) = \frac{k-2}{k} + \text{vol}(\widetilde{f}_k(s)). \end{aligned} \tag{6.3}$$

Since items in I were sorted in decreasing order of ℓ_d (line 5), $\forall i \in S_j, \ell_d(i) \geq \ell_d(S_{j+1})$. Then by (6.2), we get that for all $j \in [p-1]$,

$$\text{vol}(\widetilde{f}_k(S_j)) \geq w(S_j)\ell_d(S_{j+1}) \geq \ell_d(S_{j+1}) \tag{6.4}$$

Therefore,

$$\begin{aligned}
H = \sum_{j=1}^{p} \ell_d(S_j) &\leq \frac{1}{k} + \sum_{j=1}^{p-1} \ell_d(S_{j+1}) && \text{(since } \ell_d(S_1) \leq 1/k\text{)} \\
&\leq \frac{1}{k} + \sum_{j=1}^{p-1} \mathrm{vol}(\widetilde{f}_k(S_j)) && \text{(by (6.4))} \\
&< \frac{1}{k} + \mathrm{vol}(\widetilde{f}_k(I)) \\
&< \frac{1}{k} + \frac{k-2}{k} + w(s)\ell_d(s) && \text{(by (6.3))} \\
&\leq \frac{k-1}{k} + \frac{1}{k} = 1. && \text{(since } \ell_d(s) \leq 1/k \text{ and } w(s) = \mathrm{vol}(f_k(s^{(d-1)})) \leq 1\text{)}
\end{aligned}$$

Since $H \leq 1$, the shelves fit in a dD bin. Therefore, $\mathcal{P}(d)$ holds.

Therefore, by mathematical induction, $\mathcal{P}(d)$ holds for all $d \geq 1$. □

Note that HDH-unit-pack$_k$ has a running time of $O(n \log n)$.

Comment on Caprara's [18] analysis of HDH$_k$. Caprara [18] implicitly proves Theorem 6.26 in Lemma 4.1 in their paper and their proof is less detailed than ours. Their algorithm is similar to ours, except that they allow arbitrarily reordering I when $t_d \neq k$, and instead of choosing a prefix of I (line 9 in HDH-unit-pack$_k$), they choose a subset of I that is minimal for some properties.

6.6 Harmonic Algorithm for Strip Packing

6.6.1 Multiple-Choice Strip Packing

Let I be a set of dD cuboidal items. In the dD strip packing problem (dSP), we have to compute a feasible packing of I (without rotating the items) into a dD cuboid (called a strip) that has length one in the first $d-1$ dimensions and has the minimum possible length (called height) in the d^{th} dimension. Let $\mathrm{opt}_{d\text{SP}}(I)$ denote the minimum height of a strip needed to pack I.

In the dD multiple-choice strip packing problem (dMCSP), we are given as input a set $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, where for each j, I_j is a set of items, called an *itemset*. We have to pick exactly one item from each itemset and pack those items into a strip of minimum height.

Equivalently, given an input instance $\mathcal{I}$, we have to select an assortment $K \in \Psi(\mathcal{I})$ and output a strip packing of K, such that the total height of the strip is minimized. Therefore, $\mathrm{opt}_{d\text{MCSP}}(\mathcal{I}) := \min_{K \in \Psi(\mathcal{I})} \mathrm{opt}_{d\text{SP}}(K)$.

6.6.2 Revisiting the $\texttt{HDH}_k$ Algorithm

Caprara [18] gave an algorithm for dSP, which we call $\texttt{HDH-SP}_k$. We will first prove a few useful properties of $\texttt{HDH-SP}_k$ and then see how to extend it to dMCSP.

For a dD item i, btype(i) (called *base type*) is defined to be a $(d-1)$-dimensional vector whose j^{th} component is $\text{type}_k(\ell_j(i))$. Define $\widetilde{f}_k(i)$ to be the cuboid $\tilde{\imath}$ where $\ell_j(\tilde{\imath}) := f_k(\ell_j(i))$ for $j \in [d-1]$ and $\ell_d(\tilde{\imath}) := \ell_d(i)$. Define $\widetilde{f}_k(I) := \{\widetilde{f}_k(i) : i \in I\}$. Similarly define $\widetilde{H}_k(i)$ and $\widetilde{H}_k(I)$. Define $i^{(j)}$ to be the j-dimensional item obtained by ignoring all dimensions of i other than the first j. For a set I of dD items, let $I^{(j)} := \{i^{(j)} : i \in I\}$.

$\texttt{HDH-SP}_k$ works by first partitioning the items based on btype. Then for each partition, it repeatedly picks the smallest prefix J such that $\text{vol}(f_k(J^{(d-1)})) \geq 1$ and packs J into a dD shelf by using $\texttt{HDH-unit-pack}_k$ on $J^{(d-1)}$ (see Section 6.5.1 for the definition of a dD shelf). See Algorithm 14 for a more precise description of $\texttt{HDH-SP}_k$. Note that $\texttt{HDH-SP}_k(I)$ has a running time of $O(n \log n)$, where $n := |I|$.

Algorithm 14 $\texttt{HDH-SP}_k(I)$: Returns a strip packing of dD items I ($d \geq 2$).

```
Let P be an empty list.
for each btype t do
    I^[t] = {i ∈ I : btype(i) = t}.
    Sort items in I^[t] in non-increasing order of their length in the d^th dimension.
    while |I^[t]| > 0 do
        Find J, the smallest prefix of I^[t] such that J = I^[t] or vol(f_k(J^(d-1))) ≥ 1.
        S = HDH-unit-pack_k^[t](J^(d-1))        // S is a dD shelf containing items J.
        Append S to the list P.
        Remove J from I^[t].
    end while
end for
Return the strip packing formed by the shelves P.
```

Theorem 6.27. *The height of the strip packing produced by* $\texttt{HDH-SP}_k(I)$ *is less than* $Q + \text{vol}(\widetilde{f}_k(I))$*, where* Q *is the number of distinct* btype*s of items (so* $Q \leq k^{d-1}$*).*

Proof. Let there be $p^{[q]}$ shelves of btype q produced by $\texttt{HDH-SP}_k(I)$. Let $S_j^{[q]}$ be the set of items in the j^{th} shelf of btype q. Define $\ell_d(S_j^{[q]}) := \max_{i \in S_j^{[q]}} \ell_d(i)$ to be the height of shelf $S_j^{[q]}$.

Since items in $I^{[q]}$ were sorted in decreasing order of ℓ_d, $\forall i \in S_j^{[q]}$, $\ell_d(i) \geq \ell_d(S_{j+1}^{[q]})$. Given the way we choose prefixes, $\text{vol}(f_k(S_j^{[q](d-1)})) \geq 1$ for $j \in [p-1]$.

$$\text{vol}(\widetilde{f}_k(S_j^{[q]})) \geq \text{vol}(f_k(S_j^{[q](d-1)}))\ell_d(S_{j+1}^{[q]}) \geq \ell_d(S_{j+1}^{[q]}) \tag{6.5}$$

Total height of the strip packing is

$$\sum_{q=1}^{Q}\sum_{j=1}^{p^{[q]}}\ell_d(S_j^{[q]}) \le \sum_{q=1}^{Q}\left(1+\sum_{j=1}^{p^{[q]}-1}\ell_d(S_{j+1}^{[q]})\right) \qquad \text{(since } \ell_d(S_1^{[q]})\le 1\text{)}$$

$$\le Q+\sum_{q=1}^{Q}\sum_{j=1}^{p^{[q]}-1}\operatorname{vol}(\widetilde{f}_k(S_j^{[q]})) \qquad \text{(by (6.5))}$$

$$< Q+\sum_{q=1}^{Q}\sum_{j=1}^{p^{[q]}}\operatorname{vol}(\widetilde{f}_k(S_j^{[q]})) = Q+\operatorname{vol}(\widetilde{f}_k(I)). \qquad \square$$

Theorem 6.28. *For a set I of dD items,* $\operatorname{vol}(\widetilde{f}_k(I)) \le T_k^{d-1}\operatorname{opt}_{d\mathrm{SP}}(I)$.

Proof. I fits in a unit strip of height $\operatorname{opt}_{d\mathrm{SP}}(I)$. Let I' be the items obtained by scaling each item's height by $1/\operatorname{opt}_{d\mathrm{SP}}(I)$. Then I' fits in a unit cube.

Since H_k is a weighting function, $\widetilde{H}_k(I')$ fits in a unit cube by Theorem 6.1. Therefore, $\widetilde{H}_k(I)$ can be packed into a unit strip of height $\operatorname{opt}_{d\mathrm{SP}}(I)$. Therefore, $\operatorname{vol}(\widetilde{f}_k(I)) \le T_k^{d-1}\operatorname{vol}(\widetilde{H}_k(I)) \le T_k^{d-1}\operatorname{opt}_{d\mathrm{SP}}(I)$. $\square$

Corollary 6.29. $\texttt{HDH-SP}_k(I)$ *packs I into a strip of height less than $Q+T_k^{d-1}\operatorname{opt}_{d\mathrm{SP}}(I)$, where Q is the number of distinct* btype*s of items.*

Proof. Follows from Theorems 6.27 and 6.28. $\square$

6.6.3 Extending $\texttt{HDH-SP}_k$ to dMCSP

Theorem 6.30. *Let $\mathcal{I}$ be a dMCSP instance. Let $\widehat{K} := \{\operatorname{argmin}_{i\in I}\operatorname{vol}(\widetilde{f}_k(i)) : I\in\mathcal{I}\}$, i.e., $\widehat{K}$ is the assortment obtained by picking from each itemset the item i having the minimum value of $\operatorname{vol}(\widetilde{f}_k(i))$. Then the height of the strip packing produced by $\texttt{HDH-SP}_k(\widehat{K})$ is less than $Q+T_k^{d-1}\operatorname{opt}_{d\mathrm{MCSP}}(\mathcal{I})$, where Q is the number of distinct* btype*s of items in* $\operatorname{flat}(\mathcal{I})$ *(so $Q\le k^{d-1}$).*

Proof. For any assortment K, $\operatorname{vol}(\widetilde{f}_k(\widehat{K})) \le \operatorname{vol}(\widetilde{f}_k(K))$. Let K^* be the assortment in an optimal packing of $\mathcal{I}$. By Theorems 6.27 and 6.28, the height of the strip packing produced by $\texttt{HDH-SP}_k(\widehat{K})$ is less than

$$Q+\operatorname{vol}(\widetilde{f}_k(\widehat{K})) \le Q+\operatorname{vol}(\widetilde{f}_k(K^*)) \le Q+T_k^{d-1}\operatorname{opt}_{d\mathrm{SP}}(K^*) = Q+T_k^{d-1}\operatorname{opt}_{d\mathrm{MCSP}}(\mathcal{I}). \qquad \square$$

Let $N := |\operatorname{flat}(\mathcal{I})|$ and $n := |\mathcal{I}|$. Then we can find $\widehat{K}$ in $O(N)$ time and compute $\texttt{HDH-SP}_k(\widehat{K})$ in $O(n \log n)$ time. Therefore, we get a T_k^{d-1}-asymptotic-approximate algorithm for dMCSP that runs in $O(N + n \log n)$ time.

6.7 Harmonic Algorithm for dMCKS

In the dD knapsack problem (dKS), we are given a set I of dD items, and a profit $p(i)$ for each item $i \in I$. We have to compute a maximum-profit packing of a subset of I (without rotating the items) into a dD unit cube (called a knapsack).

In the dD multiple-choice knapsack problem (dMCKS), we are given a set $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ as input, where for each j, I_j is a set of items, called an *itemset*, and each item $i \in I_j$ has a profit $p(i)$. We have to pick at most one item from each itemset and pack those items into a dD bin such that the total profit is maximized.

For a dD item i, $\operatorname{btype}(i)$ (called *base type*) is defined to be a $(d-1)$-dimensional vector whose j^{th} component is $\operatorname{type}_k(\ell_j(i))$. Define $\widetilde{f}_k(i)$ to be the cuboid $\tilde{\imath}$ where $\ell_j(\tilde{\imath}) := f_k(\ell_j(i))$ for $j \in [d-1]$ and $\ell_d(\tilde{\imath}) := \ell_d(i)$. Define $\widetilde{f}_k(I) := \{\widetilde{f}_k(i) : i \in I\}$. Similarly define $\widetilde{H}_k(i)$ and $\widetilde{H}_k(I)$.

We will see a fast and simple algorithm $\texttt{HDH-NF}_k(I)$ (Algorithm 15) for dBP that we will use to design an algorithm for dMCKS.

Algorithm 15 $\texttt{HDH-NF}_k(I)$: Returns a bin packing of dD items I ($d \geq 2$).

1: Let P be the list of shelves output by $\texttt{HDH-SP}_k(I)$. *// cf. Section 6.6 for* $\texttt{HDH-SP}_k$.
2: Let P' be an empty list.
3: **for** each btype q **do**
4: Let $S_1^{[q]}, S_2^{[q]}, \ldots, S_{p^{[q]}}^{[q]}$ be the shelves in P of btype q, in decreasing order of height.
5: Pack $S_1^{[q]}$ in a dD bin.
6: For $j \geq 2$, add $S_j^{[q]}$ to P'.
7: **end for**
8: Interpreting each shelf $S_j^{[q]}$ in P' as a 1D item of size $\ell_d(S_j^{[q]})$, pack the shelves into dD bins using Next-Fit.

Note that $\texttt{HDH-NF}_k(I)$ runs in $O(n \log n)$ time.

Theorem 6.31. $\texttt{HDH-NF}_k(I)$ *uses at most* $Q + \lceil 2 \operatorname{vol}(\widetilde{f}_k(I)) \rceil$ *bins, where* Q *is the number of distinct* btype*s of items.*

Proof. For each $q \in [Q]$, $S_1^{[q]}$ occupies one bin.

As per Eq. (6.5) in the proof of Theorem 6.27, for all $t \leq p^{[q]} - 1$, we get $\operatorname{vol}(\widetilde{f}_k(S_t^{[q]})) \geq \ell_d(S_{t+1}^{[q]})$.

Let H be the total height of the shelves in P'. Then

$$H = \sum_{q=1}^{Q} \sum_{t=1}^{p^{[q]}-1} \ell_d(S_{t+1}^{[q]}) \le \sum_{q=1}^{Q} \sum_{t=1}^{p^{[q]}-1} \operatorname{vol}(\widetilde{f}_k(S_t^{[q]})) \qquad \text{(by (6.5))}$$
$$< \sum_{q=1}^{Q} \sum_{t=1}^{p^{[q]}} \operatorname{vol}(\widetilde{f}_k(S_t^{[q]})) = \operatorname{vol}(\widetilde{f}_k(I)).$$

Next-Fit guarantees that for a 1BP instance J, number of bins used is at most $\lceil 2 \operatorname{vol}(J) \rceil$ (see Lemma 3.1). So for the shelves in P', we use $\lceil 2H \rceil$ bins. The total number of bins used is therefore $Q + \lceil 2H \rceil \le Q + \lceil 2 \operatorname{vol}(\widetilde{f}_k(I)) \rceil$. □

By Theorems 6.28 and 6.31, we get that $\texttt{HDH-NF}_k$ is $2T_k^{d-1}$-asymptotic-approximate.

Lawler gave an FPTAS for 1MCKS that has a running time of $O(N \log N + Nn/\varepsilon)$ [55], where $N := |\operatorname{flat}(\mathcal{I})|$ and $n := |\mathcal{I}|$. We will use it along with $\texttt{HDH-NF}_3$ to get an algorithm for dMCKS, called HDH-KS (see Algorithm 16).

Our algorithm for dMCKS, called $\texttt{HDH-KS}(\mathcal{I})$, works as follows: It computes a 1MCKS instance $\widehat{\mathcal{I}}$ by replacing each item i in $\mathcal{I}$ by a 1D item $\operatorname{vol}(\widetilde{H}_3(i))$. It uses the FPTAS for 1MCKS to obtain a $(1+\varepsilon)$-approximate solution J to $\widehat{\mathcal{I}}$. It uses $\texttt{HDH-NF}_3$ to pack the corresponding dD items of J into bins. It then selects the most profitable bin. See Algorithm 16 for a more detailed description.

Algorithm 16 $\texttt{HDH-KS}(\mathcal{I})$: algorithm for dMCKS.

1: $\widehat{\mathcal{I}} = \{\{\operatorname{vol}(\widetilde{H}_3(i)) : i \in I\} : I \in \mathcal{I}\}$. *// Reduction to 1MCKS.*
2: Let $\widehat{J}$ be a $(1+\varepsilon)$-approximate solution to the 1MCKS instance $\widehat{\mathcal{I}}$ output by the FPTAS for 1MCKS.
3: Let J be the items of $\mathcal{I}$ corresponding to $\widehat{J}$.
4: Let $[J_1, J_2, \ldots, J_b]$ be the bin packing of J produced using $\texttt{HDH-NF}_3$.
5: $j_{\max} = \operatorname{argmax}_{j=1}^{b} p(J_j)$
6: **return** $J_{j_{\max}}$.

HDH-KS runs in $O(N \log N + Nn/\varepsilon)$ time.

Theorem 6.32. HDH-KS *is* $3^d(1+\varepsilon)$*-approximate.*

Proof. Let I be a set of dD items. Suppose $S \subseteq I$ can be packed into a bin. Then by Theorem 6.1, $\widehat{S} = \{\operatorname{vol}(\widetilde{H}_3(i)) : i \in S\}$ can also be packed into a bin. Therefore, $\operatorname{opt}_{1\text{MCKS}}(\widehat{\mathcal{I}}) \ge \operatorname{opt}_{d\text{MCKS}}(\mathcal{I})$.

The FPTAS for 1MCKS gives us $\widehat{J}$ such that $p(\widehat{J}) \geq \mathrm{opt}_{\mathrm{1MCKS}}(\widehat{\mathcal{I}})/(1+\varepsilon)$. $\texttt{HDH-NF}_k$ packs J into $b \leq 3^{d-1} + \lceil 2T_3^{d-1} \operatorname{vol}(\widetilde{H}_3(J)) \rceil \leq 3^d$ bins. Given the way we choose $j_{\max}$,

$$p(J_{j_{\max}}) \geq \frac{p(J)}{b} = \frac{p(\widehat{J})}{b} \geq \frac{\mathrm{opt}_{\mathrm{1MCKS}}(\widehat{\mathcal{I}})}{b(1+\varepsilon)} \geq \frac{\mathrm{opt}_{d\mathrm{MCKS}}(\mathcal{I})}{3^d(1+\varepsilon)}.$$ □

6.8 Weighting Function Transform

In this section, we prove Theorem 6.1.

Lemma 6.33. *Let I be a set of dD items that can be packed into a bin. Let g be a weighting function. Let $q \in [d]$. For $i \in I$, define $g(i)$ to be the item $\hat{\imath}$ for which $\ell_j(\hat{\imath}) := \ell_j(i)$ when $j \neq q$ and $\ell_q(\hat{\imath}) := g(\ell_q(i))$. Then the items $\{g(i) : i \in I\}$ can be packed into a dD bin (without rotating the items).*

Bansal, Caprara and Sviridenko [10] give a brief proof sketch for $d = 2$, based on which we provide a full proof below.

Proof. Any dD cuboid can be represented as the Cartesian product of d closed intervals on the real line. Let the bin be $[0,1]^d$. Any item $i \in I$ can be written as $\prod_{j=1}^{d} [v_j(i), v_j(i) + \ell_j(i)]$. Here $v_j(i)$ is called the *position* of item i in dimension j. Since each item i lies completely inside the bin, $0 \leq v_j(i) < v_j(i) + \ell_j(i) \leq 1$. Two cuboids A and B are said to overlap if their intersection has positive volume. Since I is a valid packing, no two items overlap.

Assume without loss of generality that $q = d$. Let $\mathrm{proj}(i)$ be the projection of item i onto the hyperplane perpendicular to the d^{th} dimension. This hyperplane can be thought of as the *base* of the bin.

We will now show that for each item i, we can change $\ell_d(i)$ to $g(\ell_d(i))$ and change $v_d(i)$ so that the items continue to fit in the bin. But to define what the new value of $v_d(i)$ would be, we need to first introduce some notation.

