Improved Algorithms for Variants of Bin Packing and Knapsack

A THESIS
SUBMITTED FOR THE DEGREE OF
Master of Technology (Research)
IN THE
Faculty of Engineering

BY

Venkata Naga Sreenivasulu Karnati

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

July, 2022

Declaration of Originality

I, **Venkata Naga Sreenivasulu Karnati**, with SR No. **04-04-00-10-22-20-1-18699** hereby declare that the material presented in the thesis titled

Improved Algorithms for Variants of Bin Packing and Knapsack

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2020-2022**.
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-July-2022

Sreenivasulu
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
Bengaluru, 25/07/22

Advisor Name: Dr. Arindam Khan

Advisor Signature

DEDICATED TO

My Parents, Sister, and Brother

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Arindam Khan for believing in me and offering me a Project Associate position in his lab before I joined the M. Tech. (Research) program under his guidance. Without his constant support, this work wouldn't have been possible. Apart from being a tremendous researcher, he has been an excellent advisor who keeps striving for my academic excellence (even more than I do). The patience he has shown in certain situations – e.g., finding a bug just a few days before submission, sloppy writing – is truly commendable. I am also thankful for how he respects my time, allowing my schedule to be flexible. In my opinion, he is the best kind of advisor one can possibly find.

I am grateful to IISc for being such an outstanding research center, and for its beautiful and lively environment. I only regret that I haven't tried for an internship here during my undergrad studies. I am also thankful to the CSA department for the courses and the research environment it offered. The Theoretical CS faculty here is one of the best in the country, and I am glad to have come here. I am thankful to Prof. Arvind Ayyer (Math Department), Prof. Siddharth Barman, Prof. Chiranjib Bhattacharya, Dr. Ankit Garg (Microsoft Research India), Prof. Anand Louis, and Prof. Rahul Saladi for the courses they offered, through which I learned a lot. I would especially like to thank Prof. Siddharth and Prof. Rahul for guiding me about my future plans. A special mention goes to the CSA staff, especially Padmavati Madam and Kushael Madam, for handling my numerous administrative queries promptly.

I would like to thank Eklavya Sharma and Swati Allabadi for being such good friends and lab-mates when I was a Project Associate. Eklavya is one of the most hard-working students I have ever met, and I feel privileged to have collaborated with him. His discipline, LaTeX skills, and ability to produce ideas used to amaze and inspire me. I am also thankful to Aditya Vardhan Varre, my undergraduate friend, due to whom I first contacted Prof. Arindam.

I thank Aditya Lonkar, Aditya Subramanian, Arka Ray, Sudarshan, Rishikesh Gajjala, and Rameesh Paul for the technical discussions I had with them. I thank Prayas Rautray for carefully reviewing my thesis. I am also grateful to Rameesh and Vishakha Patil for patiently helping me with the thesis submission process, travel to ICALP, and continuation to Ph. D.

Acknowledgements

I am grateful to all my badminton and cricket friends, and mess buddies because of whom I had a lot of fun. A special mention goes to Likhith Marrapu for being a constant badminton partner and a good friend.

Finally, I would like to thank my parents for everything they have done for me. My wish to make them happy keeps me motivated to achieve academic success. I thank my brother for his company during the online semesters at home. I thank my sister for persuading me to apply to GATE, sans which I would not have ended up studying at IISc.

Abstract

We study variants of two classical optimization problems: Bin Packing and Knapsack. Both bin packing and knapsack fall under the regime of "Packing and Covering Problems". In bin packing, we are given a set of input items, each with an associated size, and the objective is to pack these into the minimum number of unit capacity bins. On the other hand, in knapsack, each item has an additional profit associated with it, and the objective is to find a maximum profitable subset that can be packed into a unit capacity knapsack. Both bin packing and knapsack find numerous applications; however, both turn out to be NP-Hard. Hence, it is natural to seek approximation algorithms for these problems. Ibarra and Kim, and Lawler settled the knapsack problem by giving an FPTAS, whereas the progressive works of de la Vega and Lueker, Karmarkar and Karp, and Rothvoss have given improved approximation schemes for the bin packing problem. However, many variants of these problems (e.g. multidimensional, geometric, stochastic) also find wide applicability, but haven't been settled. We make progress on this front by providing new and improved algorithms for several such variants.

First, we study online bin packing under the i.i.d. model, where item sizes are sampled independently and identically from a distribution in (0,1]. Both the distribution and the total number of items are unknown. The items arrive one by one and their sizes are revealed upon their arrival and they must be packed immediately and irrevocably in bins of size 1. We provide a simple meta-algorithm that takes an offline α-asymptotic approximation algorithm and provides a polynomial-time $(\alpha+\varepsilon)$-competitive algorithm for online bin packing under the i.i.d. model, where $\varepsilon > 0$ is a small constant. Using the AFPTAS for offline bin packing, we thus provide a linear time $(1+\varepsilon)$-competitive algorithm for online bin packing under the i.i.d. model, thus settling the problem.

Then we study a well-known geometric generalization of the knapsack problem, the 3-D Knapsack problem. In this problem, the items are cuboids in three dimensions and the knapsack is a unit cube. The objective is to pack a maximum profitable subset of the input set in a non-overlapping, axis-parallel manner inside the knapsack. Depending on whether rotations around axes (by ninety degrees) are allowed or not, we obtain two further variants. [DHJ+07] gave

Abstract

a $(7+\varepsilon)$ (resp. $(5+\varepsilon)$) approximation algorithm for the 3-D Knapsack problem without rotations (resp. with rotations). Despite the importance of the problem, there has been no improvement in the ratios for fifteen years. First, we give alternate algorithms that achieve the same approximation ratios $(7+\varepsilon, 5+\varepsilon)$. These algorithms and their analyses are far simpler. Then, in the case when rotations are allowed, we give an improved $\left(\frac{31}{7}+\varepsilon\right)$ approximation algorithm in the general setting, and a $(3+\varepsilon)$ approximation algorithm in the special setting where each item has a profit equal to its volume.

We also introduce and study a generalization of the knapsack problem with geometric and vector constraints. The input is a set of rectangular items, each with an associated profit and d nonnegative weights (d-D vector), and a square knapsack. The goal is to find a non-overlapping, axis-parallel packing of a subset of items into the given knapsack such that the vector constraints are not violated, i.e., the sum of weights of all the packed items in any of the d dimensions does not exceed one. Two variants are defined: rotations allowed by $90°$, rotations not allowed. We give $(2+\varepsilon)$ approximation algorithms for both the variants.

Finally, we consider the problem of packing d-D hypercubes into a knapsack defined by the region $[0,1]^d$. Each hypercube has an associated profit, and the goal is to find a maximum profitable non-overlapping, axis-parallel packing. We consider two special cases of this problem: *(i)* cardinality case, where each item has unit profit, *(ii)* bounded profit-volume ratio case, where the profit-to-volume ratio of each item lies in the range $[1, r]$ for some fixed constant r. We give near-optimal algorithms for both the cases.

Publications based on this Thesis

1. **Near-Optimal Algorithms for Stochastic Online Bin Packing**
   Joint work with Nikhil Ayyadevara, Rajni Dabas, and Arindam Khan.
   The 49th EATCS International Colloquium on Automata, Languages and Programming (ICALP 2022)

2. **A PTAS for Packing Hypercubes into a Knapsack**
   Joint work with Klaus Jansen, Arindam Khan, and Marvin Lira.
   The 49th EATCS International Colloquium on Automata, Languages and Programming (ICALP 2022)

3. **Approximation Algorithms for Generalized Multidimensional Knapsack**
   Joint work with Arindam Khan, and Eklavya Sharma.
   https://arxiv.org/abs/2102.05854

4. **Improved Approximation Algorithms for 3-D Knapsack**
   Joint work with Arindam Khan.
   Manuscript (to be submitted)

Other Related Publications

1. **Geometry Meets Vectors: Approximation Algorithms for Multidimensional Packing**
   Joint work with Arindam Khan, and Eklavya Sharma.
   https://arxiv.org/abs/2106.13951
   Under Submission

Contents

Acknowledgements **i**

Abstract **iii**

Publications based on this Thesis **v**

Contents **vi**

List of Figures **x**

1 Introduction **1**

1.1 Approximation and Online Algorithms 1

1.1.1 Approximation Algorithms 2

1.1.2 Online Algorithms 3

1.1.3 Stochastic Models 3

1.1.3.1 The i.i.d. Model 4

1.1.3.2 The Random-order Model 4

1.2 Bin Packing, Knapsack, and their Variants 5

1.2.1 Classical Bin Packing 5

1.2.2 Online Bin Packing 5

1.2.3 Classical Knapsack 6

1.2.4 Geometric Knapsack 7

1.2.5 Vector Knapsack 9

1.3 Contribution of this Thesis 10

1.3.1 Stochastic Online Bin Packing 10

1.3.1.1 Online Bin Packing under the i.i.d. Model 10

1.3.1.2 Online Bin Packing under the Random-order Model 11

1.3.2 3-D Geometric Knapsack 11

1.3.3 Generalized Multidimensional Knapsack . . . 12
1.3.4 d-D Hypercube Knapsack . . . 13

2 Notations and Preliminaries 14
2.1 General Notations . . . 14
2.2 Performance Measures of Algorithms . . . 14
2.2.1 Approximation Ratio . . . 15
2.2.2 Asymptotic Approximation Ratio . . . 15
2.2.3 Competitive Ratio . . . 16
2.2.4 Expected Competitive Ratio . . . 17
2.2.4.1 ECR in the i.i.d. Model . . . 17
2.2.4.2 ECR in the Random-order Model . . . 18
2.3 Special Approximation Schemes . . . 18
2.4 Greedy Packing Algorithms . . . 20
2.4.1 Next-Fit and Best-Fit . . . 20
2.4.1.1 Next-Fit . . . 20
2.4.1.2 Best-Fit . . . 21
2.4.2 Next-Fit Decreasing Height . . . 21
2.4.2.1 NFDH in Higher Dimensions . . . 22

3 Near-optimal Algorithms for Stochastic Online Bin Packing 23
3.1 Related Work . . . 23
3.2 Overview of Results and Techniques . . . 25
3.3 Online Bin Packing Problem under the i.i.d. Model . . . 29
3.3.1 Algorithm Assuming that the Value of n is Known . . . 29
3.3.1.1 Analysis of Case 1: $|L_0| \leq \delta^3 \cdot \mathcal{W}(T_0)$ . . . 35
3.3.1.2 Analysis of Case 2: $|L_0| > \delta^3 \cdot \mathcal{W}(T_0)$ . . . 37
3.3.2 Getting Rid of the Assumption on the Knowledge of the Input Size . . . 42
3.3.2.1 Analysis . . . 43
3.4 Best-Fit under the Random-Order Model . . . 50
3.4.1 When Item Sizes are Larger than 1/3 . . . 50
3.4.2 The 3-Partition Problem under Random-Order Model . . . 52
3.4.2.1 Deferred Proof of Claim 3.19 . . . 55
3.5 Online Vector Packing Problem under i.i.d. model . . . 57
3.5.1 Algorithm . . . 58