For two items i_1 and i_2, we say that $i_1 \prec i_2$ (i_1 is a predecessor of i_2) iff $v_d(i_1) < v_d(i_2)$ and $\mathrm{proj}(i_1)$ overlaps $\mathrm{proj}(i_2)$. Call a sequence $[i_0, i_1, \ldots, i_{m-1}]$ of items a *chain* iff $i_{m-1} \prec i_{m-2} \prec \ldots \prec i_0$. i_0 is called the head of this chain. The *augmented height* of a chain S is defined to be $\sum_{i \in S} g(\ell_d(i))$. For each item i, we wish to find the chain headed at i with the maximum augmented height.

For an item i, define

$$\text{level}(i) := \begin{cases} 0 & \text{if } i \text{ has no predecessors} \\ 1 + \max\limits_{i' \prec i} \text{level}(i') & \text{otherwise} \end{cases}.$$

Since $\prec$ is anti-symmetric, level is well-defined. Define π and u as

$$u(i) := g(\ell_d(i)) + \begin{cases} 0 & \text{if level}(i) = 0 \\ u(\pi(i)) & \text{otherwise} \end{cases} \qquad \pi(i) := \begin{cases} \texttt{null} & \text{if level}(i) = 0 \\ \operatorname{argmax}\limits_{i' \prec i} u(i') & \text{otherwise} \end{cases}.$$

In the definition of π, ties can be broken arbitrarily for argmax. $i' \prec i$ implies $\text{level}(i') < \text{level}(i)$, so $\text{level}(\pi(i)) < \text{level}(i)$. This ensures that the definitions of π and u are not mutually circular.

We can prove, by inducting on $\text{level}(i)$, that $\Pi(i) := [i, \pi(i), \pi(\pi(i)), \ldots]$ is the chain headed at i with the maximum augmented height, and that the augmented height of $\Pi(i)$ is $u(i)$.

Transformation 6.13. *For each item $i \in I$, change $\ell_d(i)$ to $g(\ell_d(i))$ and change $v_d(i)$ to $v'_d(i) := u(i) - g(\ell_d(i))$.*

We need to prove that Transformation 6.13 produces a valid packing, i.e. items don't overlap and all items lie completely inside the bin $[0, 1]^d$.

Let i_1 and i_2 be any two items. We will prove that they don't overlap in the new packing. If $\text{proj}(i_1)$ and $\text{proj}(i_2)$ don't overlap, then i_1 and i_2 don't overlap and we are done, so assume $\text{proj}(i_1)$ and $\text{proj}(i_2)$ overlap. Assume without loss of generality that $i_1 \prec i_2$. Then $\text{level}(i_2) \geq 1$ and

$$v'_d(i_2) = u(i_2) - g(\ell_d(i_2)) = \max_{i' \prec i_2} u(i') \geq u(i_1) = v'_d(i_1) + g(\ell_d(i_1)).$$

Therefore, i_1 and i_2 don't overlap in the new packing.

After Transformation 6.13, item i lies completely inside the bin iff $v'_d(i) + g(\ell_d(i)) = u(i) \leq 1$. Let $i_0 := i$ and $\Pi(i) = [i_0, i_1, i_2, \ldots, i_{m-1}]$. Then $u(i) = \sum_{j=0}^{m-1} g(\ell_d(i_j))$ and for all $j \in [m-1], i_j \prec i_{j-1}$. Since i_j and i_{j-1} don't overlap in the original packing, but $\text{proj}(i_j)$ and $\text{proj}(i_{j-1})$ overlap, we get $v_d(i_{j-1}) \geq v_d(i_j) + \ell_d(i_j)$. Therefore,

$$\begin{aligned} \sum_{j=0}^{m-1} \ell_d(i_j) &\leq \ell_d(i) + \sum_{j=1}^{m-1} (v_d(i_{j-1}) - v_d(i_j)) && \text{(since } v_d(i_{j-1}) \geq v_d(i_j) + \ell_d(i_j)\text{)} \\ &= \ell_d(i) + v_d(i) - v_d(i_{m-1}) \leq 1. && (\because \text{in the original packing, } i \text{ lies in the bin}) \end{aligned}$$

Since g is a weighting function and $\sum_{j=0}^{m-1} \ell_d(i_j) \leq 1$, we get $u(i) = \sum_{j=0}^{m-1} g(\ell_d(i_j)) \leq 1$. Therefore, the packing obtained by Transformation 6.13 is valid. So $\{g(i) : i \in I\}$ can be packed into a bin. □

Theorem 6.1. *Let I be a set of dD items that can be packed into a bin. Let $g_1, g_2, \ldots, g_d$ be weighting functions. For $i \in I$, define $g(i)$ as the item whose length is $g_j(\ell_j(i))$ in the j^{th} dimension. Then $\{g(i) : i \in I\}$ can be packed into a dD bin (without rotating the items).*

Proof. Apply Lemma 6.33 multiple times, with q ranging from 1 to d. □

6.9 Hard Instance for Shelf-Based Packing

We will prove that no shelf-based algorithm can get an asymptotic approximation ratio better than T_∞^{d-1} for dD SP or dD BP. Caprara [18] proved this for $d = 2$. To do this, for any $k \geq 3$ and $m > 0$, we will show a set of items that fit into m dD bins but their optimal shelf-based strip-packing has height more than $m(1-\varepsilon)S_k^{d-1}$, where $S_k := \sum_{j=1}^{k-1} \frac{1}{r_j}$. Define $S_\infty := \lim_{k\to\infty} S_k$. It can be proved that $T_\infty = S_\infty \approx 1.6910302$.

It's important to note what exactly we mean by shelf-based. Here we forbid item rotation and only allow stacking shelves along the d^{th} dimension (for $d = 3$, this means that the base of each shelf is perpendicular to the z-axis, and for $d = 2$, this means that all shelves have width 1). As noted by Caprara [18] for $d = 2$, if at the beginning of the algorithm we can choose whether to use horizontal shelves (width=1) or vertical shelves (height=1), and this choice depends on the input items, then we may get an asymptotic approximation ratio less than T_∞.

For simplicity of presentation, we will only consider the $d = 3$ case. We call the first dimension x-axis, the second dimension y-axis and the third dimension z-axis. An item's length in the d^{th} dimension is called *height*. It's easy to extend our result to higher dimensions, and we will give a few hints on how to do so.

Let r_j be the j^{th} harmonic number. Choose an integer $k \geq 2$ and positive constant $0 < \delta \leq 1/(r_k - 1)$ and $\varepsilon > 0$ such that $1/\varepsilon \in \mathbb{Z}$. Define $a_0 := 0$ and for $j \in [k-1]$, define $a_j := (1+\delta)(1 - 1/r_{j+1})$. Therefore, $0 = a_0 < a_1 < a_2 < \ldots < a_{k-1} \leq 1$. Create a cube of side length a_{k-1}. We will cut this cube into pieces, and then cut those pieces into items. Therefore, the items will fit into a bin. See Fig. 6.4 for an example.

First cut the cube using the planes $x = a_j$ for $j \in [k-2]$. Then cut the cube using the planes $y = a_j$ for $j \in [k-2]$. This will give us $Q := (k-1)^2$ pieces. For the piece between the planes $x = a_{q_1-1}$, $x = a_{q_1}$, $y = a_{q_2-1}$ and $y = a_{q_2}$, we call $\vec{q} := (q_1, q_2)$ the *base type* of that

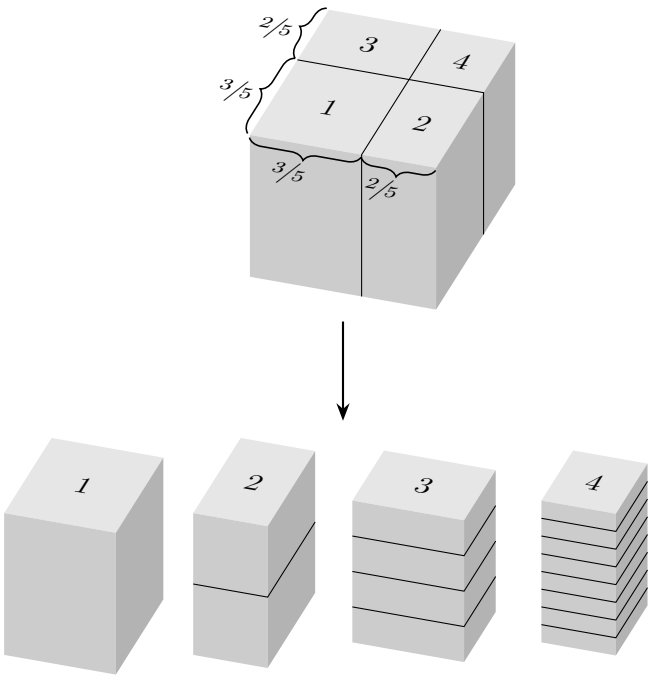


Figure 6.4: Constructing a hard instance for dD shelf-based packing, where $d = 3$, $k = 3$, $\varepsilon = 1/2$, and $\delta = 1/(r_k - 1) = 1/5$. The top half of the figure shows how to cut out $(k-1)^{d-1} = 4$ pieces from a cube of side length $a_{k-1} = 1$. The bottom half shows how to cut out $1/\varepsilon^{j-1}$ items from piece number j.

piece. (For general d, make such cuts in each of the first $d-1$ dimensions in a dD cube to get $Q := (k-1)^{d-1}$ pieces. The type of a piece is a $(d-1)$-dimensional vector.) Now arbitrarily order these pieces and number them from 1 to Q. This number is called the *height type* of the piece. For the piece having height type j, use planes perpendicular to the z-axis to cut it into items of height ε^{j-1}.

Repeat this process for $m-1$ additional cubes. Let I be the resulting set of items. So I can fit into m bins. We call I a *hard instance for shelf-based packing*.

Theorem 6.34. *Let I be a hard instance for shelf-based packing, parametrized by $m > 0$, $k \geq 3$, $0 < \delta \leq 1/(r_k - 1)$, $\varepsilon > 0$. Then the height of an optimal strip-packing of I is more than $m(1-\varepsilon)S_k^{d-1}$.*

Proof. Consider items of height type j. They all have the same base type $\vec{q}$. These items have length $(1+\delta)/(r_{q_1}+1)$ along the x-axis and length $(1+\delta)/(r_{q_2}+1)$ along the y-axis. For any $j \in [k-1]$, we have $1/(r_j+1) < (1+\delta)/(r_j+1) \leq 1/r_j$. Hence, we can have at most $r_{q_1} r_{q_2}$ such items in one shelf. Let $R_j := r_{q_1} r_{q_2}$. Since there are m/ε^{j-1} items of height type j, these items will be spread across at least $m/R_j\varepsilon^{j-1}$ shelves. Those shelves will have height

at least ε^{j-1}. Therefore, for all $j \in [Q]$, we have at least $m_j := m/R_j\varepsilon^{j-1}$ shelves of height at least $h_j := \varepsilon^{j-1}$. We will use this fact to lower-bound the height of the optimal shelf-based strip-packing of I.

Let x_j be the number of shelves of height exactly ε^{j-1}. Then the total height of the optimal shelf-based strip-packing of I is lower-bounded by the following linear program:

$$\min_{x \in \mathbb{R}^Q} \sum_{i=1}^{Q} h_i x_i \quad \text{where} \quad \sum_{j=1}^{i} x_j \geq m_i \qquad \forall i \in [Q].$$

Define $m_0 := h_{Q+1} := 0$. Let $\widehat{x}_i := m_i - m_{i-1}$. Then $\widehat{x}$ is a feasible solution to this linear program and has objective value

$$\sum_{i=1}^{Q} h_i \widehat{x}_i = \sum_{i=1}^{Q} h_i (m_i - m_{i-1}) = \sum_{i=1}^{Q} (h_i - h_{i+1}) m_i.$$

The dual of this linear program is

$$\max_{y \in \mathbb{R}^Q} \sum_{i=1}^{Q} y_i m_i \quad \text{where} \quad \sum_{j=i}^{Q} y_j \leq h_i \qquad \forall i \in [Q].$$

Let $\widehat{y}_i := h_i - h_{i+1}$. Then $\widehat{y}$ is a feasible solution to the dual linear program and has objective value $\sum_{i=1}^{Q} h_i \widehat{x}_i$. By the weak duality of linear programs, $\widehat{x}$ is an optimal solution to the linear program. Therefore, the total height of the optimal shelf-based strip-packing of I is lower-bounded by

$$\begin{aligned}
\sum_{i=1}^{Q} (h_i - h_{i+1}) m_i > m \sum_{i=1}^{Q} (\varepsilon^{i-1} - \varepsilon^i) \frac{1}{R_i \varepsilon^{i-1}} &= m(1-\varepsilon) \sum_{i=1}^{Q} \frac{1}{R_i} \\
&= m(1-\varepsilon) \sum_{q_1=1}^{k-1} \sum_{q_2=1}^{k-1} \frac{1}{r_{q_1} r_{q_2}} = m(1-\varepsilon) \left(\sum_{q=1}^{k-1} \frac{1}{r_q} \right)^{d-1} \\
&= m(1-\varepsilon) S_k^{d-1}.
\end{aligned}$$

□

Chapter 7

Guillotine-Separable Packing of Skewed Rectangles

In this chapter, we consider the problem of obtaining tight upper and lower bounds on the Asymptotic Price of Guillotinability (APoG). See Section 1.2.3 to recall the definition and significance of APoG. See Section 1.1.4 to recall the definition of guillotinable packing and k-stage packing.

We focus on the special case where the items are (δ_W, δ_H)-skewed rectangles, i.e., each item has width at most δ_W or height at most δ_H, where δ_W and δ_H are constants. We give lower and upper bounds of roughly $4/3$ when δ_W and δ_H are very small constants.

7.1 Overview of the Chapter

In Section 7.2, we show a lower bound of $4/3$ on APoG for skewed rectangles. Formally, we prove the following theorem.

Definition 7.1 (Hard instance)**.** *Let m and k be positive integers and $\varepsilon \in (0, 1)$. Define* $\text{hardItems}(m, k, \varepsilon)$ *as a set of $4mk$ rectangular items, where $2mk$ items have width $(1+\varepsilon)/2$ and height $(1-\varepsilon)/2k$, and $2mk$ items have height $(1+\varepsilon)/2$ and width $(1-\varepsilon)/2k$.*

Theorem 7.1. *Let $I := \text{hardItems}(m, k, \varepsilon)$. Let $\text{opt}(I)$ be the number of bins in the optimal packing of I and $\text{opt}_g(I)$ be the number of bins in the optimal guillotinable packing of I. Then*

$$\frac{\text{opt}_g(I)}{\text{opt}(I)} \geq \frac{4}{3}(1-\varepsilon).$$

This holds true even if items in I are allowed to be rotated.

In Section 7.3, we give an algorithm for non-rotational 2D GBP, called $\texttt{skewed4Pack}_\varepsilon$, that takes a parameter $\varepsilon \in (0,1)$ as input. For a set I of (δ_W, δ_H)-skewed rectangles, we show that when δ_W, δ_H and ε are close to 0, $\texttt{skewed4Pack}_\varepsilon(I)$ outputs a guillotinable packing of I into roughly $4\,\mathrm{opt}(I)/3 + O(1)$ bins. This proves an upper bound of roughly 4/3 on APoG for skewed rectangles. Formally, we prove the following theorems about `skewed4Pack`.

Theorem 7.2. $\texttt{skewed4Pack}_\varepsilon(I)$ *outputs a 4-stage packing of I in time $O((1/\varepsilon)^{O(1/\varepsilon)} + n \log n)$.*

Theorem 7.3. *Let I be a set of items where each item has width at most δ_W or height at most δ_H. Then $\texttt{skewed4Pack}_\varepsilon(I)$ uses less than $\alpha(1+\varepsilon)\,\mathrm{opt}(I) + 2\beta$ bins, where*

$$\Delta := \frac{1}{2}\left(\frac{\delta_H}{1-\delta_H} + \frac{\delta_W}{1-\delta_W}\right)$$
$$\alpha := \frac{4}{3}(1+4\Delta) + 4\varepsilon\left(1 + \frac{7\Delta}{3}\right) \le \frac{4}{3}(1+4\Delta)(1+3\varepsilon)$$
$$\beta := \frac{2\Delta(1+\varepsilon)}{\varepsilon^2} + \frac{10}{3} + \frac{19\Delta}{3} + \frac{16\Delta\varepsilon}{3}.$$

In Section 7.4, we show that the APoG for the rotational case is at most the APoG for the non-rotational case. So, when are items are skewed, we get an upper bound of 4/3 on APoG in the rotational case too.

7.2 Lower Bound on APoG

Lemma 7.4. *Let m and k be positive integers and ε be a positive real number. Let J be a set of items packed into a bin, where each item has the longer dimension equal to $(1+\varepsilon)/2$ and the shorter dimension equal to $(1-\varepsilon)/2k$. If the bin is guillotine-separable, then $a(J) \le 3/4 + \varepsilon/2 - \varepsilon^2/4$.*

Proof. For an item packed in the bin, if the height is $(1-\varepsilon)/2k$, call it a wide item, and if the width is $(1-\varepsilon)/2k$, call it a tall item. Let W be the set of wide items in J.

The packing of items in the bin can be represented as a tree, called the *guillotine tree* of the bin, where each node u represents a rectangular region of the bin and the child nodes $v_1, v_2, \ldots, v_p$ of node u represent the sub-regions obtained by parallel guillotine cuts. The ordering of the children has a significance here: if the guillotine cuts were vertical, children

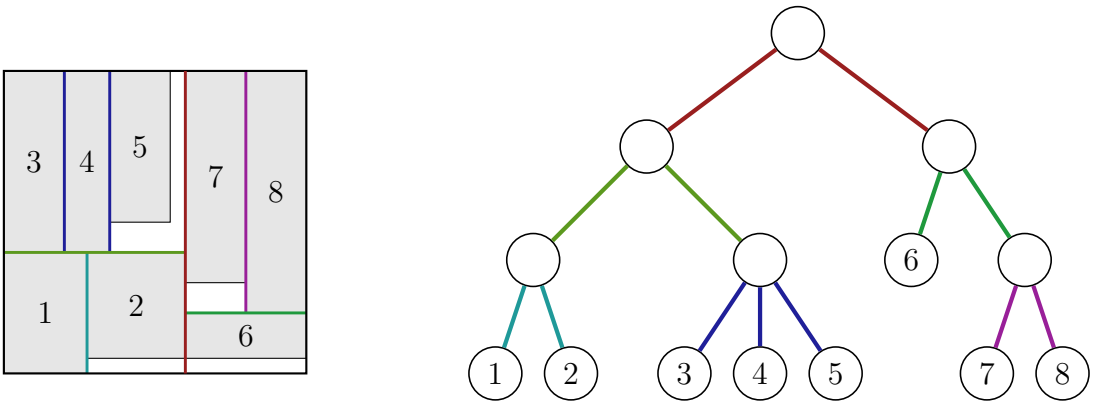


Figure 7.1: A guillotinable packing of items into a bin and the corresponding guillotine tree.

are ordered by increasing x-coordinate, and if the cuts were horizontal, children are ordered by increasing y-coordinate. See Fig. 7.1 for an example.

We will now see how to rearrange the items in the bin so that the packing remains guillotine-separable but becomes more structured. We will exploit this structure to show that the packing has a large unpacked area. See Fig. 7.2 for an example.

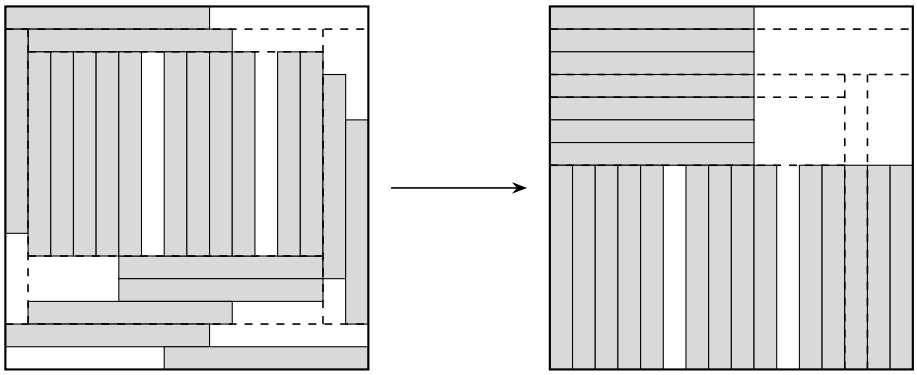


Figure 7.2: Structuring a guillotine-separable packing.