3.5.2 Analysis 58

4 3-D Knapsack Problem and its Variants **60**

4.1 Notations and Preliminaries 60

4.1.1 Steinberg's Algorithm 61

4.1.2 Packing Large Items 61

4.2 A simpler $(7+\varepsilon)$-approximation Algorithm (without Rotations) 62

4.2.1 Algorithm for the Cardinality case 62

4.2.1.1 Analysis 65

4.2.2 Extending to General Case 67

4.3 A simpler $(5+\varepsilon)$-approximation Algorithm (with Rotations) 68

4.3.1 Algorithm for the Cardinality Case 68

4.3.2 Extending to General Case 71

4.4 A $(31/7+\varepsilon)$-approximation Algorithm (with Rotations) 71

4.4.1 Algorithm for the Cardinality Case 72

4.4.1.1 Analysis of the Algorithm 73

4.4.2 Extending to the General Case 75

4.5 A $(3+\varepsilon)$-approximation Algorithm for Maximizing the Packed Volume (with Rotations) 76

4.5.1 The Algorithm 76

5 Generalized Multidimensional Knapsack **80**

5.1 Preliminaries and Notations 80

5.1.1 Prior Work 80

5.1.2 Notations and Definitions 81

5.1.3 Organization of the Chapter 81

5.2 Vector-Max-GAP problem and a PTAS 82

5.2.1 Dynamic-Programming Algorithm for Integral Input 83

5.2.2 Optimal Solution with Resource Augmentation 83

5.2.3 Packing Small Items 85

5.2.3.1 Trimming 85

5.2.3.2 Near-optimal Packing of Small Items 85

5.2.4 A Structural Result 86

5.2.5 PTAS for Vector-Max-GAP 86

5.3 Algorithm for $(2, d)$ Knapsack Problem 88

5.3.1 A Structural Result 88
5.3.2 Proof of the Structural Result 89
5.3.2.1 Corridors 90
5.3.2.2 Corridor Decomposition 91
5.3.2.3 Reorganizing Boxes into Containers 92
5.3.2.4 Packing Small Items 94
5.3.2.5 Shifting Argumentation 96
5.3.3 Container Packing Problem and Vector-Max-GAP 102
5.3.4 Reduction of Container Packing Problem to Vector-Max-GAP 103
5.3.5 Algorithm 105

6 The d-D Hypercube Knapsack Problem 107
6.1 Prior Work 107
6.2 PTAS for the Bounded-density Case 108
6.2.1 NFDH for Small Items 109
6.2.2 Algorithm 109
6.2.3 Analysis 111
6.3 PTAS for the Cardinality Case 113

7 Conclusion 115
7.1 Open Problems 116

Bibliography 118

List of Figures

1.1 An example instance of bin packing. The sizes of the items are labeled below. An optimal packing requires 4 bins. 6

1.2 The sizes are labeled below the items and the profits are labeled on the items. An optimal packing into a knapsack is shown. 7

1.3 Three invalid packings and one valid packing. *(i)* An item protrudes from the knapsack, so it's invalid. *(ii)* Two items intersect among themselves, so it's invalid. *(iii)* An example of a non-axis-parallel packing. *(iv)* A valid packing. . . 8

1.4 At the top, we have the input items with profits labeled below. The left figure shows an optimal packing when rotations are forbidden. The right figure shows an optimal packing when rotations are allowed. 9

2.1 Example run of the Next-Fit algorithm into 5 bins. The items are processed from left to right. 20

2.2 Example run of the Best-Fit algorithm into 4 bins. The difference between Next-Fit and Best-Fit can be seen while packing the sixth item. Both the first and the second bins can accommodate this item, but we will pack it in the first bin since it is the fullest. 21

2.3 The NFDH Algorithm. After packing the first two items on the base of the bin, the third item can't be packed on the same level. Hence, we close the shelf and create a new shelf and continue. 22

3.1 The division of input into super-stages to get rid of the assumption on the knowledge of n. The $(j+1)^{\text{th}}$ super-stage is denoted by Γ_j. Stage Γ_0 contains $n_0 = 1/\delta^3$ items, $\Gamma_0 \cup \Gamma_1$ contains $(1+\mu)n_0$ items, $\Gamma_0 \cup \Gamma_1 \cup \Gamma_2$ contains $(1+\mu)^2 n_0$ items and so on. The last super-stage may not be full, but since it is very small in size compared to the entire input, it doesn't affect the performance of the algorithm. 43

4.1 (The second dimension which is into the paper or the screen is ignored) The layers of items in B' are indicated in red. After packing B', the items in A' are packed as follows: First, they are ordered in non-increasing order of lengths. Then they are stacked up one on top of each other as much as possible (shown in green). Then they are rotated and packed side-by-side as shown in orange. . . 73

4.2 Packing rectangles long in one dimension and short in other dimension. The blue rectangle indicates the first rectangle that couldn't be packed without overlapping. 77

5.1 A corridor which is just a union of 9 subcorridors. 90

5.2 Skewed items in corridors. Dark items indicate the set S_{corr}. Red items indicate the set $S^{\text{nice}}_{\text{cross}}$. They are constant in number. Blue items indicate the set $S^{\text{bad}}_{\text{cross}}$. They carry very marginal profit. Orange items which indicate S_{free} are not contained in any corridor. They are constant in number. 92

5.3 Obtaining containers from a box in two steps. First, we remove the strip (shaded in light grey) with least profit of items completely contained in it (shown in red). We also remove the items partially intersecting that strip (shown in blue. These will be constant in number). Then in the second step, we apply resource augmentation lemma to obtain constant number of containers. 94

5.4 Division of box into smaller boxes: The dark item is the item in $\mathcal{K}(t)$ that overlaps with the box. The dashed lines are the extended edges and the red items are those that will be included in $K(t+1)$. The blue items are the small items that intersect the extended edges. They will be added to TempSmall. This leads to three smaller boxes. 99

5.5 The item (red) will be assigned to the cell that is shaded darker. 101

5.6 Green boxes contribute to a free area of at least 2ε fraction of their intersection with the cell. Red boxes on the other hand may not contribute any empty space. 102

Chapter 1

Introduction

Bin Packing and Knapsack are two closely related combinatorial optimization problems that have served as the cornerstones of approximation algorithms. In the *Bin Packing* (BP) problem, we are given a set of n items each with size in the range $(0, 1]$, and the objective is to pack all the items in the minimum number of unit capacity bins. On the other hand, in the *Knapsack* (KS) problem, each item, while having a size, further has an associated profit and the goal is to choose a maximum profitable subset whose total size is at most one.