In the guillotine tree, suppose there is a node u that has children $v_1, v_2, \ldots, v_p$. Without loss of generality, assume that the children are obtained by making vertical cuts. At most one of these children can contain items from W. We can assume without loss of generality that the other children contain only one item, because otherwise we can separate them by vertical cuts. We can reorder the children (which is equivalent to repacking the guillotine partitions) so that the child containing items from W (if any) is the first child. Therefore, we can assume without loss of generality that at any level in the guillotine tree, only the first node has children.

Based on the argument above, we can see that the first node in each level touches the bottom-left corner of the bin. All the other nodes either contain a single wide item and touch

the left edge of the bin but not the bottom edge, or they contain a single tall item and touch the bottom edge of the bin but not the left edge. In each node containing a wide item, shift the item leftwards, and in each node containing a tall item, shift the item downwards. Then each wide item touches the left edge of the bin and each tall item touches the bottom edge of the bin.

Therefore, the square region of side length $(1-\varepsilon)/2$ at the top-right corner of the bin is empty. Hence, the area occupied in each bin is at most $3/4 + \varepsilon/2 - \varepsilon^2/4$. □

Definition 7.1 (Hard instance)**.** *Let m and k be positive integers and $\varepsilon \in (0,1)$. Define* $\text{hardItems}(m,k,\varepsilon)$ *as a set of $4mk$ rectangular items, where $2mk$ items have width $(1+\varepsilon)/2$ and height $(1-\varepsilon)/2k$, and $2mk$ items have height $(1+\varepsilon)/2$ and width $(1-\varepsilon)/2k$.*

Theorem 7.1. *Let $I := \text{hardItems}(m,k,\varepsilon)$. Let $\text{opt}(I)$ be the number of bins in the optimal packing of I and $\text{opt}_g(I)$ be the number of bins in the optimal guillotinable packing of I. Then*

$$\frac{\text{opt}_g(I)}{\text{opt}(I)} \geq \frac{4}{3}(1-\varepsilon).$$

This holds true even if items in I are allowed to be rotated.

Proof. For an item $i \in I$, if $h(i) = (1-\varepsilon)/2k$, call it a wide item, and if $w(i) = (1-\varepsilon)/2k$, call it a tall item. Let W be the set of wide items and H be the set of tall items.

We will show that $\text{opt}(I)$ and $\text{opt}_g(I)$ have a big difference, which will give us a lower-bound on APoG.

Partition W into groups of k elements. In each group, stack items one-over-the-other. This gives us $2m$ containers of width $(1+\varepsilon)/2$ and height $(1-\varepsilon)/2$. Similarly, get $2m$ containers of height $(1+\varepsilon)/2$ and height $(1-\varepsilon)/2$ by stacking items from H side-by-side. We can pack 4 containers in one bin, so I can be packed into m bins. See Fig. 7.3 for an example. Therefore, $\text{opt}(I) \leq m$.

We will now show a lower-bound on $\text{opt}_g(I)$. In any guillotine-separable packing of I, the area occupied by each bin is at most $3/4 + \varepsilon/2 - \varepsilon^2/4$ (by Lemma 7.4). Note that $a(I) = m(1-\varepsilon^2)$. Therefore,

$$\text{opt}_g(I) \geq \frac{m(1-\varepsilon^2)}{3/4 + \varepsilon/2 - \varepsilon^2/4}$$
$$\implies \frac{\text{opt}_g(I)}{\text{opt}(I)} \geq \frac{4}{3} \times \frac{1-\varepsilon^2}{1 + 2\varepsilon/3 - \varepsilon^2/3} = \frac{4}{3} \times \frac{1-\varepsilon}{1-\varepsilon/3} \geq \frac{4}{3}(1-\varepsilon).$$ □

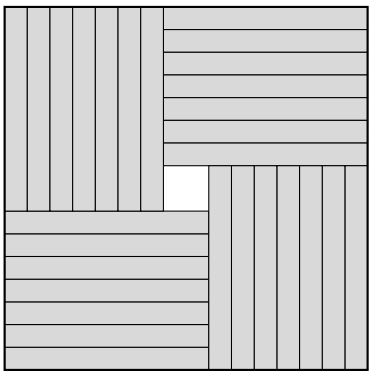


Figure 7.3: Packing $4k$ items in one bin. Here $k = 7$.

7.3 Algorithm `skewed4Pack`

7.3.1 Packing With Slicing

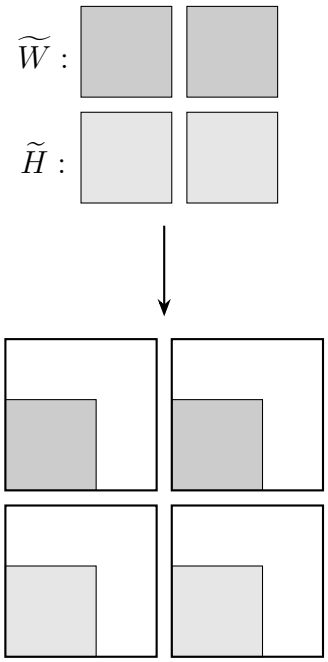
Before we look at the problem of computing a (guillotinable) packing of skewed rectangles, let us first look at a closely-related variant of this problem, called the *2D sliceable bin packing problem*, denoted as 2D SBP. In this problem, we are given two sets of rectangular items, $\widetilde{W}$ and $\widetilde{H}$, where items in $\widetilde{W}$ have width more than 1/2, and items in $\widetilde{H}$ have height more than 1/2. $\widetilde{W}$ is called the set of wide items and $\widetilde{H}$ is called the set of tall items. We are allowed to *slice* items in $\widetilde{W}$ using horizontal cuts and slice items in $\widetilde{H}$ using vertical cuts, and our task is to pack $\widetilde{W} \cup \widetilde{H}$ into the minimum number of bins without rotating the items. See Fig. 7.4 for an example that illustrates the difference between 2D GBP and 2D SBP.

We first describe a fast and simple 4/3-asymptotic-approximation algorithm for 2D SBP, called `greedyPack`, that outputs a 2-stage packing. Later, we will show how to use `greedyPack` to design `skewed4Pack`.

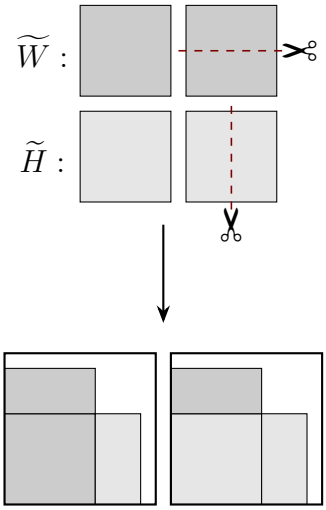
We assume that the bin is a square of side length 1. Since we can slice items, we allow items in $\widetilde{W}$ to have height more than 1 and items in $\widetilde{H}$ to have width more than 1.

For a set $X \subseteq \widetilde{W}$ of items, define

$$\text{hsum}(X) := \sum_{i \in X} h(i) \quad \text{and} \quad \text{wmax}(X) := \begin{cases} \max_{i \in X} w(i) & \text{if } X \neq \emptyset \\ 0 & \text{if } X = \emptyset \end{cases}.$$



(a) Packing items into 4 bins without slicing.



(b) Packing items into 2 bins by horizontally slicing an item in $\widetilde{W}$ and vertically slicing an item in $\widetilde{H}$.

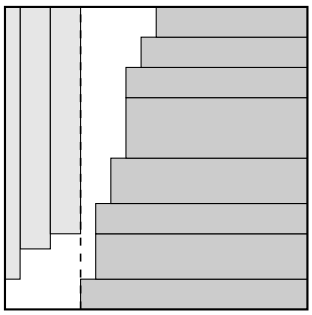
Figure 7.4: Example to illustrate the difference between 2D geometric bin packing and 2D sliceable bin packing. There are 2 wide items ($\widetilde{W}$) and 2 tall items ($\widetilde{H}$). The items are squares of side length 0.6 and the bins are squares of side length 1.

For a set $X \subseteq \widetilde{H}$ of items, define

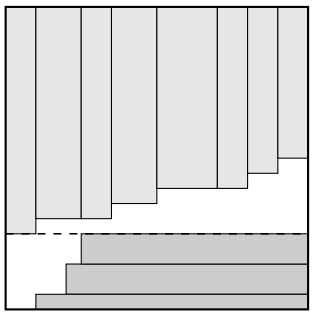
$$\mathrm{wsum}(X) := \sum_{i \in X} w(i) \quad \text{and} \quad \mathrm{hmax}(X) := \begin{cases} \max_{i \in X} h(i) & \text{if } X \neq \emptyset \\ 0 & \text{if } X = \emptyset \end{cases}.$$

In the algorithm `greedyPack`$(\widetilde{W}, \widetilde{H})$, we first sort the items $\widetilde{W}$ in decreasing order of width and sort the items $\widetilde{H}$ in decreasing order of height. If $\mathrm{hsum}(\widetilde{W}) \geq \mathrm{wsum}(\widetilde{H})$, then we pack the largest possible prefix of $\widetilde{W}$ into a bin such that the items touch the right edge of the bin. Then we pack a prefix of $\widetilde{H}$ into the remaining space in the side of the bin. We call this a type-1 bin. See Fig. 7.5 for an example. If $\mathrm{hsum}(\widetilde{W}) < \mathrm{wsum}(\widetilde{H})$, we proceed analogously in a coordinate-swapped way, i.e., we first pack tall items in the bin and then pack wide items in the remaining space. Call this bin a type-2 bin. We pack the rest of the items into bins in the same way. See Algorithm 17 for a more precise description of `greedyPack`.

Definition 7.2. *Let $\widetilde{W}$ be a sequence of wide items. Define* $\mathrm{hprefix}(\widetilde{W}, \gamma)$ *as the prefix of $\widetilde{W}$ of total height γ if* $\mathrm{hsum}(\widetilde{W}) > \gamma$ *(slice items if necessary). If* $\mathrm{hsum}(\widetilde{W}) \leq \gamma$*, define* $\mathrm{hprefix}(\widetilde{W}, \gamma)$ *to be $\widetilde{W}$. Let $\widetilde{H}$ be a sequence of tall items. Define* $\mathrm{wprefix}(\widetilde{H}, \gamma)$ *as the prefix of $\widetilde{H}$ of total*



(a) A type-1 bin. Wide items are packed on the right. Tall items are packed on the left.



(b) A type-2 bin. Tall items are packed above. Wide items are packed below.

Figure 7.5: Examples of type-1 and type-2 bins produced by `greedyPack`.

width γ *if* $\text{wsum}(\widetilde{H}) > \gamma$ *(slice items if necessary). If* $\text{wsum}(\widetilde{H}) \leq \gamma$*, define* $\text{wprefix}(\widetilde{H}, \gamma)$ *to be* $\widetilde{H}$*.*

Algorithm 17 `greedyPack`$(\widetilde{W}, \widetilde{H})$: Packs items $\widetilde{W} \cup \widetilde{H}$ into bins. The items $\widetilde{W}$ have width more than 1/2 and can be sliced using horizontal cuts. The items $\widetilde{H}$ have width more than 1/2 and can be sliced using vertical cuts.

Sort the items in $\widetilde{W}$ in decreasing order of width.
Sort the items in $\widetilde{H}$ in decreasing order of height.
while $\widetilde{W} \neq \emptyset$ or $\widetilde{H} \neq \emptyset$ **do**
    Create an empty bin.
    if $\text{hsum}(\widetilde{W}) \geq \text{wsum}(\widetilde{H})$ **then**
        Let $X := \text{hprefix}(\widetilde{W}, 1)$. *// see Definition 7.2*
        Pack X in a region of width $\text{wmax}(X)$ on the right side of the bin.
        Remove X from $\widetilde{W}$.
        Let $Y := \text{wprefix}(\widetilde{H}, 1 - \text{wmax}(X))$. *// see Definition 7.2*
        Pack Y in a region of width $1 - \text{wmax}(X)$ on the left side of the bin.
        Remove Y from $\widetilde{H}$.
        Label the bin as a type-1 bin.
    else
        Proceed analogous to the previous case, i.e., X is a prefix of $\widetilde{H}$ of width at most 1 and Y is a prefix of $\widetilde{W}$ of total height at most $1 - \text{hmax}(X)$.
        Label the bin as a type-2 bin.
    end if
end while

Claim 7.5. `greedyPack`$(\widetilde{W}, \widetilde{H})$ *always outputs a 2-stage packing of* $\widetilde{W} \cup \widetilde{H}$*. It runs in time* $O(m + |\widetilde{W}| \log |\widetilde{W}| + |\widetilde{H}| \log |\widetilde{H}|)$*, where* m *is the number of bins used. Furthermore, it slices*

items in $\widetilde{W}$ by making at most $m-1$ horizontal cuts and slices items in $\widetilde{H}$ by making at most $m-1$ vertical cuts.

If all bins are of type 1, then the number of bins used is $\lceil \text{hsum}(\widetilde{W}) \rceil$. If all bins are of type 2, then the number of bins used is $\lceil \text{wsum}(\widetilde{H}) \rceil$. Since items in $\widetilde{W}$ have width more than $1/2$, no two items can be placed side-by-side. If $\widetilde{H} = \{\}$, then the optimal solution is to stack the items $\widetilde{W}$ one-over-the-other. Therefore, $\lceil \text{hsum}(\widetilde{W}) \rceil \leq \text{opt}(\widetilde{W} \cup \widetilde{H})$. Similarly, $\lceil \text{wsum}(\widetilde{H}) \rceil \leq \text{opt}(\widetilde{W} \cup \widetilde{H})$. Hence, if all bins are of the same type, the number of bins used is at most $\text{opt}(\widetilde{W} \cup \widetilde{H})$.

We will now focus on the more interesting case, i.e., some bins are of type 1 and some are of type 2.

Definition 7.3 (Full bin)**.** *In a type-1 bin, let X be the items from $\widetilde{W}$ and Y be the items from $\widetilde{H}$. The bin is said to be* full *iff* $\text{hsum}(X) = 1$ *and* $\text{wsum}(Y) = 1 - \text{wmax}(X)$*. Define fullness for a type-2 bin analogously.*

We first show that full bins pack items of a large total area, and then we show that if some bins are of type 1 and some bins are of type 2, then there can be at most 2 non-full bins. This will help us get an upper-bound on the number of bins used by `greedyPack`$(\widetilde{W}, \widetilde{H})$ in terms of $a(\widetilde{W} \cup \widetilde{H})$.

Lemma 7.6. *Let there be m_1 full bins of type 1. Let J_1 be the items inside those bins. Then $m_1 \leq 4a(J_1)/3 + 1/3$.*

Proof. In the j^{th} full bin of type 1, let X_j be the items from $\widetilde{W}$ and Y_j be the items from $\widetilde{H}$. Let

$$\ell_j := \begin{cases} \text{wmax}(X_j) & \text{if } j \leq m_1 \\ 1/2 & \text{if } j = m_1 + 1 \end{cases}.$$

Since all items have their larger dimension more than $1/2$, $\ell_j \geq 1/2$ and $\text{hmax}(Y_j) > 1/2$.

$a(X_j) \geq \ell_{j+1}$, since X_j has height 1 and width at least ℓ_{j+1}. $a(Y_j) \geq (1 - \ell_j)/2$, since Y_j

has width $1 - \ell_j$ and height more than $1/2$. Therefore,

$$\begin{aligned} a(J_1) &= \sum_{j=1}^{m_1} (a(X_j) + a(Y_j)) \geq \sum_{j=1}^{m_1} (\ell_{j+1} + (1 - \ell_j)/2) \\ &\geq \sum_{j=1}^{m_1} \left(\frac{\ell_{j+1}}{2} + \frac{1}{4} + \frac{1}{2} - \frac{\ell_j}{2} \right) && (\ell_{j+1} \geq 1/2) \\ &= \frac{3m_1}{4} + \frac{1}{4} - \frac{\ell_1}{2} \geq \frac{3m_1 - 1}{4}. \end{aligned}$$

Therefore, $m_1 \leq 4a(J_1)/3 + 1/3$. □

An analogue of Lemma 7.6 can be proven for type-2 bins.

Let m be the number of bins used by `greedyPack`$(\widetilde{W}, \widetilde{H})$. After j bins have been packed, let A_j be the height of the remaining items in $\widetilde{W}$ and B_j be the width of the remaining items in $\widetilde{H}$. Let t_j be the type of the j^{th} bin (1 for type-1 bin and 2 for type-2 bin). So $t_j = 1 \iff A_{j-1} \geq B_{j-1}$.

We first show that $|A_{j-1} - B_{j-1}| \leq 1 \implies |A_j - B_j| \leq 1$. This means that once the difference between $h(\widetilde{W})$ and $w(\widetilde{H})$ becomes at most 1, it continues to stay at most 1. Next, we show that $t_j \neq t_{j+1} \implies |A_{j-1} - B_{j-1}| \leq 1$. This means that if there are some bins of type 1 and some bins of type 2, then the difference between $h(\widetilde{W})$ and $w(\widetilde{H})$ will eventually become at most 1. In the first non-full bin, we will use up all the wide items or the tall items. Then the remaining items will have total height or total width at most 1, so we can have at most 1 more non-full bin. This would imply that there can be at most 2 non-full bins when we have both type-1 and type-2 bins.

In the j^{th} bin, let a_j be the height of items from $\widetilde{W}$ and b_j be the width of items from $\widetilde{H}$. Hence, for all $j \in [m]$, $A_{j-1} = A_j + a_j$ and $B_{j-1} = B_j + b_j$.

Lemma 7.7. $|A_{j-1} - B_{j-1}| \leq 1 \implies |A_j - B_j| \leq 1$.

Proof. **Case 1:** $A_{j-1} - B_{j-1} \in [0, 1]$.
This means that $t_j = 1$.

Assume (for the sake of proof by contradiction) that $a_j < b_j$. Then $a_j < 1$, so we used up all of $\widetilde{W}$ in the j^{th} bin. Therefore, $A_j = 0$ and $A_{j-1} = a_j$. Therefore,

$$A_{j-1} = a_j < b_j \leq b_j + B_j = B_{j-1} \implies \perp.$$

Therefore, $a_j \geq b_j$. Since $A_{j-1} - B_{j-1} \in [0, 1]$ and $a_j - b_j \in [0, 1]$, we get

$$A_j - B_j = (A_{j-1} - B_{j-1}) - (a_j - b_j) \in [-1, 1].$$

Therefore, $|A_j - B_j| \leq 1$.

Case 2: $A_{j-1} - B_{j-1} \in [-1, 0)$.

This means that $t_j = 2$. By an analysis similar to case 1, we get $|A_j - B_j| \leq 1$. □

Lemma 7.8. $t_j \neq t_{j+1} \implies |A_{j-1} - B_{j-1}| \leq 1$.

Proof.

$$\begin{aligned}
&t_j = 1 \text{ and } t_{j+1} = 2 \\
&\implies A_{j-1} \geq B_{j-1} \text{ and } A_j < B_j \\
&\implies B_{j-1} \leq A_{j-1} < B_{j-1} + a_j - b_j \\
&\implies A_{j-1} - B_{j-1} \in [0, 1).
\end{aligned}$$

$$\begin{aligned}
&t_j = 2 \text{ and } t_{j+1} = 1 \\
&\implies A_{j-1} < B_{j-1} \text{ and } A_j \geq B_j \\
&\implies A_{j-1} < B_{j-1} \leq A_{j-1} + (b_j - a_j) \\
&\implies A_{j-1} - B_{j-1} \in [-1, 0).
\end{aligned}$$

□

Lemma 7.9. *Let there be p full bins. If all bins don't have the same type, then $|A_p - B_p| \leq 1$.*

Proof. If $m = p$, then $A_p = B_p = 0$, so $|A_p - B_p| \leq 1$ trivially holds. If $m = p + 1$, then $A_p \leq 1$ and $B_p \leq 1$, so $|A_p - B_p| \leq 1$ trivially holds. So now assume $m \geq p + 2$.