Both bin packing and knapsack find several applications in the real-life world. Bin packing appears in cutting stock [GG63], container loading, and job scheduling. On the other hand, knapsack appears in various resource allocation problems with budget limitations. Interestingly, knapsack solutions have also been explored to design encryption protocols [DH76]. Skiena [Ski99], in his survey on WWW traffic, found out that bin packing and knapsack were among the top five problems whose implementations were needed the most. However, both bin packing and knapsack turn out to be NP-hard (see below). Hence, if we need practical run-time, we can only hope for some good approximation algorithms.

1.1 Approximation and Online Algorithms

The complexity class P is the class of decision problems which can be answered "yes" or "no" in time polynomial in the size of input. On the other hand, the class NP contains the decision problems for which a potential solution can be verified to be valid or not in polynomial time. A decision problem $\mathcal{P}$ is said to be NP-Hard if every problem in NP can be reduced to $\mathcal{P}$ in polynomial time. A problem is said to NP-Complete if it lies in NP and is NP-Hard as well. The famous Cook-Levin theorem shows that the boolean satisfiability problem is NP-Complete. Later, Richard Karp in his seminal paper [Kar72] showed that a set of 21 problems, including

Set Cover, Hamiltonian Cycle, Steiner Tree, Knapsack, are NP-Complete. Many other problems such as bin packing, travelling salesman, vehicle routing have also been proven to be NP-Complete. [GJ79]'s compendium shows several such problems. All these problems are of high practical relevance and several attempts have been made to design efficient algorithms. Yet, no polynomial-time algorithm has been found for any of these problems. Thus, it is widely believed that $\mathsf{P} \neq \mathsf{NP}$.

Combinatorial optimization problems are those where the goal is to optimize (minimize or maximize) a certain objective function while satisfying certain constraints. Both bin packing and knapsack are combinatorial optimization problems. It turns out that polynomial-time algorithms for bin packing and knapsack would imply polynomial-time algorithms for their decision versions too. But since the decision versions are NP-Hard, it is believed that there do not exist polynomial-time algorithms for the optimization versions too.

1.1.1 Approximation Algorithms

Since $\mathsf{P} \neq \mathsf{NP}$ is conjectured to be true, it is widely believed that for problems like bin packing and knapsack, there do not exist algorithms which are *exact* while simultaneously being *fast* (polynomial runtime). If an algorithm compromises exactness for polynomial runtime, then it is called an *Approximation Algorithm*.

Once an algorithm is designed, the next natural step is to 'measure' how good it is. Just to give an introduction, we will define these measures slightly informally here. In Section 2.2, we will redefine them more formally. Consider an algorithm $\mathcal{A}$ for some optimization problem and let Opt denote the optimal algorithm. If it is a minimization (resp. maximization) problem, we define the *approximation ratio* (AR) of $\mathcal{A}$ as the maximum value of $\frac{\mathcal{A}(I)}{\mathrm{Opt}(I)}$ $\left(\text{resp. } \frac{\mathrm{Opt}(I)}{\mathcal{A}(I)}\right)$ over all the input instances I. The *asymptotic approximation ratio* (AAR) is just the approximation ratio when we confine ourselves to instances I whose optimal value, $\mathrm{Opt}(I)$, is sufficiently large.

We generally assume that a constant accuracy parameter $\varepsilon \in (0, 1)$ is given as a part of the problem description. Some approximation algorithms, which are the best possible in some sense, are given special names.

- A Polynomial Time Approximation Scheme (PTAS) is an approximation algorithm which runs in polynomial time (assuming ε is a constant) and which has an AR of at most $(1 + \varepsilon)$.

- Similarly, an Asymptotic Polynomial Time Approximation Scheme (APTAS) is an approximation algorithm which runs in polynomial time (assuming ε is a constant) and which has an AAR of at most $(1 + \varepsilon)$.

- A Fully Polynomial Time Approximation Scheme (FPTAS) is a PTAS which runs in time polynomial in both input size as well as $1/\varepsilon$.

- An Asymptotic Fully Polynomial Time Approximation Scheme (AFPTAS) is an APTAS which runs in time polynomial in both input size as well as $1/\varepsilon$.

1.1.2 Online Algorithms

When we talk about a combinatorial optimization problem like bin packing, we implicitly assume that we are in the *offline* model, i.e., the entire input is known to the algorithm before it starts so that we can do some preprocessing. For example, in the (offline) bin packing problem, we assume that the set of input items are given to us beforehand. While this is a reasonable assumption, there also exist several scenarios where the input arrives in parts. Consider, for example, an assembly line in a factory where items of varying sizes arrive on conveyor belts, and each item has to be loaded into a container as soon as it arrives. This is called the *online* model.

Note that computing exact optimal algorithms may not be possible in the online setting even if we allow for exponential runtime. This is because of lack of information beforehand and not because of computational intractability.

To measure the performance of an online algorithm, we use the term *Competitive Ratio* (CR) instead of AAR (the formal definitions of CR and AAR are the same).

1.1.3 Stochastic Models

Above, we introduced the online model where items arrive one by one. We can think of this model as one in which an adversary reveals the input each at a time. When we confine to real-life scenarios though, this model turns out to be very restrictive. To give an idea, it has been proved by [BBG10] that no online algorithm for bin packing can have a competitive ratio lesser than 1.54. However, even simple and almost-linear time greedy algorithms seem to perform fairly well in practice. The reason for this discrepancy is that in the online model, the adversary has too much power, and gets to control both the input set as well as the arrival order. Real-life instances, however, aren't generated by such all-powerful adversaries. One way to weaken the adversary is to include randomness in the model. In this work, we will discuss two stochastic models for online bin packing.

- The i.i.d. model.

- The random-order model.

1.1.3.1 The i.i.d. Model

In the i.i.d. model, the elements of the input are chosen independently from an arbitrary but fixed probability distribution. For example, in the online bin packing problem, the item sizes are sampled from some probability distribution (say, the normal distribution).

The i.i.d. model is of high practical relevance. For example, consider a sequence of files to be processed on a set of servers. In real-life, the file sizes are not entirely arbitrary; rather they follow what is called a *power law distribution*. Similarly, distributions such as Gaussian and Poisson are widespread in practice. In general, when we confine ourselves to practical scenarios, we can assume that the elements of the input to an algorithm are sampled from a distribution. Thus, the i.i.d. model is well motivated, and a number of classical and important problems such as k-server [DEH+17], knapsack [BGK11], and bin packing [RT93a] have been studied under this model. A variant of the online knapsack problem under the i.i.d. model is the budgeted multi-armed bandits problem [MLG04] which has gained a lot of attention in the past few years.

For an online algorithm in the i.i.d. model, we use the term *Expected Competitive Ratio* (ECR) to measure its performance. A slightly informal definition of ECR is as follows. Let $\mathcal{A}$ denote an algorithm for a problem, and let Opt denote the optimal algorithm. If it is a minimization (resp. maximization) problem, the ECR of $\mathcal{A}$ is defined as the maximum ratio $\frac{\mathbb{E}[\mathcal{A}(I)]}{\mathbb{E}[\mathrm{Opt}(I)]}$ $\left(\text{resp. } \frac{\mathbb{E}[\mathrm{Opt}(I)]}{\mathbb{E}[\mathcal{A}(I)]}\right)$ over all instances I whose optimal solutions are sufficiently large.

1.1.3.2 The Random-order Model

The random-order model also weakens the adversary to a certain extent. In this model, the adversary is free to specify the input set, but the order in which the input elements arrive is chosen uniformly randomly from all the permutations of the input elements. For instance, in the random-order model for online bin packing, the items sizes are chosen by the adversary but the order in which they arrive is according to a uniform random permutation.

Again, this is an important relaxation since the order of arrival crucially affects an algorithm's performance, and worst-case arrival orders seldom occur in practice. This model has also been helpful in designing randomized algorithms with good guarantees for problems for which it is difficult to design efficient deterministic algorithms. One such problem is the prototypical *Secretary Problem*. Many classical problems such as Convex Hulls [CS88], Set Cover [GKL22], Adwords [DH09], Bin Packing [AKL21a], and Generalized Assignment Problem [KTRV14] have been studied under this model. See the chapter by Gupta and Singla [GS21] for a primer.

For an online algorithm in the random-order model, we use the notion of *random-order ratio* (RR) to measure the performance. A slightly informal definition of RR is as follows. Let

$\mathcal{A}$ denote an algorithm for a problem, and let Opt denote the optimal algorithm. If it is a minimization problem (resp. maximization problem), the RR of $\mathcal{A}$ is defined as the maximum ratio $\frac{\mathbb{E}[\mathcal{A}(I_\sigma)]}{\mathrm{Opt}(I)}$ $\left(\text{resp. } \frac{\mathrm{Opt}(I)}{\mathbb{E}[\mathcal{A}(I_\sigma)]}\right)$ over all instances I whose optimal solutions are sufficiently large. Here I_σ denotes a uniform random permutation of the input list I.

1.2 Bin Packing, Knapsack, and their Variants

In this section, we will formally discuss the classical bin packing and knapsack problems, and some well-studied variants on which this thesis is built.

1.2.1 Classical Bin Packing

In the classical bin packing problem, we are given a set of n reals (items) $s_1, s_2, \dots, s_n$ such that for each $i \in \{1, 2, \dots, n\}, s_i \in (0, 1]$. The goal is to partition these reals into minimum number of subsets (bins) such that in any bin B, $\sum_{s \in B} s \leq 1$.

It can also be stated as an integer program as follows: We define a *configuration* to be any subset of the input set whose total size is at most 1. Let $\mathcal{C}$ denote the set of all the configurations. Then, the bin packing problem asks to find a minimum cardinality subset of $\mathcal{C}$ such that each item is contained in at least one of the subsets. An integer program for this formulation is as follows.

$$\begin{aligned} \min \quad & \sum_{C \in \mathcal{C}} x_C \\ \text{s.t.} \quad & \sum_{C \ni s_i} x_C \geq 1 \quad \text{for all } i \in \{1, 2, \dots, n\} \\ & x_C \in \{0, 1\} \quad \text{for all } C \in \mathcal{C} \end{aligned}$$

An example instance of bin packing and its optimal packing is shown in Fig. 1.1.

1.2.2 Online Bin Packing

In online bin packing, the items arrive one-by-one. As soon as an item arrives, it must be packed into a bin immediately and irrevocably. Typically, we assume that in a completely online setting, even the number of items n is unknown. If we assume the knowledge of n, it is known as a *semi-online* setting.