Suppose that in the $(p+1)^{\text{th}}$ bin, we used up all items from $\widetilde{W}$ but not $\widetilde{H}$. Then $\forall i \geq p+2$, $t_i = 2$. Since all bins don't have the same type, $\exists k \leq p + 1$ such that $t_k = 1$ and $t_{k+1} = 2$. By Lemmas 7.7 and 7.8, we get $|A_p - B_p| \leq 1$. Similarly, if we used up all items from $\widetilde{H}$ in the $(p + 1)^{\text{th}}$ bin, then $|A_p - B_p| \leq 1$. □

Lemma 7.10. *If all bins don't have the same type, then there can be at most 2 non-full bins.*

Proof. Let there be p full bins. Assume that there are more than 2 non-full bins. Without loss of generality, assume that the first non-full bin used up all wide items. Hence, $A_{p+1} = 0$. By Lemma 7.9, we get $|A_p - B_p| \leq 1$. By Lemma 7.7, we get $|A_{p+1} - B_{p+1}| \leq 1$, which implies

that $B_{p+1} \leq 1$. Hence, the $(p+1)^{\text{th}}$ bin will have type 2 and will use up all tall items, so there can be at most 2 non-full bins. □

Theorem 7.11. *The number of bins used by* `greedyPack` *is at most*

$$\max\left(\lceil \text{hsum}(\widetilde{W})\rceil, \lceil \text{wsum}(\widetilde{H})\rceil, \frac{4}{3}a(\widetilde{W}\cup\widetilde{H}) + \frac{8}{3}\right).$$

Proof. Let there be m bins in the output of `greedyPack`$(\widetilde{W}, \widetilde{H})$. If all bins have the same type, then $m \leq \max(\lceil \text{hsum}(\widetilde{W})\rceil, \lceil \text{wsum}(\widetilde{H})\rceil)$.

Let there be m_1 full bins of type 1 and let J_1 be the items inside those bins. Let there be m_2 full bins of type 2 and let J_2 be the items inside those bins. Then by Lemma 7.6, we get $m_1 \leq 4a(J_1)/3 + 1/3$ and $m_2 \leq 4a(J_2)/3 + 1/3$. Hence, $m_1 + m_2 \leq 4a(\widetilde{W}\cup\widetilde{H})/3 + 2/3$. If all bins don't have the same type, then by Lemma 7.10, there can be at most 2 non-full bins, so `greedyPack`$(\widetilde{W}, \widetilde{H})$ uses at most $4a(\widetilde{W}\cup\widetilde{H})/3 + 8/3$ bins. □

7.3.2 Overview of `skewed4Pack`

`skewed4Pack` takes as input a set I of rectangular items and a parameter $\varepsilon \in (0, 1)$. It outputs a 4-stage bin packing of I.

`skewed4Pack` has the following outline:

1. Use linear grouping to round up the width or height of each item in I. This gives us a new instance $\widehat{I}$.
2. Pack $\widehat{I}$ into $1/\varepsilon^2 + 1$ shelves, after possibly *slicing* some items. Each shelf has width or height more than $1/2$ and is fully packed, i.e., the total area of items in a shelf equals the area of the shelf. If we treat each shelf as an item, we get a new instance $\widetilde{I}$.
3. Compute a packing of $\widetilde{I}$ into bins, after possibly slicing some items, using `greedyPack`.
4. Pack most of the items of I into the shelves in the bins. We will prove that the remaining items have very small area, so they can be packed separately.

To simplify our algorithm, we assume that $\varepsilon^{-1} \in \mathbb{Z}$.

7.3.3 Item Classification and Rounding

Define $W := \{i \in I : h(i) \leq \delta_H\}$ and $H := I - W$. Items in W are called *wide* and items in H are called *tall*. Let $W^{(L)} := \{i \in W : w(i) > \varepsilon\}$ and $W^{(S)} := W - W^{(L)}$. Similarly, let

$H^{(L)} := \{i \in H : h(i) > \varepsilon\}$ and $H^{(S)} := H - H^{(L)}$.

We will now use *linear grouping* [26, 49] to round up the widths of items in $W^{(L)}$ and the heights of items in $H^{(L)}$. Arrange the items of $W^{(L)}$ in decreasing order of width and stack them one-over-the-other (i.e., the widest item in $W^{(L)}$ is at the bottom). Let h_L be the height of the stack. Let $y(i)$ be the y-coordinate of the bottom edge of item i. Split the stack into sections of height $\varepsilon^2 h_L$ each. For $j \in [1/\varepsilon^2]$, let w_j be the width of the widest item intersecting the j^{th} section from the bottom, i.e.,

$$w_j := \max(\{w(i) : i \in W^{(L)} \text{ and } (y(i), y(i) + h(i)) \cap ((j-1)\varepsilon^2 h_L, j\varepsilon^2 h_L) \neq \emptyset\}).$$

Round up the width of each item i to the smallest w_j that is at least $w(i)$ (see Fig. 7.6). Let $W_j^{(L)}$ be the items whose width got rounded to w_j and let $\widehat{W}_j^{(L)}$ be the resulting rounded items. (There may be ties, i.e., there may exist $j_1 < j_2$ such that $w_{j_1} = w_{j_2}$. In that case, define $W_{j_2}^{(L)} := \widehat{W}_{j_2}^{(L)} = \emptyset$. This ensures that all $W_j^{(L)}$ are disjoint.) Let $\widehat{W}^{(L)} := \bigcup_j \widehat{W}_j^{(L)}$.

Allow horizontally slicing each item in $\widehat{W}^{(L)}$. Let $\widehat{W}^{(S)}$ be the same as $W^{(S)}$, except that we are allowed to slice items in $\widehat{W}^{(S)}$ both horizontally and vertically. Let $\widehat{W} := \widehat{W}^{(L)} \cup \widehat{W}^{(S)}$. Define $\widehat{H}$ analogously. Let $\widehat{I} := \widehat{W} \cup \widehat{H}$.

Claim 7.12. *Items in $\widehat{W}^{(L)}$ have at most $1/\varepsilon^2$ distinct widths. Items in $\widehat{H}^{(L)}$ have at most $1/\varepsilon^2$ distinct heights.*

Lemma 7.13. $\operatorname{opt}(\widehat{I}) < (1+\varepsilon)\operatorname{opt}(I) + 2$.

Proof. Consider the optimal packing of I. To convert this to a packing of $\widehat{I} - (\widehat{W}_1^{(L)} \cup \widehat{H}_1^{(L)})$, unpack $W_1^{(L)}$ and $H_1^{(L)}$, and for each $j \in [1/\varepsilon^2 - 1]$, pack $\widehat{W}_{j+1}^{(L)}$ in the place of $W_j^{(L)}$ and pack $\widehat{H}_{j+1}^{(L)}$ in the place of $H_j^{(L)}$, possibly after slicing the items. Therefore,

$$\operatorname{opt}(\widehat{I} - (\widehat{W}_1^{(L)} \cup \widehat{H}_1^{(L)})) \leq \operatorname{opt}(I). \tag{7.1}$$

We can pack $\widehat{H}_1^{(L)}$ in a bin by stacking the items side-by-side on the base of bins. We can pack $\widehat{W}_1^{(L)}$ in a bin by stacking the items one-over-the-other. Let w_L be the total width of items in $\widehat{H}^{(L)}$. The number of bins used is $\lceil \varepsilon^2 h_L \rceil + \lceil \varepsilon^2 w_L \rceil$. Also,

$$\operatorname{opt}(I) \geq \operatorname{opt}(W^{(L)} \cup H^{(L)}) \geq a(W^{(L)}) + a(H^{(L)}) \geq \varepsilon(h_L + w_L).$$

Therefore,

$$\operatorname{opt}(\widehat{W}_1^{(L)} \cup \widehat{H}_1^{(L)}) \leq \lceil \varepsilon^2 h_L \rceil + \lceil \varepsilon^2 w_L \rceil < \varepsilon \operatorname{opt}(I) + 2. \tag{7.2}$$

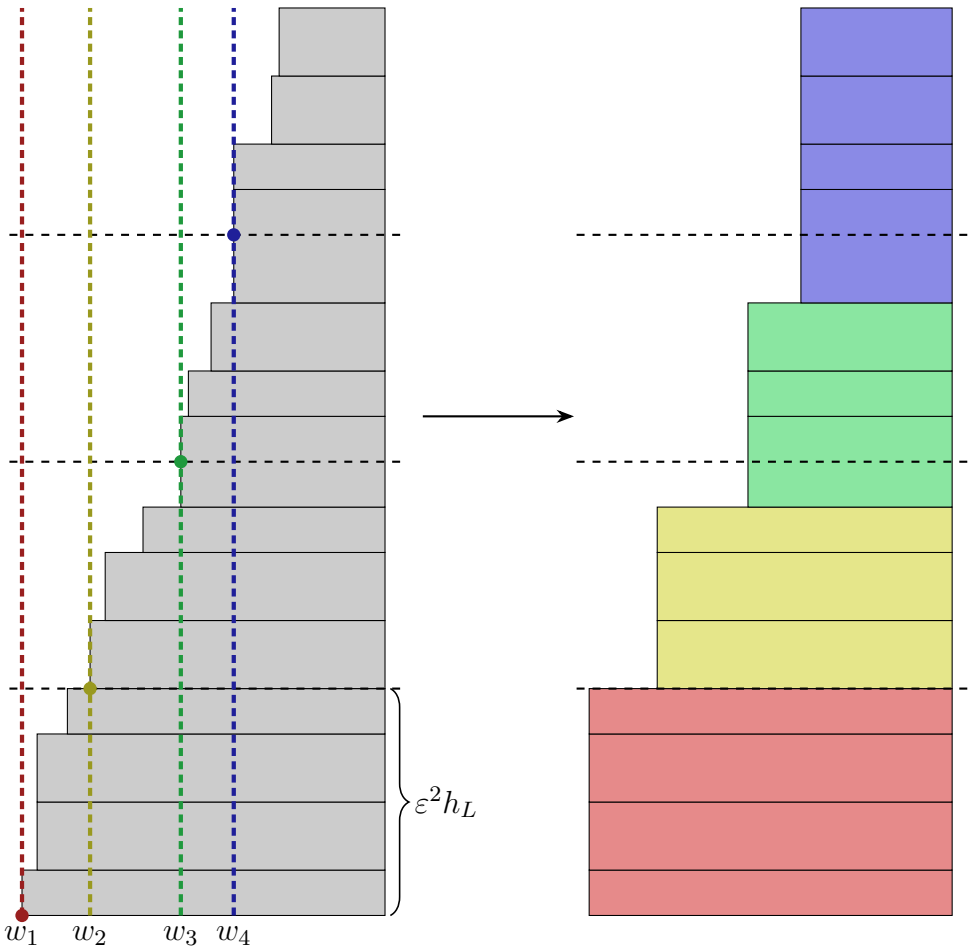


Figure 7.6: Linear grouping of $W^{(L)}$ for $\varepsilon = 1/2$.

On combining (7.1) and (7.2), we get

$$\operatorname{opt}(\widehat{I}) \leq \operatorname{opt}(\widehat{I} - (\widehat{W}_1^{(L)} \cup \widehat{H}_1^{(L)})) + \operatorname{opt}(\widehat{W}_1^{(L)} \cup \widehat{H}_1^{(L)}) < (1+\varepsilon)\operatorname{opt}(I) + 2. \qquad \square$$

7.3.4 Creating Containers

We will use ideas from Kenyon and Rémila's strip packing algorithm [49] to pack $\widehat{I}$ into containers and pack the containers into shelves. In the strip packing problem, we are given a set of rectangular items, and we have to pack them into a bin of width 1 and minimum height.

Since we allow horizontally slicing items in $\widehat{W}$, a packing of $\widehat{W}$ into m bins gives us a packing of $\widehat{W}$ into a strip of height m, and a packing of $\widehat{W}$ into a strip of height h' gives us a packing of $\widehat{W}$ into $\lceil h' \rceil$ bins. Hence, if we denote the optimal strip packing of $\widehat{W}$ by $\operatorname{opt}_{\mathrm{SP}}(\widehat{W})$, then $\operatorname{opt}(\widehat{W}) = \lceil \operatorname{opt}_{\mathrm{SP}}(\widehat{W}) \rceil$. We will now try to compute a near-optimal strip packing of $\widehat{W}$.

Define a horizontal configuration S as a tuple of $1/\varepsilon^2 + 1$ non-negative integers, where $S_0 \in \{0, 1\}$ and $\sum_{j=1}^{1/\varepsilon^2} S_j w_j \leq 1$. For any horizontal line at height y in a strip packing of $\widehat{W}$, the multiset of items intersecting the line corresponds to a configuration. S_0 indicates whether the line intersects items from $\widehat{W}^{(S)}$, and S_j is the number of items from $\widehat{W}_j^{(L)}$ that the line intersects. Let $\mathcal{S}$ be the set of all horizontal configurations. Let $N := |\mathcal{S}|$.

To obtain an optimal packing, we need to determine the height of each configuration. This can be done with the following linear program.

$$\begin{aligned}
\min_{x \in \mathbb{R}^N} \quad & \sum_{S \in \mathcal{S}} x_S & & \\
\text{where} \quad & \sum_{S \in \mathcal{S}} S_j x_S = h(\widehat{W}_j^{(L)}) & & \forall j \in [1/\varepsilon^2] \\
\text{and} \quad & \sum_{S : S_0 = 1} \left(1 - \sum_{j=1}^{1/\varepsilon^2} S_j w_j\right) x_S = a(\widehat{W}^{(S)}) & & \\
\text{and} \quad & x_S \geq 0 & & \forall S \in \mathcal{S}
\end{aligned}$$

Let x^* be an optimal extreme-point solution to the above LP. This gives us a packing where the strip is divided into rectangular regions called *shelves* that are stacked on top of each other. Each shelf has a configuration S associated with it and has height $h(S) := x_S^*$ and contains S_j *containers* of width w_j. Containers of width w_j only contain items from $\widehat{W}_j^{(L)}$, and we call them *type-j* containers. If $S_0 = 1$, S also contains a container of width $1 - \sum_{j=1}^{1/\varepsilon^2} S_j w_j$ that contains small items. We call this container a *type*-0 container. Each container is fully filled with items. Let $w(S)$ denote the width of shelf S, i.e., the sum of widths of all containers in S. Note that if $S_0 = 1$, then $w(S) = 1$. Otherwise, $w(S) = \sum_{j=1}^{1/\varepsilon^2} S_j w_j$.

Lemma 7.14. *x^* contains at most $1/\varepsilon^2 + 1$ positive entries.*

Proof sketch. Follows by applying rank lemma (Corollary 3.8) to the linear program. □

Lemma 7.15. $x_S^* > 0 \implies w(S) > 1/2$.

Proof. Suppose $w(S) \leq 1/2$. Then we could have split S into two parts by making a horizontal cut in the middle and packed the parts side-by-side, reducing the height of the strip by $x_S^*/2$. But that would contradict the fact that x^* is optimal. □

Now treat each shelf S as an item of width $w(S)$ and height $h(S)$. Allow each such item to be sliced using horizontal cuts. This gives us a new set $\widetilde{W}$ of items such that $\widehat{W}$ can be packed inside $\widetilde{W}$. By Lemma 7.15, each item in $\widetilde{W}$ has width more than $1/2$.

By applying an analogous approach to $\widehat{H}$, we get a new set $\widetilde{H}$ of items. Let $\widetilde{I} := \widetilde{W} \cup \widetilde{H}$. We call the shelves of $\widetilde{W}$ *horizontal shelves* and the shelves of $\widetilde{H}$ *vertical shelves*. The containers in horizontal shelves are called *wide containers* and the containers in vertical shelves are called *tall containers*.

Claim 7.16. $a(\widetilde{I}) = a(\widehat{I})$.

Lemma 7.17. *Let $h(\widetilde{W})$ be the sum of heights of all items in $\widetilde{W}$. Let $w(\widetilde{H})$ be the sum of widths of all items in $\widetilde{H}$. Then* $\max(\lceil h(\widetilde{W})\rceil, \lceil w(\widetilde{H})\rceil) \leq \mathrm{opt}(\widehat{I})$.

Proof. Since x^* is the optimal solution to the linear program for strip packing $\widehat{W}$, $h(\widetilde{W}) = \sum_{S\in\mathcal{S}} x^*_S = \mathrm{opt}_{\mathrm{SP}}(\widehat{W})$. Therefore, $\lceil h(\widetilde{W})\rceil = \mathrm{opt}(\widehat{W}) \leq \mathrm{opt}(\widehat{I})$. Similarly, $\lceil w(\widetilde{H})\rceil = \mathrm{opt}(\widehat{H}) \leq \mathrm{opt}(\widehat{I})$. □

7.3.5 Packing Shelves Into Bins

So far, we have packed $\widehat{I}$ into shelves $\widetilde{W}$ and $\widetilde{H}$. We will now use `greedyPack`$(\widetilde{W}, \widetilde{H})$ to pack the shelves into bins. By Claim 7.5, we get a 2-stage packing of $\widetilde{W} \cup \widetilde{H}$ into m bins, where we make at most $m-1$ horizontal cuts in $\widetilde{W}$ and at most $m-1$ vertical cuts in $\widetilde{H}$.

By Lemma 7.14, we get a packing of $m + 1/\varepsilon^2$ horizontal shelves and $m + 1/\varepsilon^2$ vertical shelves into m bins.

By Theorem 7.11, Lemma 7.17, and Claim 7.16, we get that

$$m \leq \max\left(\lceil h(\widetilde{W})\rceil, \lceil w(\widetilde{H})\rceil, \frac{4}{3}a(\widetilde{I}) + \frac{8}{3}\right) \leq \frac{4}{3}\,\mathrm{opt}(\widehat{I}) + \frac{8}{3}.$$

7.3.6 Packing Items Into Containers

We will now try to pack a large subset of items into the containers. See Fig. 7.7 for an example output.

Lemma 7.18. *Let P be a packing of $\widetilde{I}$ into m bins, where we sliced horizontal shelves by making at most $m-1$ horizontal cuts and sliced vertical shelves by making at most $m-1$ vertical cuts. Then we can pack a large subset of items I into the containers in P such that the unpacked items from W have area less than*

$$\varepsilon h(\widetilde{W}) + \delta_H(1+\varepsilon)(m + 1/\varepsilon^2),$$

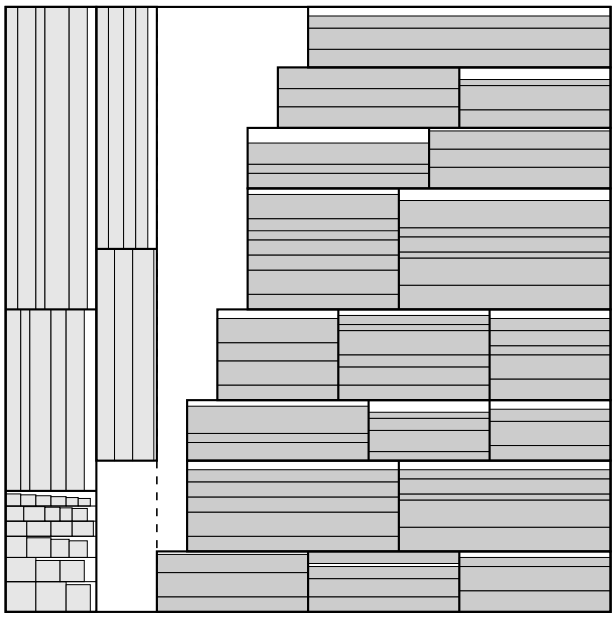


Figure 7.7: A type-1 bin in the packing of $\widehat{I}$ computed by `skewed4Pack`. The packing contains 5 tall containers in 2 tall shelves and 18 wide containers in 8 wide shelves.

and the unpacked items from H have area less than

$$\varepsilon w(\widetilde{H}) + \delta_W(1+\varepsilon)(m + 1/\varepsilon^2).$$

Proof. For each $j \in [1/\varepsilon^2]$, number the type-j wide containers arbitrarily, and number the items in $W_j^{(L)}$ arbitrarily. Now greedily assign items from $W_j^{(L)}$ to the first container C until the total height of the items exceeds $h(C)$. Then move to the next container and repeat. As per the constraints of the linear program, all items in $W_j^{(L)}$ will get assigned to some type-j wide container. Similarly, number the type-0 wide containers arbitrarily and number the items in $W^{(S)}$ arbitrarily. Greedily assign items from $W^{(S)}$ to the first container C until the total area of the items exceeds $a(C)$. Then move to the next container and repeat. As per the constraints of the linear program, all items in $W^{(S)}$ will get assigned to some type-0 wide container. Similarly, assign all items from H to tall containers.