A very folklore online algorithm for bin packing is Next-Fit: we keep exactly one bin *open*. If an item fits in this open bin, we pack it there. Otherwise, we open a new bin and continue (see Section 2.4.1 for more details on Next-Fit). A non-example of online algorithms is

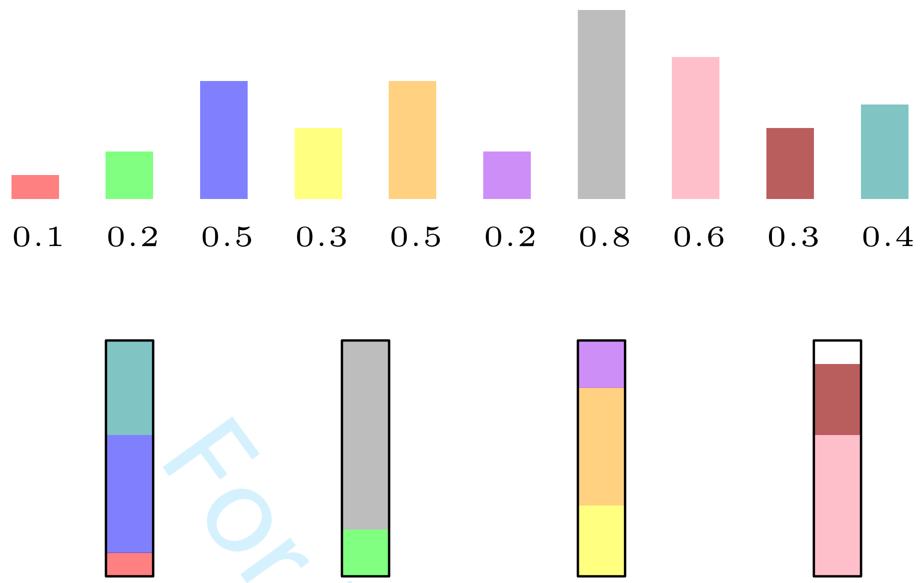


Figure 1.1: An example instance of bin packing. The sizes of the items are labeled below. An optimal packing requires 4 bins.

Next-Fit-Decreasing, which first sorts the items in decreasing order of sizes (we don't have this luxury in the online setting!) and then applies Next-Fit.

A simple online algorithm which gained a lot of importance due to its excellent performance in practice is Best-Fit. In this algorithm, we keep all the bins open. Whenever an item arrives, we look at all the bins that can accommodate this item. Among these *feasible* bins, we choose the one with the least empty space (breaking ties arbitrarily). If no bin can accommodate the item, we open a new bin. We will study more about the Best-Fit algorithm when we deal with online bin packing under the random-order model in Section 3.4.

Another simple and practical algorithm is First-Fit, which packs an item in the oldest feasible bin, opening a new bin if required. There exist several other online algorithms such as Worst-Fit, Harmonic algorithm [LL85a].

1.2.3 Classical Knapsack

A closely related problem to bin packing is the Knapsack problem. In this problem, we are given a set of n items. The i^{th} item has a size given by $s(i) \in [0, 1]$ and profit given by $p(i) > 0$. The objective is to choose a subset $B \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{j \in B} p(j)$ is maximized while satisfying the condition $\sum_{j \in B} s(j) \leq 1$.

Like bin packing, the knapsack problem also can be formulated as an integer program as follows.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p(i)x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} s(i)x_i \leq 1 \\ & x_i \in \{0,1\} \quad \text{for all } i \in \{1,2,\ldots,n\} \end{aligned}$$

An example instance of knapsack is shown in Fig. 1.2.

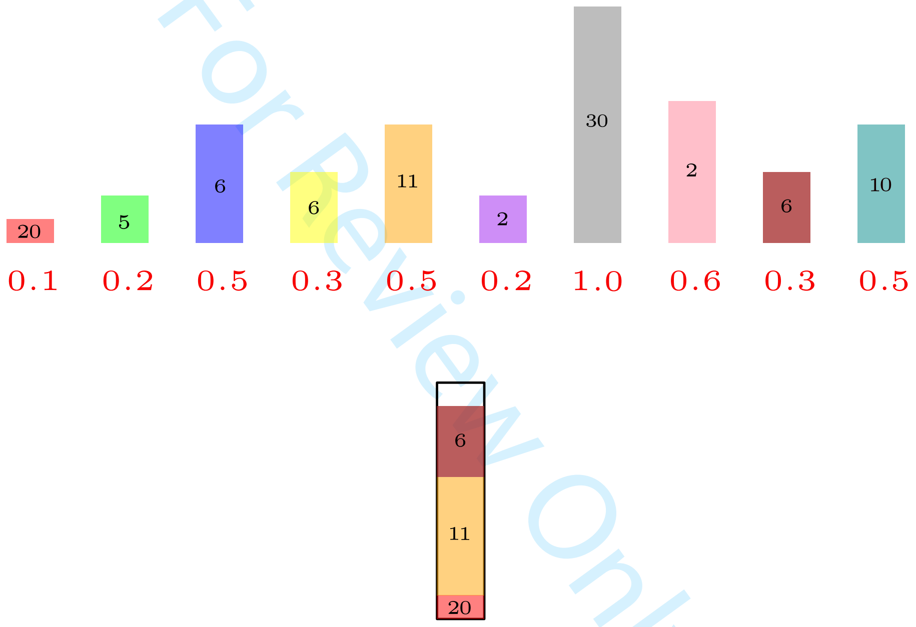


Figure 1.2: The sizes are labeled below the items and the profits are labeled on the items. An optimal packing into a knapsack is shown.

1.2.4 Geometric Knapsack

The d-D Geometric Knapsack problem is an important generalization of the classical knapsack problem. Let d be a positive integer fixed beforehand (say $d = 2$ or 3 or 100). In the d-D geometric knapsack problem, we are given a set of d-D (read "d dimensional") hypercuboids (items), each associated with a positive profit. The objective is to pack a subset of these items in an *axis-parallel, non-overlapping* fashion into a unit d-D hypercube (knapsack) such that the packed profit is maximized. By an axis-parallel packing, we mean that each facet of a packed item must be parallel to some facet of the knapsack. An non-overlapping packing is in which no two items intersect so that their common volume is non-zero. We also require all the packed items to be completely contained inside the knapsack. See Fig. 1.3 for an illustration of valid and invalid packings in 3 dimensions.