Let C be a type-j wide container and J be the items assigned to it. If we discard the last item from J, then the items can be packed into C. The area of the discarded item is at most $w(C)\delta_H$. Let C be a type-0 wide container and J be the items assigned to it. Arrange the items in J in decreasing order of height and pack the largest prefix $J' \subseteq J$ into C using NFDW (Next-Fit Decreasing Width), which is an analogue of NFDH with the coordinate axes swapped.

Discard the items $J - J'$. By Lemma 3.4, $a(J - J') < \varepsilon h(C) + \delta_H w(C) + \varepsilon\delta_H$. Therefore, for a horizontal shelf S, the total area of discarded items is less than $\varepsilon h(S) + \delta_H(1 + \varepsilon)$.

After slicing the shelves in $\widetilde{I}$ to get P, we get at most $m + 1/\varepsilon^2$ horizontal shelves and at most $m + 1/\varepsilon^2$ vertical shelves. Therefore, the total area of discarded items from W is less than

$$\varepsilon h(\widetilde{W}) + \delta_H(1 + \varepsilon)(m + 1/\varepsilon^2),$$

and the total area of discarded items from H is less than

$$\varepsilon w(\widetilde{H}) + \delta_W(1 + \varepsilon)(m + 1/\varepsilon^2).$$ □

We will pack the discarded items into new bins using NFDH (or NFDW), and NFDH always outputs a 2-stage packing. Since `greedyPack` outputs a 2-stage packing of the shelves and the packing of items into the shelves is a 2-stage packing, the bin packing of non-discarded items is a 4-stage packing.

7.3.7 Summary

A summary of `skewed4Pack` is given in Algorithm 18.

Algorithm 18 `skewed4Pack`$_\varepsilon(I)$: Packs items I into square bins of side length 1, where each item in I has width at most δ_W or height at most δ_H.

1: Let $W := \{i \in I : h(i) \leq \delta_H\}$ and $H := I - W$.
2: Compute $\widehat{I}$ using linear grouping with parameter ε as per Section 7.3.3.
3: Create shelves $\widetilde{I}$ from items $\widehat{I}$ as per Section 7.3.4.
4: Pack $\widetilde{I}$ into bins using `greedyPack`.
5: Pack a large subset of I into the shelves using Lemma 7.18. Let W^d be the unpacked items from W and H^d be the unpacked items from H.
6: Pack W^d into new bins using NFDH.
7: Pack H^d into new bins using NFDW.

Lemma 7.19. *Let I be a set of rectangular items where each item has height at most δ. Then the number of bins required by NFDH to pack I is less than $(2a(I) + 1)/(1 - \delta)$.*

Proof. The bin packing version of NFDH first packs I into shelves and then packs the shelves into bins using Next-Fit. Let H be the sum of heights of all the shelves. By Lemma 3.3, $H < 2a(I) + \delta$. By Lemma 3.2, the number of bins is less than $1 + H/(1 - \delta) < (2a(I) + 1)/(1 - \delta)$. □

Theorem 7.2. `skewed4Pack`$_\varepsilon(I)$ *outputs a 4-stage packing of I in time $O((1/\varepsilon)^{O(1/\varepsilon)} + n \log n)$.*

Proof. `greedyPack` outputs a 2-stage packing, so `skewed4Pack` outputs a 4-stage packing.

Linear grouping takes $O(n \log n + 1/\varepsilon^2)$ time. Computing the shelves requires solving a linear program in at most $2(1/\varepsilon^2)^{1/\varepsilon}$ variables and $1 + 1/\varepsilon^2$ constraints. `greedyPack` takes $O(n \log n)$ time. Packing I into containers takes $O(n \log n)$ time. NFDH and NFDW take $O(n \log n)$ time. □

Theorem 7.3. *Let I be a set of items where each item has width at most δ_W or height at most δ_H. Then $\texttt{skewed4Pack}_\varepsilon(I)$ uses less than $\alpha(1+\varepsilon)\operatorname{opt}(I) + 2\beta$ bins, where*

$$\begin{aligned}
\Delta &:= \frac{1}{2}\left(\frac{\delta_H}{1-\delta_H} + \frac{\delta_W}{1-\delta_W}\right) \\
\alpha &:= \frac{4}{3}(1+4\Delta) + 4\varepsilon\left(1 + \frac{7\Delta}{3}\right) \le \frac{4}{3}(1+4\Delta)(1+3\varepsilon) \\
\beta &:= \frac{2\Delta(1+\varepsilon)}{\varepsilon^2} + \frac{10}{3} + \frac{19\Delta}{3} + \frac{16\Delta\varepsilon}{3}.
\end{aligned}$$

Proof. Suppose `greedyPack` uses at most m bins. Then by Theorem 7.11,

$$m \le 4\operatorname{opt}(\widehat{I})/3 + 8/3.$$

Let W^d and H^d be the unpacked items from W and H, respectively. By Lemmas 7.17 and 7.18,

$$a(W^d) < \varepsilon \operatorname{opt}(\widehat{I}) + \delta_H(1+\varepsilon)(m + 1/\varepsilon^2),$$

$$a(H^d) < \varepsilon \operatorname{opt}(\widehat{I}) + \delta_W(1+\varepsilon)(m + 1/\varepsilon^2).$$

By Lemma 7.19, the number of bins used by $\texttt{skewed4Pack}_\varepsilon(I)$ is less than

$$\begin{aligned}
&m + \frac{2a(W^d)+1}{1-\delta_H} + \frac{2a(H^d)+1}{1-\delta_W} \\
&\le (1 + 4\Delta(1+\varepsilon))m + 4\varepsilon(1+\Delta)\operatorname{opt}(\widehat{I}) + 2(1+\Delta) + 4\Delta(1+\varepsilon)/\varepsilon^2 \\
&\le \alpha \operatorname{opt}(\widehat{I}) + 2(\beta - 1) < \alpha(1+\varepsilon)\operatorname{opt}(I) + 2\beta. \qquad \text{(by Lemma 7.13)}
\end{aligned}$$

□

7.4 APoG for the Rotational Case

Theorem 7.20. *For a set I of rectangular items, let $\operatorname{opt}^{\mathrm{nr}}(I)$ and $\operatorname{opt}^{\mathrm{r}}(I)$ be the minimum number of bins needed to pack I in the non-rotational and rotational versions, respectively.*

Let $\mathrm{opt}_g^{\mathrm{nr}}(I)$ *and* $\mathrm{opt}_g^{\mathrm{r}}(I)$ *be the minimum number of guillotinable bins needed to pack* I *in the non-rotational and rotational versions, respectively.*

Let $\mathcal{S}$ *be a family of inputs that is closed under rotation, i.e., for a set* $I \in \mathcal{S}$ *of items, if we rotate some items in* I *to get a set* J *of items, then* $J \in \mathcal{S}$*. Let* $\mathrm{APoG}^{\mathrm{nr}}$ *and* $\mathrm{APoG}^{\mathrm{r}}$ *be the APoG for the non-rotational and rotational versions, respectively, restricted to the family* $\mathcal{S}$*. Then* $\mathrm{APoG}^{\mathrm{r}} \le \mathrm{APoG}^{\mathrm{nr}}$*.*

Proof. Let I be any set of items in $\mathcal{S}$. Let K be the corresponding rotated items in the optimal rotational packing of I, i.e., $\mathrm{opt}^{\mathrm{r}}(I) = \mathrm{opt}^{\mathrm{nr}}(K)$. Then

$$\begin{aligned}\mathrm{opt}_g^{\mathrm{r}}(I) &\le \mathrm{opt}_g^{\mathrm{nr}}(K)\\ &\le \mathrm{APoG}^{\mathrm{nr}}\,\mathrm{opt}^{\mathrm{nr}}(K) + o(\mathrm{opt}^{\mathrm{nr}}(K))\\ &= \mathrm{APoG}^{\mathrm{nr}}\,\mathrm{opt}^{\mathrm{r}}(I) + o(\mathrm{opt}^{\mathrm{r}}(I)).\end{aligned}$$

Since this is true for all $I \in \mathcal{S}$, we get $\mathrm{APoG}^{\mathrm{r}} \le \mathrm{APoG}^{\mathrm{nr}}$. □

Assume without loss of generality that bins have width and height at least 1. The class of (δ, δ)-skewed rectangles is closed under rotation, so by Theorem 7.20, the APoG for the rotational case is upper-bounded by

$$\frac{4}{3}\left(1 + \frac{4\delta}{1-\delta}\right).$$

When δ is very small, this is close to 4/3.

Chapter 8

Almost-Optimal Bin Packing of Skewed Rectangles

For a constant $\delta > 0$, a rectangle is said to be δ-skewed iff either its width is at most δ or its height is at most δ. We give an approximation algorithm for bin packing δ-skewed rectangles where the algorithm's AAR approaches 1 as δ approaches 0. Formally, we give an algorithm for 2D GBP, called `skewedCPack` (abbreviates *skewed compartmental packing*), that accepts a parameter ε, and we show that for every constant $\varepsilon \in (0, 1)$, there exists a constant $\delta \in (0, \varepsilon)$ such that the algorithm has an AAR of $1+\varepsilon$ when all items in the input are δ-skewed rectangles.

Our result shows that the approximability of the δ-skewed case is very different from the general 2BP problem, since it is NP-hard to obtain an asymptotic approximation ratio better than $1 + 1/2196$ for general 2BP [21].

The best-known AAR for 2D GBP is $1+\ln(1.5)+\varepsilon \approx 1.405+\varepsilon$. Our result indicates that to improve upon algorithms for 2D GBP, we should focus on big rectangles, i.e., rectangles whose width and height are both more than a constant δ.

Overview of the Algorithm

`skewedCPack` takes a set I of items as input and has the following outline:

1. Invoke the subroutine `round`(I) (described in Section 8.1). `round`(I) removes some items $I_{\text{med}} \subseteq I$ of low total area and rounds up the width or height of each remaining item so that the resulting items $\widetilde{I}$ have special properties that help us pack them easily.

2. Compute the optimal *fractional compartmental* bin-packing of $\widetilde{I}$ (we will define *compartmental* and *fractional* later).

3. Use this packing of $\widetilde{I}$ to obtain a packing of I that uses slightly more number of bins.

Let $\mathrm{opt}(I)$ be the minimum number of bins needed to pack I. To bound the AAR of `skewedCPack`, we will prove a structural theorem in Section 8.2, i.e., we will prove that the optimal fractional compartmental packing of $\widetilde{I}$ uses close to $\mathrm{opt}(I)$ bins.

We will focus on the case where the items cannot be rotated, so we will assume without loss of generality that the bin is a square of side length 1. In Section 8.4, we show how to extend `skewedCPack` to the case where the items can be rotated by 90°.

Organization of the Chapter

- In Section 8.1, we describe the subroutine `round` and define *fractional* packing.
- In Section 8.2, we define *compartmental* packing and prove the structural theorem.
- In Section 8.3, we describe the `skewedCPack` algorithm.
- In Section 8.4, we show how to extend `skewedCPack` to handle item rotations.

8.1 Classifying and Rounding Items

In this section, we will describe the algorithm $\texttt{round}(I)$. $\texttt{round}(I)$ returns a pair $(\widetilde{I}, I_{\mathrm{med}})$, where $\widetilde{I}$ is called the set of rounded items and $I_{\mathrm{med}} \subseteq I$ is called the set of medium items. We will show that $a(I_{\mathrm{med}}) \leq \varepsilon a(I)$ and $\widetilde{I}$ is obtained by rounding up the width or height of each item in $I - I_{\mathrm{med}}$.

We assume that $\varepsilon \leq 1/2$ and that $\varepsilon^{-1} \in \mathbb{Z}$.

8.1.1 Removing Medium Items

We will choose $I_{\mathrm{med}} \subseteq I$ such that for two constants ε_2 and ε_1, no item in $I - I_{\mathrm{med}}$ has its width or height in the interval $(\varepsilon_2, \varepsilon_1]$ and $\varepsilon_2 \ll \varepsilon_1 < 1$ (we will soon precisely define the meaning of $\varepsilon_2 \ll \varepsilon_1$). For `skewedCPack` to work, we require $\delta \leq \varepsilon_2$.

Definition 8.1. *Let $\mu_0 \in (0,1]$ be a constant and let $f : (0,1] \mapsto (0,1]$ be a function such that $\forall x \in (0,1], f(x) < x$. Let $T := \lceil 2/\varepsilon \rceil$. For $t \in [T]$, define $\mu_t := f(\mu_{t-1})$ and define*

$$J_t := \{i \in I : w(i) \in (\mu_t, \mu_{t-1}] \text{ or } h(i) \in (\mu_t, \mu_{t-1}]\}.$$

Define $\texttt{removeMedium}(I, \varepsilon, f, \mu_0)$ *as the tuple (J_r, μ_r, μ_{r-1}), where $r := \mathrm{argmin}_{t=1}^{T} a(J_t)$.*

Lemma 8.1. *Let* $(I_{\text{med}}, \varepsilon_2, \varepsilon_1) := \texttt{removeMedium}(I, \varepsilon, f, \mu_0)$. *Then* $a(I_{\text{med}}) \leq \varepsilon a(I)$.

Proof. Each item belongs to at most 2 sets J_t. Therefore,

$$a(I_{\text{med}}) = \min_{t=1}^{T} a(J_t) \leq \frac{1}{T}\sum_{t=1}^{T} a(J_t) \leq \frac{2}{\lceil 2/\varepsilon \rceil} a(I) \leq \varepsilon a(I). \qquad \square$$

No item in $I - I_{\text{med}}$ has width or height in the interval $(\varepsilon_2, \varepsilon_1]$.
Let $\mu_0 = \varepsilon$. So, $\varepsilon_1 \leq \varepsilon$ and $\varepsilon_2 := f(\varepsilon_1)$. We choose f to be

$$f(x) := \frac{\varepsilon x}{104(1 + 1/(\varepsilon x))^{2/x-2}}. \tag{8.1}$$

We will explain this choice later in Section 8.3.4. Intuitively, such an f ensures that $\varepsilon_2 = f(\varepsilon_1) \ll \varepsilon_1$. Note that f is independent of I, so ε_1 and ε_2 are constants. Also note that $x^{-1} \in \mathbb{Z} \implies f(x)^{-1} \in \mathbb{Z}$, so $\varepsilon_1^{-1}, \varepsilon_2^{-1} \in \mathbb{Z}$.

8.1.2 Classifying Items

Classify the items in $I - I_{\text{med}}$ into three disjoint classes:

- Wide items: $W := \{i \in I : w(i) > \varepsilon_1 \text{ and } h(i) \leq \varepsilon_2\}$.
- Tall items: $H := \{i \in I : w(i) \leq \varepsilon_2 \text{ and } h(i) > \varepsilon_1\}$.
- Small items: $S := \{i \in I : w(i) \leq \varepsilon_2 \text{ and } h(i) \leq \varepsilon_2\}$.

8.1.3 Linear Grouping

We will now use *linear grouping* [26, 49] to round up the widths of items in W and the heights of items in H. Arrange the items of W in decreasing order of width and stack them one-over-the-other (i.e., the widest item in W is at the bottom). Let h_L be the height of the stack. Let $y(i)$ be the y-coordinate of the bottom edge of item i. Split the stack into sections of height $\varepsilon\varepsilon_1 h_L$ each. For $j \in [1/\varepsilon\varepsilon_1]$, let w_j be the width of the widest item intersecting the j^{th} section from the bottom, i.e.,

$$w_j := \max(\{w(i) : i \in W \text{ and } (y(i), y(i) + h(i)) \cap ((j-1)\varepsilon\varepsilon_1 h_L, j\varepsilon\varepsilon_1 h_L) \neq \emptyset\}).$$

Round up the width of each item i to the smallest w_j that is at least $w(i)$. Let W_j be the items whose width got rounded to w_j and let $\widetilde{W}_j$ be the resulting rounded items. (There may be ties,

i.e., there may exist $j_1 < j_2$ such that $w_{j_1} = w_{j_2}$. In that case, define $W_{j_2} := \widetilde{W}_{j_2} = \emptyset$. This ensures that all W_j are disjoint.) Let $\widetilde{W} := \bigcup_j \widetilde{W}_j$.

Define $\widetilde{H}$ analogously. Let $\widetilde{I} := \widetilde{W} \cup \widetilde{H} \cup S$.

Claim 8.2. *Items in $\widetilde{W}$ have at most $1/\varepsilon\varepsilon_1$ distinct widths. Items in $\widetilde{H}$ have at most $1/\varepsilon\varepsilon_1$ distinct heights.*

Definition 8.2 (Fractional packing)**.** *Suppose we are allowed to slice wide items in $\widetilde{I}$ using horizontal cuts, slice tall items in $\widetilde{I}$ using vertical cuts and slice small items in $\widetilde{I}$ using both horizontal and vertical cuts. For any $\widetilde{X} \subseteq \widetilde{I}$, a bin packing of the slices of $\widetilde{X}$ is called a* fractional packing *of $\widetilde{X}$. The optimal fractional packing of $\widetilde{X}$ is denoted by* $\operatorname{fopt}(\widetilde{X})$.

Lemma 8.3. $\operatorname{fopt}(\widetilde{I}) < (1+\varepsilon)\operatorname{opt}(I) + 2$.

Proof. Consider the optimal packing of I. To convert this to a packing of $\widetilde{I} - (\widetilde{W}_1 \cup \widetilde{H}_1)$, unpack W_1 and H_1, and for each $j \in [1/\varepsilon\varepsilon_1 - 1]$, pack $\widetilde{W}_{j+1}$ in the place of W_j and pack $\widetilde{H}_{j+1}$ in the place of H_j, possibly after slicing the items. Therefore,

$$\operatorname{fopt}(\widetilde{I} - (\widetilde{W}_1 \cup \widetilde{H}_1)) \leq \operatorname{opt}(I). \tag{8.2}$$

We can pack $\widetilde{H}_1$ in a bin by stacking the items side-by-side on the base of bins. We can pack $\widetilde{W}_1$ in a bin by stacking the items one-over-the-other. Let w_L be the total width of items in $\widetilde{H}$. The number of bins used is $\lceil \varepsilon\varepsilon_1 h_L \rceil + \lceil \varepsilon\varepsilon_1 w_L \rceil$. Also,

$$\operatorname{opt}(I) \geq \operatorname{opt}(W \cup H) \geq a(W) + a(H) \geq \varepsilon_1(h_L + w_L).$$

Therefore,

$$\operatorname{fopt}(\widetilde{W}_1 \cup \widetilde{H}_1) \leq \lceil \varepsilon\varepsilon_1 h_L \rceil + \lceil \varepsilon\varepsilon_1 w_L \rceil < \varepsilon \operatorname{opt}(I) + 2. \tag{8.3}$$

On combining (8.2) and (8.3), we get

$$\operatorname{fopt}(\widetilde{I}) \leq \operatorname{fopt}(\widetilde{I} - (\widetilde{W}_1 \cup \widetilde{H}_1)) + \operatorname{fopt}(\widetilde{W}_1 \cup \widetilde{H}_1) < (1+\varepsilon)\operatorname{opt}(I) + 2. \qquad \square$$

8.2 Structural Theorem

In this section, we will define compartmental packing and we will prove the structural theorem, which says that the number of bins in the optimal fractional compartmental packing of $\widetilde{I}$ is roughly equal to $\operatorname{fopt}(\widetilde{I})$.

For any rectangle i packed in a bin, let $x_1(i)$ and $x_2(i)$ denote the x-coordinates of its left and right edges, respectively, and let $y_1(i)$ and $y_2(i)$ denote the y-coordinates of its bottom and top edges, respectively. Let R be the set of distinct widths of items in $\widetilde{W}$. Given the way we rounded items, $|R| \leq 1/\varepsilon\varepsilon_1$.

Recall that $\varepsilon_1 \leq \varepsilon \leq 1/2$.

8.2.1 Discretizing Horizontal Positions

We will show that given a fractional packing of items in a bin, we can remove a small fraction of tall and small items and shift the remaining items leftwards so that the left and right edges of each wide item belong to a constant-sized set $\mathcal{T}$.

Let $T_0 := \{0\}$ and $t_0 := 1$. For any $j > 0$, define

- $t_j := (1 + 1/\varepsilon\varepsilon_1)^{2j}$.
- $\delta_j := \varepsilon\varepsilon_1/t_{j-1}$.
- $S_j := T_{j-1} \cup \{k\delta_j : k \in \mathbb{Z} \text{ and } 0 \leq k < 1/\delta_j\}$.
- $T_j := \{x + y : x \in S_j \text{ and } y \in R \cup \{0\}\}$.

Observation 8.4. *For all $j > 0$, we have $T_{j-1} \subseteq S_j \subseteq T_j$ and $\delta_j^{-1} \in \mathbb{Z}$.*

Lemma 8.5. *For all $j \geq 0$, $|T_j| \leq t_j$.*

Proof. We will prove this by induction. The base case holds because $|T_0| = t_0 = 1$.

Now assume $|T_{j-1}| \leq t_{j-1}$. Then

$$|T_j| \leq (|R| + 1)|S_j| \leq \left(\frac{1}{\varepsilon\varepsilon_1} + 1\right)\left(|T_{j-1}| + \frac{1}{\delta_j}\right) \leq \left(\frac{1}{\varepsilon\varepsilon_1} + 1\right)^2 t_{j-1} = t_j.$$

Hence, by mathematical induction, $|T_j| \leq t_j$ for all $j \geq 0$. □

Define $\mathcal{T} := T_{1/\varepsilon_1 - 1}$. Therefore, $|\mathcal{T}| \leq t_{1/\varepsilon_1 - 1} = (1 + 1/\varepsilon\varepsilon_1)^{2/\varepsilon_1 - 2}$.

Lemma 8.6. *Given a fractional packing of items $\widetilde{J} \subseteq \widetilde{I}$ into a bin, we can remove tall and small items of total area less than ε and shift some of the remaining items to the left such that for every wide item i, we get $x_1(i), x_2(i) \in \mathcal{T}$.*

Proof. We will describe an algorithm for such a transformation.

For wide items u and v, we say that $u \prec v$ iff the right edge of u is to the left of the left edge of v. Formally $u \prec v \iff x_2(u) \leq x_1(v)$. We call u a *predecessor* of v. Note that the relation $\prec$ is transitive. A sequence $[i_1, i_2, \ldots, i_k]$ such that $i_1 \prec i_2 \prec \ldots \prec i_k$ is called a *chain* ending at i_k. For a wide item i, define level(i) as the number of items in the longest chain ending at i. Formally,

$$\text{level}(i) := \begin{cases} 1 & \text{if } i \text{ has no predecessors} \\ 1 + \max\limits_{j \prec i} \text{level}(j) & \text{otherwise} \end{cases}.$$

Let W_j be the items at level j, i.e., $W_j := \{i : \text{level}(i) = j\}$.

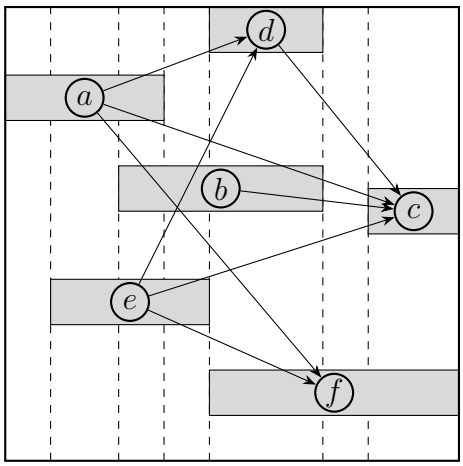


Figure 8.1: Example illustrating the $\prec$ relationship between wide items in a bin. An edge is drawn from u to v iff $u \prec v$. Here $W_1 = \{a, e, b\}$, $W_2 = \{d, f\}$ and $W_3 = \{c\}$.

Note that the level of an item can be at most $1/\varepsilon_1 - 1$, since each wide item has width more than ε_1.

Our algorithm will proceed in stages, where in the j^{th} stage, we apply two transformations to the items in the bin. In the first transformation, called *strip-removal*, we will remove some tall and small items. In the second transformation, called *compaction*, we will first shift some tall and small items leftwards and then shift each item in W_j leftwards.

We will maintain the following invariant throughout the algorithm:

Invariant: *after k stages, for each $j \in [k]$, each item $i \in W_j$ has $x_1(i) \in S_j$ (and hence $x_2(i) \in T_j$).* Note that the invariant is trivially true for $k = 0$.

Definition 8.3 (Strip-removal)**.** *In the j^{th} stage, for each $x \in T_{j-1}$, consider a strip of width*

δ_j *and height 1 in the bin whose left edge has coordinate* x*. Discard the slices of tall and small items inside the strips. This transformation is called* strip-removal.

Lemma 8.7. *Items discarded from a bin by strip-removal (across all stages) have total area less than* ε*.*

Proof. In the j^{th} stage, we create $|T_{j-1}|$ strips, and each strip has total area at most δ_j. Therefore, the area discarded in the j^{th} stage is at most $|T_{j-1}|\delta_j \leq t_{j-1}\delta_j = \varepsilon\varepsilon_1$. Since there can be at most $1/\varepsilon_1 - 1$ stages, we discard an area of less than ε across all stages. □

Definition 8.4 (Compaction)**.** *In the* j^{th} *stage, move all tall and small items as much towards the left as possible (imagine a gravitational force acting leftwards on the tall and small items) while keeping the wide items fixed. Then move each wide item* $i \in W_j$ *leftwards till* $x_1(i) \in S_j$*. This transformation is called* compaction*.*

Lemma 8.8. *Compaction always succeeds, i.e., in the* j^{th} *stage, while moving item* $i \in W_j$ *leftwards, no other item will block its movement.*

Proof. Let $i \in W_j$. Let z be the x-coordinate of the left edge of the strip immediately to the left of item i, i.e., $z := \max(\{x \in T_{j-1} : x \leq x_1(i)\})$.

For any wide item i', we have $x_2(i') \leq x_1(i) \iff i' \prec i \iff \text{level}(i') \leq j-1$. By our invariant, we get

$$\text{level}(i') \leq j-1 \implies x_2(i') \in T_{j-1} \implies x_2(i') \leq z.$$

Therefore, for every wide item i', $x_2(i') \notin (z, x_1(i)]$.

In the j^{th} strip-removal, we cleared the strip $[z, z+\delta_j] \times [0,1]$. If $x_1(i) \in [z, z+\delta_j]$, then i can freely move to z, and $z \in T_{j-1} \subseteq S_j$. Since no wide item has its right edge in $(z, x_1(i)]$, if $x_1(i) > z + \delta_j$, all the tall and small items in $[z+\delta_j, x_1(i)]$ will move leftwards by at least δ_j during compaction. Hence, there would be an empty space of width at least δ_j to the left of item i (see Fig. 8.2). Therefore, we can move i leftwards to make $x_1(i)$ a multiple of δ_j, and then $x_1(i)$ would belong to S_j. □

Since compaction in the j^{th} stage would force $x_1(i)$ to belong to S_j for each $i \in W_j$, the invariant is maintained after each stage. Therefore, after $1/\varepsilon_1 - 1$ stages, we get that for each wide item i, $x_1(i) \in S_{1/\varepsilon_1 - 1} \subseteq \mathcal{T}$ and $x_2(i) \in T_{1/\varepsilon_1 - 1} = \mathcal{T}$. □

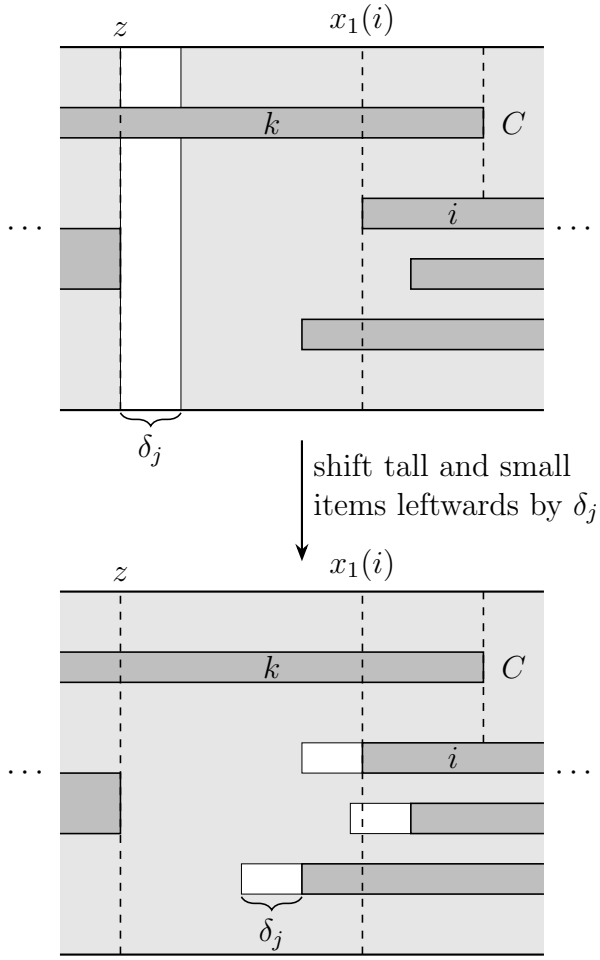


Figure 8.2: This figure shows a region in the bin in the vicinity of item $i \in W_j$. It illustrates how shifting tall and small items during compaction in the j^{th} stage creates a free space of width δ to the left of some wide items, including i. Wide items are shaded dark and the lightly shaded region potentially contains tall and small items. Note that some tall and small items in the region C may be unable to shift left because item k is blocking them. All other tall and small items in this figure to the right of z can shift left by δ_j.

8.2.2 Creating Compartments

Definition 8.5 (Compartmental packing)**.** *Consider a bin with some items packed into it. A* compartment C *is defined as a rectangular region in the bin satisfying the following properties:*

- $x_1(C), x_2(C) \in \mathcal{T}$.
- $y_1(C), y_2(C)$ *are multiples of* $\varepsilon_{\text{cont}} := \varepsilon\varepsilon_1/6|\mathcal{T}|$.
- C *does not contain both wide items and tall items.*

- *If C contains tall items, then $x_1(C)$ and $x_2(C)$ are consecutive values in $\mathcal{T}$.*

If a compartment C contains a wide item, it is called a wide compartment*. Otherwise it is called a* tall compartment*.*

A packing of items $\widetilde{J}$ into a bin is said to be compartmental *iff there is a set of non-overlapping* compartments *in the bin such that each wide or tall item lies completely inside some compartment, and there are at most $n_W := 3(1/\varepsilon_1 - 1)|\mathcal{T}| + 1$ wide compartments in the bin and there are at most $n_H := (1/\varepsilon_1 - 1)|\mathcal{T}|$ tall compartments in the bin.*

A packing of items into bins is called compartmental iff each bin in the packing is compartmental.

Lemma 8.9. *Let there be a set I of rectangles packed inside a bin. Then there is a polynomial-time algorithm that can decompose the empty space in the bin into at most $3|I| + 1$ rectangles by making horizontal cuts only.*

Proof. Extend the top and bottom edge of each rectangle leftwards and rightwards till they hit another rectangle or an edge of the bin. This decomposes the empty region into rectangles R. See Fig. 8.3.

For each rectangle $i \in I$, the top edge of i is the bottom edge of a rectangle in R, the bottom edge of i is the bottom edge of two rectangles in R. Apart from possibly the rectangle in R whose bottom edge is at the bottom of the bin, the bottom edge of every rectangle in R is either the bottom or top edge of a rectangle in I. Therefore, $|R| \leq 3|I| + 1$. □

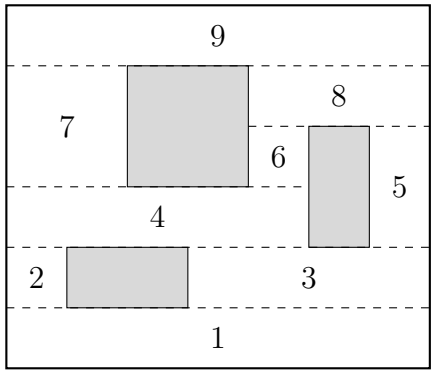
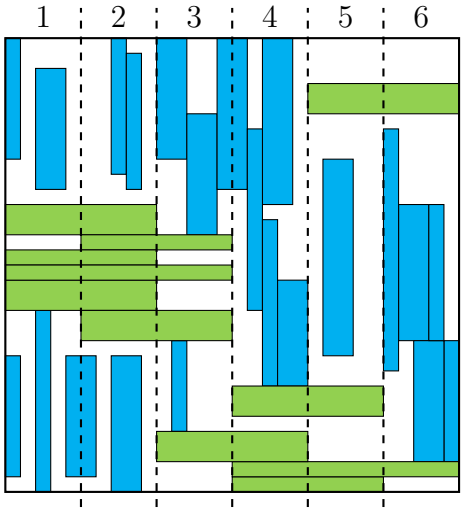


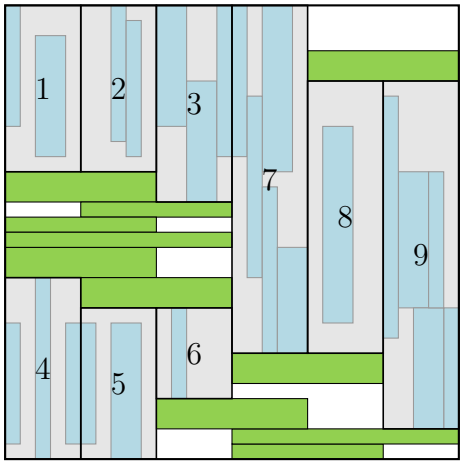
Figure 8.3: Using horizontal cuts to partition the empty space around the 3 items into 9 rectangular regions.

Lemma 8.10. *Let $\widetilde{J}$ be a packing of items into a bin such that for each wide item i, $x_1(i), x_2(i) \in \mathcal{T}$. Then by removing wide and small items of area less than ε, we can get a compartmental packing of the remaining items.*

Proof. Draw vertical lines in the bin at the x-coordinates in $\mathcal{T} - \{0\}$. This splits the bin into $|\mathcal{T}|$ columns (see Fig. 8.4a). Each column has 0 or more wide items crossing it. These wide items divide the column into cells. A cell is called tall iff it contains a tall item (see Fig. 8.4b). There can be at most $1/\varepsilon_1 - 1$ tall cells in a column, so there can be at most $(1/\varepsilon_1 - 1)|\mathcal{T}|$ tall cells in the bin.



(a) A packing of items in a bin. Wide items are green and tall items are blue. Draw vertical lines at x-coordinates from $\mathcal{T} - \{0\}$. They divide the bin into columns. In this figure, we have 6 columns.



(b) Wide items divide each column into *cells*. Each cell containing a tall item is called a *tall cell*. There are 9 tall cells in this figure, which are shaded gray.

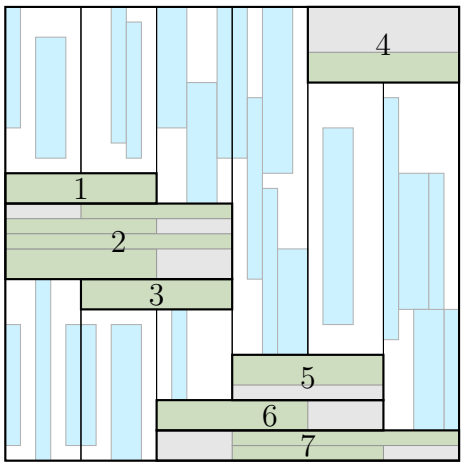
Figure 8.4: Creating tall cells in a bin

By Lemma 8.9, we can use horizontal cuts to partition the space outside tall cells into at most $3(1/\varepsilon_1 - 1)|\mathcal{T}| + 1$ rectangular regions (this can slice some wide items). See Fig. 8.5a. If a region contains a wide item, call it a box.

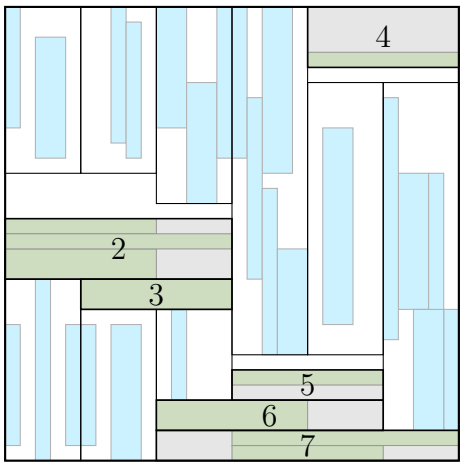
For each box i, slice and discard some items from the bottom of the box and increase $y_1(i)$ so that it becomes a multiple of $\varepsilon_{\text{cont}}$. Then slice and discard some items from the top of the box and reduce $y_2(i)$ so that it becomes a multiple of $\varepsilon_{\text{cont}}$. The total area of items discarded is less than $2\varepsilon_{\text{cont}}$. If i continues to contain a wide item, it becomes a wide compartment. Now all wide items belong to some wide compartment (see Fig. 8.5b).

Each column has 0 or more wide compartments crossing it. These wide compartments divide the column into rectangular regions. Each region that contains a tall item is a tall compartment (see Fig. 8.6).

Therefore, by removing wide and small items of area less than $6|\mathcal{T}|\varepsilon_{\text{cont}}/\varepsilon_1 \leq \varepsilon$, we get a



(a) Partition the space outside tall cells into rectangular regions by extending the horizontal edges of tall cells (see Lemma 8.9). Each rectangular region containing a wide item is called a *box*. There are 7 boxes in this figure, which are shaded gray.



(b) For each box, discard some items and shift horizontal edges to make their y-coordinates multiples of $\varepsilon_{\text{cont}}$. Boxes that continue to contain a wide item are now wide compartments.

Figure 8.5: Obtaining wide compartments

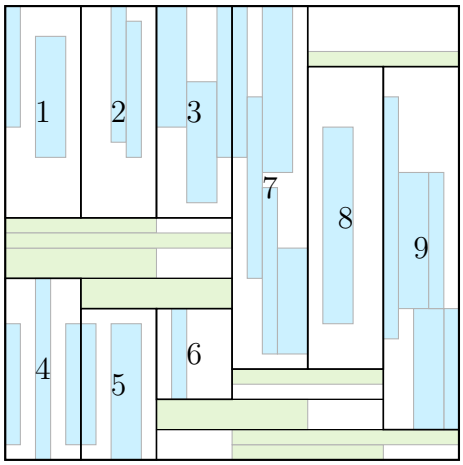


Figure 8.6: Wide compartments divide each column into rectangular regions. Each such region containing a tall item is a tall compartment. There are 9 tall compartments in this figure.

compartmental packing of items where there are at most $(1/\varepsilon_1 - 1)|\mathcal{T}|$ tall compartments and at most $3(1/\varepsilon_1 - 1)|\mathcal{T}| + 1$ wide compartments. □

8.2.3 Existence of Near-Optimal Compartmental Packing

For a set $\widetilde{I}$ of rounded items, define $\text{fcopt}(\widetilde{I})$ as the number of bins in the optimal fractional compartmental packing of $\widetilde{I}$.

Theorem 8.11. *Let $\widetilde{I}$ be a set of δ-skewed rounded items. Then* $\text{fcopt}(\widetilde{I}) < (1+4\varepsilon)\,\text{fopt}(\widetilde{I})+2$.

Proof. Consider a fractional packing of $\widetilde{I}$ into $m := \text{fopt}(\widetilde{I})$ bins. By Lemmas 8.6 and 8.10, in each bin, we can discard items of area at most 2ε from the bin and get a compartmental packing of the remaining items.