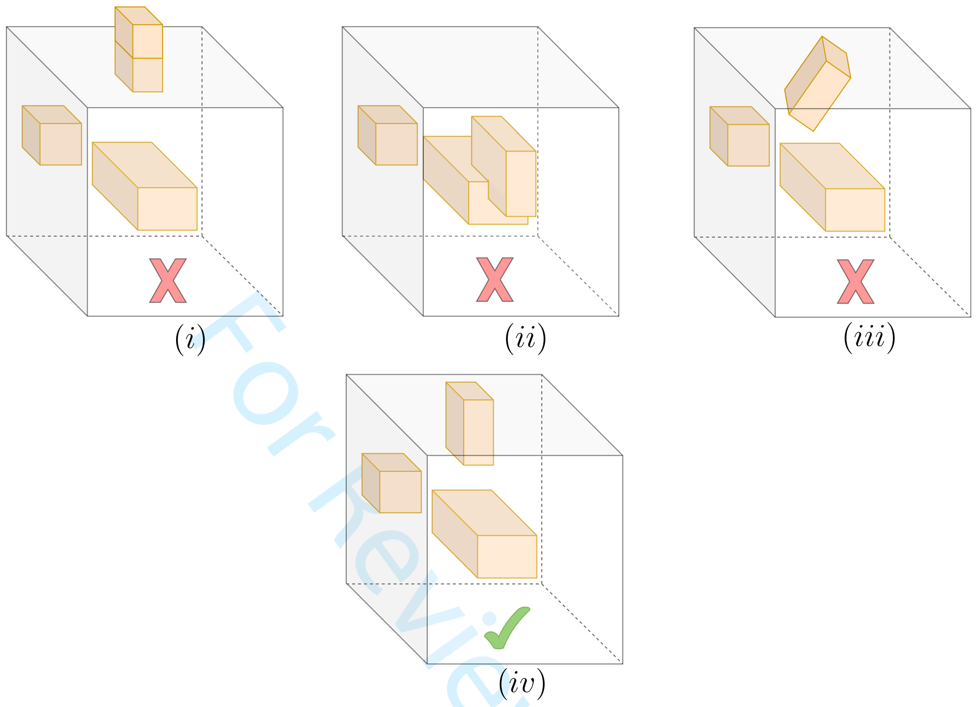


Figure 1.3: Three invalid packings and one valid packing. *(i)* An item protrudes from the knapsack, so it's invalid. *(ii)* Two items intersect among themselves, so it's invalid. *(iii)* An example of a non-axis-parallel packing. *(iv)* A valid packing.

There are two well-studied variants of the d-D geometric knapsack problem depending on whether or not we can rotate the items. In the first variant, the orientations of the items are fixed as a part of the input, and hence rotations are not allowed. In the second variant, rotations are allowed as long as the items remain axis-parallel. Note that in this variant any item can be placed in at most $d!$ orientations. Other variants such as "this-side-up" have been studied in heuristic designs.

When rotations are not allowed, the assumption that the knapsack is a unit hypercube holds without loss of generality because we can scale the knapsack and the items appropriately. However, in the case when rotations are allowed, this assumption is explicit.

Apart from the case of $d = 1$, d-D geometric knapsack is well-studied for the cases of $d = 2, 3$ because of their practical relevance. When $d = 2$, the problem is also called as *Rectangle Knapsack*. See Fig. 1.4 for an illustration of rectangle knapsack.

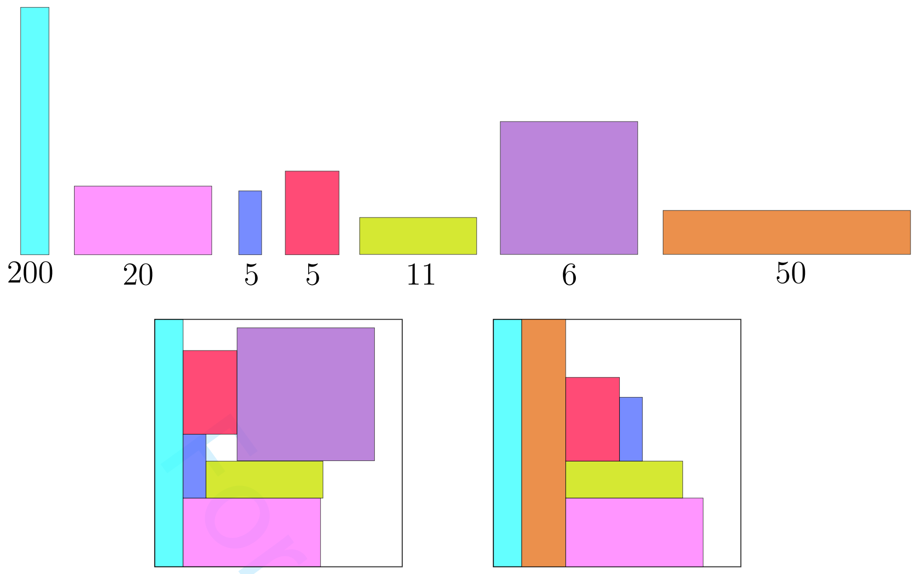


Figure 1.4: At the top, we have the input items with profits labeled below. The left figure shows an optimal packing when rotations are forbidden. The right figure shows an optimal packing when rotations are allowed.

1.2.5 Vector Knapsack

The d-D Vector Knapsack problem is also an important generalization of classical knapsack. Consider a scenario where we are to load a maximum profitable subset of items into a knapsack. However, the knapsack might have several other constraints in addition to capacity constraint such as volume constraint. Then, we will have to take into account both mass and volume of each input item.