Let X be the set of wide and small discarded items and let Y be the set of tall discarded items. For each item $i \in X$, if $w(i) \leq 1/2$, slice it using a horizontal cut in the middle and place the pieces horizontally next to each other to get a new item of width $2w(i)$ and height $h(i)/2$. Repeat until $w(i) > 1/2$. Now pack the items in bins by stacking them one-over-the-other so that for each item $i \in X$, $x_1(i) = 0$. This will require less than $2a(X)+1$ bins, and the packing will be compartmental.

Similarly, we can get a compartmental packing of Y into $2a(Y)+1$ bins. Since $a(X \cup Y) < 2\varepsilon m$, we will require less than $4\varepsilon m + 2$ bins. Therefore, the total number of compartmental bins used to pack $\widetilde{I}$ is less than $(1+4\varepsilon)m + 2$. □

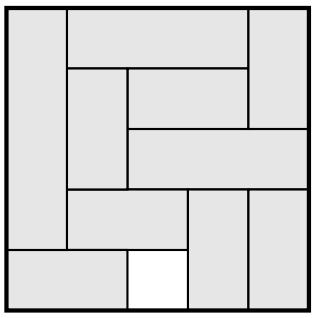
8.3 Packing Rounded Items

Let I be a set of δ-skewed items. In this section, we give an algorithm for computing a near-optimal packing of I, called `skewedCPack`. Roughly, `skewedCPack` first computes $(\widetilde{I}, I_{\text{med}}) := \texttt{round}(I)$. It then computes the optimal fractional compartmental packing of $\widetilde{I}$ by first guessing a packing of empty compartments into bins and then fractionally packing the wide and tall items into the compartments. It then converts the fractional packing of $\widetilde{I}$ to a non-fractional packing of I with only a tiny increase in the number of bins. See Fig. 8.7 for a visual overview of `skewedCPack`.

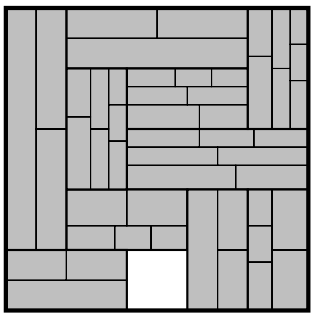
8.3.1 Enumerating Packing of Compartments

We will compute the optimal fractional compartmental packing of $\widetilde{I}$ in two steps. First, for each bin, we will guess the compartments in the bin. Each such packing of compartments into bins is called a *configuration*. Then we will fractionally pack the items into the compartments.

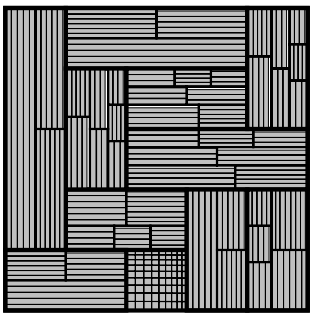
There can be at most $n_W := 3(1/\varepsilon_1 - 1)|\mathcal{T}| + 1$ wide compartments in a bin. Each wide compartment can have $(1/\varepsilon_{\text{cont}})^2$ y-coordinates of the top and bottom edges and at most $|\mathcal{T}|^2/2$



(a) Guess the packing of empty compartments in each bin (Section 8.3.1).



(b) Fractionally pack wide and tall items into compartments. This partitions each compartment into *containers* (Section 8.3.2).



(c) Pack the items non-fractionally (Section 8.3.3).

Figure 8.7: Major steps of `skewedCPack` after `rounding` I.

x-coordinates of the left and right edges, where $\varepsilon_{\text{cont}} := \varepsilon\varepsilon_1/6|\mathcal{T}|$. The rest of the space is for tall compartments. Therefore, the number of configurations is at most

$$n_C := \left((1/\varepsilon_{\text{cont}})^2|\mathcal{T}|^2/2\right)^{n_W} \leq \left(\frac{3|\mathcal{T}|^2}{\varepsilon\varepsilon_1}\right)^{6|\mathcal{T}|/\varepsilon_1} \leq \left(1+\frac{1}{\varepsilon\varepsilon_1}\right)^{\left(1+\frac{1}{\varepsilon\varepsilon_1}\right)^{2/\varepsilon_1+1}}.$$

Since each configuration can have at most n bins, the number of combinations of configurations is at most $(n+1)^{n_C}$.

Therefore, we can iterate over all possible bin packings of empty compartments in $O(n^{n_C})$ time. Let `iterPackings`$(\widetilde{I})$ be an algorithm for this, i.e., `iterPackings`$(\widetilde{I})$ outputs the set of all possible bin packings of empty compartments into at least $\lceil a(\widetilde{I})\rceil$ and at most n bins, where n is the number of items in $\widetilde{I}$.

8.3.2 Packing Items Into Compartments

For each bin packing of empty compartments, we will try to fractionally pack the items into the bins. Formally, let P be a packing of empty compartments into bins. We will create a feasibility linear program, called $\text{FP}(\widetilde{I}, P)$, that is feasible iff wide and tall items in $\widetilde{I}$ can be packed into the compartments in P. If $\text{FP}(\widetilde{I}, P)$ is feasible, then small items can also be fractionally packed since P contains at least $a(\widetilde{I})$ bins.

Let $w'_1, w'_2, \ldots, w'_p$ be the distinct widths of wide compartments in P. Let U_j be the set of wide compartments in P having width w'_j. Let $h(U_j)$ be the sum of heights of the compartments in U_j. By Definition 8.5, we know that $p \leq |\mathcal{T}|^2/2$. Let $w_1, w_2, \ldots, w_r$ be the distinct widths

of items in $\widetilde{W}$ (recall that $\widetilde{W}$ is the set of wide items in $\widetilde{I}$). Let $\widetilde{W}_j$ be the items in $\widetilde{W}$ having width w_j. Let $h(\widetilde{W}_j)$ be the sum of heights of all items in $\widetilde{W}_j$. By Claim 8.2, we get $r \leq 1/\varepsilon\varepsilon_1$.

Let $C := [C_0, C_1, \ldots, C_r]$ be a vector, where $C_0 \in [p]$ and $C_j \in \mathbb{Z}_{\geq 0}$ for $j \in [r]$. C is called a *wide configuration* iff $w(C) := \sum_{j=1}^{r} C_j w_j \leq w'_{C_0}$. Intuitively, a wide configuration C represents a set of wide items that can be placed side-by-side into a compartment of width w'_{C_0}. Let $\mathcal{C}$ be the set of all wide configurations. Then $|\mathcal{C}| \leq p/\varepsilon_1^r$, which is a constant. Let $\mathcal{C}_j := \{C \in \mathcal{C} : C_0 = j\}$.

To pack $\widetilde{W}$ into wide compartments, we must determine the height of each configuration. Let $x \in \mathbb{R}_{\geq 0}^{|\mathcal{C}|}$ be a vector where x_C denotes the height of configuration C. Then $\widetilde{W}$ can be packed into wide compartments according to x iff x is a feasible solution the following feasibility linear program, named $\mathrm{FP}_W(\widetilde{I}, P)$:

$$\begin{aligned}
\sum_{C \in \mathcal{C}} C_j x_C &\geq h(\widetilde{W}_j) \quad \forall j \in [r] && (\widetilde{W}_j \text{ should be covered}) \\
\sum_{C \in \mathcal{C} \text{ and } C_0 = j} x_C &\leq h(U_j) \quad \forall j \in [p] && (\mathcal{C}_j \text{ should fit in } U_j) \\
x_C &\geq 0 \quad \forall C \in \mathcal{C}
\end{aligned}$$

Let x^* be an extreme point solution to $\mathrm{FP}_W(\widetilde{I}, P)$ (if $\mathrm{FP}_W(\widetilde{I}, P)$ is feasible). By Rank Lemma, at most $p + r$ entries of x^* are non-zero. Since the number of variables and constraints is constant, x^* can be computed in constant time.

Let $\widetilde{H}$ be the set of tall items in $\widetilde{I}$. By Claim 8.2, we get that items in $\widetilde{H}$ have at most $1/\varepsilon\varepsilon_1$ distinct heights. Let there be q distinct heights of tall compartments in P. By Definition 8.5, we know that $q \leq 1/\varepsilon_{\text{cont}} = 6|\mathcal{T}|/\varepsilon\varepsilon_1$. We can similarly define *tall configurations* and we can similarly define a feasibility linear program for tall items, named $\mathrm{FP}_H(\widetilde{I}, P)$. $\widetilde{H}$ can be packed into tall compartments in P iff $\mathrm{FP}_H(\widetilde{I}, P)$ is feasible. Let y^* be an extreme point solution to $\mathrm{FP}_H(\widetilde{I}, P)$. Then y^* can be computed in constant time and y^* has at most $q + 1/\varepsilon\varepsilon_1$ positive entries.

Therefore, $\widetilde{I}$ can be packed into P iff the feasibility linear program $\mathrm{FP}(\widetilde{I}, P) := \mathrm{FP}_W(\widetilde{I}, P) \wedge \mathrm{FP}_H(\widetilde{I}, P)$ is feasible.

The solution (x^*, y^*) shows us how to split each compartment into *shelves*, where each shelf corresponds to a configuration C and the shelf can be split into C_j *containers* of width w_j and one container of width $w'_{C_0} - w(C)$. Let there be m bins in P. After splitting the configurations across compartments, we get at most $p + q + 2/\varepsilon\varepsilon_1 + m(n_W + n_H)$ shelves.

8.3.3 Converting a Fractional Packing to a Non-Fractional Packing

Let there be m bins in a packing P of empty compartments into bins. Suppose it is possible to pack $\widetilde{I}$ into P. Let x^* and y^* be extreme-point solutions to $\mathrm{FP}_W(\widetilde{I}, P)$ and $\mathrm{FP}_H(\widetilde{I}, P)$, respectively. This gives us a fractional compartmental packing of $\widetilde{I}$ into m bins. We will now show how to convert this to a non-fractional compartmental packing by removing some items of small total area. Formally, we give an algorithm called $\texttt{greedyPack}(\widetilde{I}, P, x^*, y^*)$. It returns a pair (Q, D), where Q is a (non-fractional) compartmental bin packing of items $\widetilde{I} - D$, where the compartments in the bin are as per P. D is called the set of discarded items, and we will prove that $a(D)$ is small.

For a configuration C in a wide compartment, there is a container of width $w'_{C_0} - w(C)$ available for packing small items. Hence, there are $p + q + 2/\varepsilon\varepsilon_1 + m(n_W + n_H)$ containers available inside compartments for packing small items. By Lemma 8.9, we can partition the space outside compartments into at most $m(3(n_W + n_H) + 1)$ containers. Therefore, the total number of containers available for packing small items is at most

$$m_S := (p + q + 2/\varepsilon\varepsilon_1) + m(4(n_W + n_H) + 1) \leq \left(\frac{|\mathcal{T}|^2}{2} + \frac{6|\mathcal{T}|}{\varepsilon\varepsilon_1} + \frac{2}{\varepsilon\varepsilon_1}\right) + \frac{16|\mathcal{T}|}{\varepsilon_1}m.$$

Greedily assign small items to small containers, i.e., keep assigning small items to a container till the area of items assigned to it is at least the area of the container, and then resume from the next container. Each small item will get assigned to some container. For each container C, pack the largest possible prefix of the assigned items using the Next-Fit Decreasing Height (NFDH) algorithm. By Lemma 3.4, the area of unpacked items would be less than $\varepsilon_2 + \delta + \varepsilon_2\delta$. Summing over all containers, we get that the unpacked area is less than $(\varepsilon_2 + \delta + \varepsilon_2\delta)m_S \leq 3\varepsilon_2 m_S$.

For each j, greedily assign wide items from $\widetilde{W}_j$ to containers of width w_j, i.e., keep assigning items till the height of items exceeds the height of the container. Each wide item will get assigned to some container. Then discard the last item from each container. For each shelf in a wide compartment having configuration C, the total area of items we discard is at most $\delta w(C)$. Similarly, we can discard tall items of area at most $\delta h(C)$ from each shelf in a tall compartment having configuration C.

Hence, across all configurations, we discard wide and tall items of area at most

$$\delta((p + q + 2/\varepsilon\varepsilon_1) + m(n_W + n_H)) \leq \delta\left(\frac{|\mathcal{T}|^2}{2} + \frac{6|\mathcal{T}|}{\varepsilon\varepsilon_1} + \frac{2}{\varepsilon\varepsilon_1}\right) + \frac{4\delta|\mathcal{T}|}{\varepsilon_1}m.$$

Therefore, for $(Q, D) := \texttt{greedyPack}(\widetilde{I}, P, x^*, y^*)$, we get

$$a(D) < \frac{52|\mathcal{T}|\varepsilon_2}{\varepsilon_1} m + 4\varepsilon_2 \left(\frac{|\mathcal{T}|^2}{2} + \frac{6|\mathcal{T}|}{\varepsilon\varepsilon_1} + \frac{2}{\varepsilon\varepsilon_1} \right) \tag{8.4}$$

where m is the number of bins used by P.

8.3.4 The Algorithm

We now summarize the algorithm for bin packing δ-skewed items I (see Algorithm 19 for a more precise description). First, use `round` on I, i.e., let $(\widetilde{I}, I_{\text{med}}) := \texttt{round}(I)$. Then enumerate all packings P of compartments into bins as per Section 8.3.1. For each packing P, check if $\widetilde{I}$ can be fractionally packed into P by solving the feasibility linear program (see Section 8.3.2). If yes, then use a solution to the feasibility linear program to compute a (non-fractional) compartmental packing of $\widetilde{I} - D$ using `greedyPack` (see Section 8.3.3), where D is the set of items discarded by `greedyPack`. Then pack $I_{\text{med}} \cup D$ into bins using the Next-Fit Decreasing Height (NFDH) algorithm. Output the best bin packing of I across all choices of P.

Algorithm 19 $\texttt{skewedCPack}_\varepsilon(I)$: Packs a set I of δ-skewed rectangular items into bins without rotating the items.

$(\widetilde{I}, I_{\text{med}}) = \texttt{round}_\varepsilon(I)$.
Initialize Q_{best} to `null`.
for $P \in \texttt{iterPackings}(\widetilde{I})$ **do** // `iterPackings` *is defined in Section 8.3.1.*
  $x^* = \text{opt}(\text{FP}_W(\widetilde{I}, P))$. // FP_W *and* FP_H *are defined in Section 8.3.2.*
  // *If* $\text{FP}_W(\widetilde{I}, P)$ *is feasible,* x^* *is an extreme-point solution to* $\text{FP}_W(\widetilde{I}, P)$.
  // *If* $\text{FP}_W(\widetilde{I}, P)$ *is infeasible,* x^* *is* `null`.
  $y^* = \text{opt}(\text{FP}_H(\widetilde{I}, P))$.
  if $x^* \neq$ `null` and $y^* \neq$ `null` **then** // *if* $\widetilde{I}$ *can be packed into* P
    $(Q, D) = \texttt{greedyPack}(\widetilde{I}, P, x^*, y^*)$. // `greedyPack` *is defined in Section 8.3.3.*
    $Q_D = \text{NFDH}(D \cup I_{\text{med}})$.
    if $Q \cup Q_D$ uses less bins than Q_{best} **then**
      $Q_{\text{best}} = Q \cup Q_D$.
    end if
  end if
end for
return Q_{best}

Recall the function f from Eq. (8.1) in Section 8.1.1. Since $\varepsilon_2 := f(\varepsilon_1)$, we get

$$\varepsilon_2 = f(\varepsilon_1) = \frac{\varepsilon\varepsilon_1}{104(1+1/\varepsilon\varepsilon_1)^{2/\varepsilon_1-2}} \leq \frac{\varepsilon\varepsilon_1}{104|\mathcal{T}|}. \tag{8.5}$$

The last inequality follows from the fact that $|\mathcal{T}| \leq (1+1/\varepsilon\varepsilon_1)^{2/\varepsilon_1-2}$.

Lemma 7.19. *Let I be a set of rectangular items where each item has height at most δ. Then the number of bins required by NFDH to pack I is less than $(2a(I)+1)/(1-\delta)$.*

Proof. (See Section 7.3.7.) □

Lemma 8.12. *Let I be a set of rectangular items where each item has width at most δ. Then the number of bins required by NFDH to pack I is less than $2a(I)/(1-\delta)+3$.*

Proof. The bin packing version of NFDH first packs I into shelves and then packs the shelves into bins using Next-Fit. Let the number of shelves be p. Let h_j be the height of the j^{th} shelf. Let S_j be the items in the j^{th} shelf. For $j \in [p-1]$, in the j^{th} shelf, the total width of items is more than $(1-\delta)$ and each item has height more than h_{j+1}. Therefore, $a(S_j) > h_{j+1}(1-\delta)$.

Let H be the sum of heights of all the shelves. Then

$$\begin{aligned} a(I) &> \sum_{i=1}^{p-1} a(S_j) > \sum_{i=1}^{p-1} h_{j+1}(1-\delta) > (1-\delta)(H-h_1) \\ \implies H &< \frac{a(I)}{1-\delta} + 1. \end{aligned}$$

By Lemma 3.1, the number of bins is less than $2H+1 < 2a(I)/(1-\delta)+3$. □

Theorem 8.13. *The number of bins used by* $\texttt{skewedCPack}_\varepsilon(\widetilde{I})$ *is less than*

$$(1+20\varepsilon)\,\mathrm{opt}(I) + \frac{1}{13}\left(1+\frac{1}{\varepsilon\varepsilon_1}\right)^{2/\varepsilon_1-2} + 23.$$

Proof. In an optimal fractional compartmental bin packing of $\widetilde{I}$, let P^* be the corresponding packing of empty compartments into bins. Hence, P^* contains $m := \mathrm{fcopt}(\widetilde{I})$ bins. Since $\texttt{iterPackings}(\widetilde{I})$ iterates over all packings of compartments into bins, $P^* \in \texttt{iterPackings}(\widetilde{I})$. Since wide and tall items in $\widetilde{I}$ can be packed into the compartments of P^*, we get that x^* and y^* are not `null`. By Lemmas 8.12 and 7.19, the number of bins used by NFDH to pack $I_{\text{med}} \cup D$ is less than $2a(I_{\text{med}} \cup D)/(1-\delta)+3+1/(1-\delta)$. Therefore, the number of bins used

by `skewedCPack`(I) is less than

$$
\begin{aligned}
&m + \frac{2a(I_{\mathrm{med}} \cup D)}{1-\delta} + 3 + \frac{1}{1-\delta} \\
&< m + \frac{2\varepsilon}{1-\delta} a(I) + \frac{2\varepsilon_2}{1-\delta} \left(\frac{52|\mathcal{T}|}{\varepsilon_1} m + 4 \left(\frac{|\mathcal{T}|^2}{2} + \frac{6|\mathcal{T}|+2}{\varepsilon\varepsilon_1} \right) \right) + 3 + \frac{1}{1-\delta} \\
&\qquad\qquad \text{(by Lemma 8.1 and Eq. (8.4))} \\
&= \left(1 + \frac{104\varepsilon_2|\mathcal{T}|}{\varepsilon_1(1-\delta)} \right) m + \frac{2\varepsilon}{1-\delta} a(I) + 3 + \frac{1}{1-\delta} + \frac{8\varepsilon_2}{1-\delta} \left(\frac{|\mathcal{T}|^2}{2} + \frac{6|\mathcal{T}|+2}{\varepsilon\varepsilon_1} \right) \\
&= \left(1 + \frac{\varepsilon}{1-\delta} \right) m + \frac{2\varepsilon}{1-\delta} a(I) + 3 + \frac{1}{13(1-\delta)} \left(\frac{\varepsilon\varepsilon_1|\mathcal{T}|}{2} + 19 + \frac{2}{|\mathcal{T}|} \right). \qquad \text{(by Eq. (8.5))}
\end{aligned}
$$

By Theorem 8.11 and Lemma 8.3, we get

$$
m = \mathrm{fcopt}(\widetilde{I}) < (1+4\varepsilon)\,\mathrm{fopt}(\widetilde{I}) + 2 < (1+4\varepsilon)(1+\varepsilon)\,\mathrm{opt}(I) + 4 + 8\varepsilon.
$$

Therefore, the number of bins used by `skewedCPack`(I) is less than

$$
\begin{aligned}
&\left((1+4\varepsilon)(1+\varepsilon) \left(1 + \frac{\varepsilon}{1-\delta} \right) + \frac{2\varepsilon}{1-\delta} \right) \mathrm{opt}(I) \\
&\qquad + (4+8\varepsilon) \left(1 + \frac{\varepsilon}{1-\delta} \right) + 3 + \frac{1}{13(1-\delta)} \left(\frac{\varepsilon\varepsilon_1|\mathcal{T}|}{2} + 19 + \frac{2}{|\mathcal{T}|} \right) \\
&\leq (1+20\varepsilon)\,\mathrm{opt}(I) + \frac{1}{13} \left(1 + \frac{1}{\varepsilon\varepsilon_1} \right)^{2/\varepsilon_1 - 2} + 23. \qquad \text{(since } \delta \leq \varepsilon_1 \leq \varepsilon \leq 1/2\text{)}
\end{aligned}
$$

□

8.4 Handling Item Rotations

In this section, we will briefly explain changes to `skewedCPack` and its analysis so that they work for the case where items can be rotated by 90°.

For the rotational version of the problem, we can assume without loss of generality that the height of each δ-skewed rectangle is at most δ, because otherwise we can rotate the item. We assume without loss of generality that the width of the bin is 1 and the height of the bin is at least 1, because we can rotate the bin and scale its width and height equally. We also assume that the height of the bin is a constant.

To handle the rotational case, we do not require any change in Section 8.1.

The structural theorem in Section 8.2 doesn't require any conceptual modifications. The

only change is that the number of tall compartments in a tall cell can be more than $1/\varepsilon_1 - 1$. Specifically, if the height of the bin is H, the number of tall compartments in a tall cell is now upper-bounded by H/ε_1. (Hence, for the number of compartments to be a constant, we require the bin's height to be a constant). This will increase the running time of `iterPackings`, but there will be no change to Theorem 8.11.

The feasibility linear program of Section 8.3.2 will have to change to take item rotations into account. Instead of using two programs—FP_W and FP_H—which fractionally pack wide and tall items separately, we will use just one program which will also decide which items to rotate. To do this, we allow an item to belong to both wide and tall configurations. The number of constraints in the feasibility linear program will be a constant that depends on ε and ε_1.

The `greedyPack` algorithm of Section 8.3.3 will remain the same, but the total area of discarded items will be slightly different because the number of compartments in a bin can now be larger. Finally, the AAR of `skewedCPack` for the rotational version will be $1 + \Theta(1)\varepsilon$ by the same kind of analysis as in Section 8.3.4.

Chapter 9

Conclusion and Future Directions

In this thesis, we studied approximation algorithms for different kinds of geometric packing problems.

In Chapters 4 and 5, we give approximation algorithms for the generalized multidimensional bin packing problem, a problem that has wide practical applications but had not yet received attention from the theoretical computer science community. We hope that our work will drive interest into this problem. Possible future directions of research for this problem include obtaining algorithms with better asymptotic approximation ratios, and identifying important special cases that can be solved efficiently.

In Section 4.3, we extended the Round-and-Approx framework [10, 14] to a larger class of bin packing algorithms. We expect that our progress will help in better understanding the power of Round-and-Approx and enable the design of better approximation algorithms for other packing problems. It would also be interesting to see applications of Round-and-Approx to other set-cover type problems, like round-SAP and round-UFP [30].

In Chapter 6, we give approximation algorithms for the rotational version of geometric bin packing. Our algorithm for 3D GBP obtains the best-known asymptotic approximation ratio. Most previous works make assumptions about bin sizes which are unlikely to hold in practice. Our algorithms, however, don't rely on such assumptions. An important problem is to obtain better asymptotic approximation ratios for dD GBP, especially 3D GBP. There is currently an exponential gap between the lower and upper bounds on the AAR achievable for dD GBP (constant lower bound, upper bound of 1.691^{d-1}). It would be interesting to reduce this huge gap.

We have seen that for 2D GBP, the algorithm of Bansal, Lodi and Sviridenko [15] has an AAR equal to the asymptotic price of guillotinability (APoG). It is known that $4/3 \leq \text{APoG} \leq T_\infty \approx 1.69103$. If we could obtain a better upper bound on APoG, that could give us the

best-known algorithm for 2D GBP. Hence, obtaining tight bounds on APoG is an important problem. In Chapter 7, we show that for the special case of δ-skewed rectangles, when δ is close to 0, both the lower and upper bounds on APoG are close to 4/3. We hope that our work will shed more light on the general case of the problem. A good next step would be to obtain tight bounds on APoG when all rectangles are non-δ-skewed, and after that focus on the general case of the problem.

In Chapter 8, we gave an approximation algorithm for 2D GBP when the rectangles are δ-skewed. Our work indicates that to improve the AAR for 2D GBP, a good next step would be to focus on the special case of non-δ-skewed rectangles.

Finally, it would be interesting to see if our work can be extended to other problems similar to bin packing, like bin covering [4] and scheduling. Another direction is to explore different problem paradigms, like online algorithms, or different analysis paradigms, like average-case analysis.

Bibliography

[1] Fidaa Abed, Parinya Chalermsook, José Correa, Andreas Karrenbauer, Pablo Pérez-Lantero, José A Soto, and Andreas Wiese. On guillotine cutting sequences. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 1–19, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.1. 6

[2] M. T. Alonso, R. Alvarez-Valdes, Manuel Iori, F. Parreño, and J. M. Tamarit. Mathematical models for multicontainer loading problems. *Omega*, 66:106–117, 2017. doi:10.1016/j.omega.2016.02.002. 8

[3] Samir V. Amiouny, John J. Bartholdi III, John H. Vande Vate, and Jixian Zhang. Balanced loading. *Operations Research*, 40(2):238–246, 1992. doi:10.1287/opre.40.2.238. 8

[4] Susan F Assmann, David S. Johnson, Daniel J. Kleitman, and JY-T Leung. On a dual version of the one-dimensional bin packing problem. *Journal of algorithms*, 5(4):502–525, 1984. doi:10.1016/0196-6774(84)90004-X. 188

[5] Brenda S Baker and Edward G Coffman, Jr. A tight asymptotic bound for next-fit-decreasing bin-packing. *SIAM Journal on Algebraic Discrete Methods*, 2(2):147–152, 1981. doi:10.1137/0602019. 17

[6] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A New and Improved Algorithm for Online Bin Packing. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2018.5. 15, 16

[7] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new lower bound for classic online bin packing. *Algorithmica*, pages 1–16, 03 2021. doi:10.1007/s00453-021-00818-7. 15

[8] János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. *Theoretical Computer Science*, 440:1–13, 2012. doi:10.1016/j.tcs.2012.04.017. 15

[9] Nikhil Bansal, Alberto Caprara, Klaus Jansen, Lars Prädel, and Maxim Sviridenko. A structural lemma in 2-dimensional packing, and its implications on approximability. In *International Symposium on Algorithms and Computation*, pages 77–86. Springer, 2009. doi:10.1007/978-3-642-10631-6_10. 18, 28

[10] Nikhil Bansal, Alberto Caprara, and Maxim Sviridenko. A new approximation method for set covering problems, with applications to multidimensional bin packing. *SIAM Journal on Computing*, 39(4):1256–1278, 2010. doi:10.1137/080736831. 7, 9, 11, 17, 20, 27, 32, 33, 34, 37, 144, 187

[11] Nikhil Bansal, José R Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: inapproximability results and approximation schemes. *Mathematics of operations research*, 31(1):31–49, 2006. doi:10.1287/moor.1050.0168. 5, 17, 18

[12] Nikhil Bansal, Marek Eliáš, and Arindam Khan. Improved approximation for vector bin packing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1561–1579. SIAM, 2016. doi:10.1137/1.9781611974331.ch106. 7, 8, 20, 32

[13] Nikhil Bansal, Xin Han, Kazuo Iwama, Maxim Sviridenko, and Guochuan Zhang. Harmonic algorithm for 3-dimensional strip packing problem. In *SODA*, volume 7, pages 1197–1206. Citeseer, 2007. 19

[14] Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *SODA*, pages 13–25, 2014. doi:10.1137/1.9781611973402.2. ii, 5, 9, 10, 11, 17, 27, 34, 35, 39, 187

[15] Nikhil Bansal, Andrea Lodi, and Maxim Sviridenko. A tale of two dimensional bin packing. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 657–666. IEEE, 2005. doi:10.1109/SFCS.2005.10. 6, 12, 38, 187

[16] Suman Kalyan Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 4955–4966, 2019. 8

[17] Andreas Bortfeldt and Gerhard Wäscher. Constraints in container loading–a state-of-the-art review. *European Journal of Operational Research*, 229(1):1–20, 2013. doi:10.1016/j.ejor.2012.12.006. 8

[18] Alberto Caprara. Packing d-dimensional bins in d stages. *Mathematics of Operations Research - MOR*, 33:203–215, 02 2008. doi:10.1287/moor.1070.0289. ii, 5, 10, 12, 13, 17, 19, 33, 49, 113, 114, 115, 119, 126, 136, 139, 140, 146

[19] Timothy M Chan. Approximation schemes for 0-1 knapsack. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/OASIcs.SOSA.2018.5. 16

[20] Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM journal on computing*, 33(4):837–851, 2004. doi:10.1137/S0097539799356265. 20

[21] Miroslav Chlebík and Janka Chlebíková. Hardness of approximation for orthogonal rectangle packing and covering problems. *Journal of Discrete Algorithms*, 7(3):291–305, 2009. doi:10.1016/j.jda.2009.02.002. 5, 13, 18, 168

[22] Fan RK Chung, Michael R Garey, and David S Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic Discrete Methods*, 3(1):66–76, 1982. doi:10.1137/0603007. 17

[23] Edward G. Coffman, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: Survey and classification. In *Handbook of combinatorial optimization*. Springer, 2013. doi:10.1007/978-1-4419-7997-1_35. 1, 15, 16

[24] Edward G. Coffman, Michael R. Garey, David S. Johnson, and Robert E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9:808–826, 1980. doi:10.1137/0209062. 17, 19, 25, 135

[25] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Maximum flow. In *Introduction to Algorithms*. MIT press, 2009. 80, 81

[26] W Fernandez De La Vega and George S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981. doi:10.1007/BF02579456. 2, 16, 20, 116, 119, 124, 135, 160, 170

[27] Florian Diedrich, Rolf Harren, Klaus Jansen, Ralf Thöle, and Henning Thomas. Approximation algorithms for 3d orthogonal knapsack. *Journal of Computer Science and Technology*, 23(5):749, 2008. doi:10.1007/s11390-008-9170-7. 18, 32, 33, 42

[28] György Dósa. The tight bound of first fit decreasing bin-packing algorithm is $\mathrm{FFD}(I) \leq 11/9\,\mathrm{OPT}(I) + 6/9$. In *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11. Springer, 2007. doi:10.1007/978-3-540-74450-4_1. 2, 16

[29] György Dósa and Jirí Sgall. First fit bin packing: A tight analysis. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.538. 2, 16

[30] Khaled M. Elbassioni, Naveen Garg, Divya Gupta, Amit Kumar, Vishal Narula, and Arindam Pal. Approximation algorithms for the unsplittable flow problem on paths and trees. In *FSTTCS*, volume 18 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 267–275, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.267. 10, 187

[31] Leah Epstein and Rob Van Stee. This side up! *ACM Transactions on Algorithms (TALG)*, 2(2):228–243, 2006. doi:10.1145/1150334.1150339. 10, 12, 18, 19

[32] Sándor P. Fekete and Jörg Schepers. A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, 60(2):311–329, 2004. doi:10.1007/s001860400376. 113

[33] Alan M Frieze, MRB Clarke, et al. Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses. *European Journal of Operational Research*, 15(1):100–109, 1984. 21, 33

[34] Waldo Gálvez, Fabrizio Grandoni, Afrouz Jabal Ameli, Klaus Jansen, Arindam Khan, and Malin Rau. A tight $(3/2+\varepsilon)$ approximation for skewed strip packing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.44. 13

[35] Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via l-packings. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 260–271. IEEE, 2017. doi:10.1109/FOCS.2017.32. 38

[36] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. doi:10.1007/BF02579273. 28

[37] Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *SODA*, pages 2616–2625, 2017. doi:10.1137/1.9781611974782.172. 16

[38] Klaus Jansen. A $(3/2 + \epsilon)$ approximation algorithm for scheduling moldable and non-moldable parallel tasks. In *SPAA*, pages 224–235, 2012. doi:10.1145/2312005.2312048. 11

[39] Klaus Jansen and Lars Prädel. New approximability results for two-dimensional bin packing. In *SODA*, pages 919–936, 2013. doi:10.1007/s00453-014-9943-z. 17

[40] Klaus Jansen and Lars Prädel. A new asymptotic approximation algorithm for 3-dimensional strip packing. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 327–338. Springer, 2014. doi:10.1007/978-3-319-04298-5_29. 19

[41] Klaus Jansen and Lars Prädel. New approximability results for two-dimensional bin packing. *Algorithmica*, 74(1):208–269, 2016. doi:10.1007/s00453-014-9943-z. 11, 17, 38, 48, 50, 52, 53

[42] Klaus Jansen and Roberto Solis-Oba. An asymptotic approximation algorithm for 3d-strip packing. In *ACM-SIAM Symposium on Discrete Algorithm (SODA)*, pages 143–152, 2006. 19

[43] Klaus Jansen and Rob van Stee. On strip packing with rotations. In *ACM Symposium on Theory of Computing (STOC)*, pages 755–761, 2005. doi:10.1145/1060590.1060702. 19

[44] Klaus Jansen and Guochuan Zhang. On rectangle packing: maximizing benefits. In *SODA*, volume 4, pages 204–213, 2004. 18

[45] Ce Jin. An improved fptas for 0-1 knapsack. In *International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.76. 16

[46] David S Johnson. *Near-Optimal Bin Packing Algorithms*. PhD thesis, Massachusetts Institute of Technology, USA, 1973. 15, 16

[47] Matthew Joseph, Michael J. Kearns, Jamie H. Morgenstern, and Aaron Roth. Fairness in learning: Classic and contextual bandits. In *Conference on Neural Information Processing Systems (NIPS)*, pages 325–333, 2016. 8

[48] 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)*, pages 312–320. IEEE, 1982. doi:10.1109/SFCS.1982.61. 16, 28

[49] Claire Kenyon and Eric Rémila. Approximate strip packing. In *FOCS*, pages 31–36, 1996. doi:10.1109/SFCS.1996.548461. 17, 19, 160, 161, 170

[50] Arindam Khan. *Approximation algorithms for multidimensional bin packing*. PhD thesis, Georgia Institute of Technology, 2015. 35, 41, 48

[51] Arindam Khan, Arnab Maiti, Amatya Sharma, and Andreas Wiese. On guillotine separable packings for the two-dimensional geometric knapsack problem. *ArXiv*, 2102.05854, 2021. arXiv:2103.09735. 6

[52] Arindam Khan and Madhusudhan Reddy Pittu. On guillotine separability of squares and rectangles. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.47. 6

[53] Arindam Khan, Eklavya Sharma, and K. V. N. Sreenivas. Approximation algorithms for generalized multidimensional knapsack. *ArXiv*, 2102.05854, 2021. arXiv:2102.05854. 43, 51, 109

[54] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011. 27

[55] Eugene L Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979. doi:10.1287/moor.4.4.339. 3, 11, 16, 143

[56] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32(3):562–572, July 1985. doi:10.1145/3828.3833. 10, 15, 16, 114

[57] Keqin Li and Kam-Hoi Cheng. On three-dimensional packing. *SIAM Journal on Computing*, 19(5):847–867, 1990. doi:10.1137/0219059. 19

[58] Colin McDiarmid. On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188, 1989. 41, 45

[59] Michael L McHale and Roshan P Shah. Cutting the guillotine down to size. *PC AI*, 13:24–26, 1999. URL: https://www.amzi2.com/articles/papercutter.htm. 6

[60] Flavio Keidi Miyazawa and Yoshiko Wakabayashi. Three-dimensional packings with rotations. *Computers & Operations Research*, 36(10):2801–2815, 2009. doi:10.1016/j.cor.2008.12.015. 10, 18, 19

[61] Maria Flavia Monaco, Marcello Sammarra, and Gregorio Sorrentino. The terminal-oriented ship stowage planning problem. *EJOR*, 239(1):256–265, 2014. doi:10.1016/j.ejor.2014.05.030. 8

[62] Célia Paquay, Michael Schyns, and Sabine Limbourg. A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International Transactions in Operational Research*, 23(1-2):187–213, 2016. doi:10.1111/itor.12111. 7

[63] Deval Patel, Arindam Khan, and Anand Louis. Group fairness for knapsack problems. *ArXiv*, 2006.07832, 2020. arXiv:2006.07832. 8

[64] Boaz Patt-Shamir and Dror Rawitz. Vector bin packing with multiple-choice. *Discrete Applied Mathematics*, 160(10-11):1591–1600, 2012. doi:10.1016/j.dam.2012.02.020. 11

[65] Serge A Plotkin, David B Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995. doi:10.1287/moor.20.2.257. 28

[66] Lars Dennis Prädel. *Approximation Algorithms for Geometric Packing Problems*. PhD thesis, Kiel University, 2012. URL: https://macau.uni-kiel.de/servlets/MCRFileNodeServlet/dissertation_derivate_00004634/dissertation-praedel.pdf?AC=N. 50, 52, 53, 79, 85, 96

[67] Jakob Puchinger, Günther R Raidl, and Gabriele Koller. Solving a real-world glass cutting problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 165–176. Springer, 2004. doi:10.1007/978-3-540-24652-7_17. 6

[68] Thomas Rothvoß. Approximating bin packing within $O(\log OPT * \log\log OPT)$ bins. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 20–29, 2013. doi:10.1109/FOCS.2013.11. 16

[69] Sai Sandeep. Almost optimal inapproximability of multidimensional packing problems. *ArXiv*, 2101.02854, 2021. arXiv:2101.02854. 20

[70] W Schneider. Trim-loss minimization in a crepe-rubber mill; optimal solution versus heuristic in the 2 (3) - dimensional case. *European Journal of Operational Research*, 34(3):273–281, 1988. doi:10.1016/0377-2217(88)90148-8. 6

[71] Eklavya Sharma. Analysis of the harmonic function used in bin-packing. *ArXiv*, 2011.11618, 2020. arXiv:2011.11618. 114

[72] Eklavya Sharma. An approximation algorithm for covering linear programs and its application to bin-packing. *ArXiv*, 2011.10963, 2020. arXiv:2011.11268. 34, 42, 51, 109

[73] Knut Olav Brathaug Sørset. A heuristic approach to the three-dimensional bin packing problem with weight constraints. Master's thesis, Høgskolen i Molde-Vitenskapelig høgskole i logistikk, 2019. 8

[74] A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997. doi:10.1137/S0097539793255801. 30, 31

[75] Gregory S. Taylor, Yupo Chan, and Ghulam Rasool. A three-dimensional bin-packing model: exact multicriteria solution and computational complexity. *Annals of Operations Research*, 251(1-2):397–427, 2017. doi:10.1007/s10479-015-2048-5. 8

[76] Alan Tsang, Bryan Wilder, Eric Rice, Milind Tambe, and Yair Zick. Group-fairness in influence maximization. In *IJCAI*, pages 5997–6005, 2019. 8

[77] David P Williamson and David B Shmoys. Deterministic rounding of linear programs. In *The design of approximation algorithms*. Cambridge university press, 2011. 27

[78] David P Williamson and David B Shmoys. Random sampling and randomized rounding of linear programs. In *The design of approximation algorithms*. Cambridge university press, 2011. 27

[79] David P Williamson and David B Shmoys. Rounding data and dynamic programming. In *The Design of Approximation Algorithms*. Cambridge university press, 2011. 3, 16

[80] Gerhard J Woeginger. There is no asymptotic ptas for two-dimensional vector packing. *Information Processing Letters*, 64(6):293–297, 1997. doi:10.1016/S0020-0190(97)00179-8. 20

[81] Guang Yang. *gbp: a bin packing problem solver*, 2017. R package version 0.1.0.4. URL: https://CRAN.R-project.org/package=gbp. 8

[82] Hu Zhang and Klaus Jansen. Scheduling malleable tasks. In *Handbook of Approximation Algorithms and Metaheuristics*. Chapman & Hall/CRC, 2007. 11