Let d be a positive integer fixed beforehand. In the d-D vector knapsack problem, each input item is a vector (a tuple) with weights in d dimensions and a profit. The objective is to choose a maximum profitable subset such that the total weight in any of the d dimensions is at most 1. Just like classical knapsack, the vector packing problem also can be stated as an integer program. Let the input set of vectors be denoted by I and let $n = |I|$. For an item $i \in I$, let $v_j(i)$ denote the weight of the i^{th} item in the j^{th} dimension ($j \in \{1, 2, \dots, d\}$). As usual, let $p(i)$ denote the profit of the i^{th} item. Then the integer program formulation is

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p(i)x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} v_j(i)x_i \leq 1 \quad \text{for all } j \in \{1,2,\ldots,d\} \\ & x_i \in \{0,1\} \quad \text{for all } i \in \{1,2,\ldots,n\} \end{aligned}$$

The d-D vector knapsack problem is also known as *multidimensional knapsack*.

1.3 Contribution of this Thesis

This thesis mainly focuses on the following four problems related to bin packing and knapsack.

1. Stochastic Online Bin Packing (Joint work with Nikhil Ayyadevara, Rajni Dabas, Arindam Khan).
2. 3-D Geometric Knapsack (Joint work with Arindam Khan).
3. Generalized Multidimensional Knapsack (Joint work with Arindam Khan, Eklavya Sharma).
4. d-D Hypercube Knapsack (Joint work with Klaus Jansen, Arindam Khan, Marvin Lira).

1.3.1 Stochastic Online Bin Packing

We consider two stochastic models of online bin packing: *(i)* The i.i.d. model *(ii)* The random-order model.

1.3.1.1 Online Bin Packing under the i.i.d. Model

We achieve near-optimal performance guarantee for the online bin packing problem under the i.i.d. model, thus settling the problem. For any arbitrary unknown distribution $\mathcal{D}$ on $(0,1]$ and an accuracy parameter ε, we devise an online meta-algorithm that takes an α-asymptotic approximation algorithm as input and provides a polynomial-time $(\alpha+\varepsilon)$-competitive algorithm. Note that both the distribution $\mathcal{D}$ as well as the number of items are unknown in this case.

Theorem 1.1. *Let $\varepsilon \in (0,1)$ be a constant parameter. For online bin packing under the i.i.d. model, where n items are sampled from an unknown distribution $\mathcal{D}$, given an offline algorithm $\mathcal{A}_\alpha$ with an AAR of α and runtime $\beta(n)$, there exists an online (meta-)algorithm which returns a solution with an ECR of $(\alpha+\varepsilon)$ and runtime $O(\beta(n))$.* [1]

[1]The $O(\cdot)$ notation hides some constants depending on ε here.

Using an AFPTAS for bin packing (e.g. [dlVL81]) as $\mathcal{A}_\alpha$, we obtain the following corollary.

Corollary 1.1.1. *There exists an algorithm for online bin packing under the i.i.d. model with an ECR of* $(1+\varepsilon)$ *for any* $\varepsilon \in (0,1)$.

1.3.1.2 Online Bin Packing under the Random-order Model

We also study online bin packing under the random-order model. Specifically, we analyze the well-known Best-Fit algorithm in the random-order model. Best-Fit is very well-studied due to its excellent performance in practice while being simple. Back in 1996, Kenyon proved that Best-Fit has a random-order ratio (RR) of at most 3/2. Since then, no progress has been made until only recently, when [AKL21a] showed that if all the item sizes are greater than 1/3, Best-Fit has an RR of at most 5/4. We show that, somewhat surprisingly, in this case, Best-Fit actually has an RR of exactly 1.

Theorem 1.2. *For online bin packing under the random-order model, Best-Fit achieves a random-order ratio of* 1 *when all the item sizes are in* $(1/3, 1]$.

Next, we study the 3-partition problem, a special case of bin packing when all the item sizes are in $(1/4, 1/2]$. This is known to be an extremely hard case [HR17a]. We break the barrier of 3/2 in this special case, by showing that Best-Fit attains a random-order ratio of at most 1.4941.

Theorem 1.3. *For online bin packing under the random-order model, Best-Fit achieves an random-order ratio of* ≤ 1.4941 *when all the item sizes are in* $(1/4, 1/2]$.

As 3-partition instances are believed to be the hardest instances for bin packing, our result gives a strong indication that the RR of Best-Fit might be strictly less than 3/2.

1.3.2 3-D Geometric Knapsack

For the 3-D Geometric Knapsack problem, the best approximation ratios known prior to this work are $(7+\varepsilon)$ when rotations of items are not allowed, and $(5+\varepsilon)$ if we are allowed to rotate the items. Both the results have been given by [DHJ+07].

In this work, we first show alternate algorithms which achieve the same approximation ratios ($(7+\varepsilon), (5+\varepsilon)$ respectively). The algorithms and their analyses are far simpler when compared to [DHJ+07]. Then, we improve the approximation ratio in the case when rotations are allowed to $(31/7+\varepsilon)$.

We also consider two special cases of this problem: The cardinality case and the profit-equals-volume case. In the cardinality case, each item has unit profit. Hence, in this case, the problem boils down to packing the maximum number of items in the knapsack. In the profit-equals-volume case, the profit of an item is just given by its volume. So, solving the 3-D knapsack problem in this case is equivalent to maximizing the packed volume in a knapsack.

In the cardinality case, when rotations are not allowed, we give a $(6 + \varepsilon)$ approximation algorithm. When rotations are allowed, this factor is improved to $(4 + \varepsilon)$. In the profit-equals-volume case, when rotations are allowed, we give a $(3+\varepsilon)$-approximate algorithm. See Table 1.1 for a summary of our results for the 3-D Knapsack problem.

	With Rotations	Without Rotations
General	$\frac{31}{7} + \varepsilon$	$7 + \varepsilon$
Cardinality	$\frac{24}{7} + \varepsilon$	$6 + \varepsilon$
Profit=Volume	$3 + \varepsilon$	$7 + \varepsilon$

Table 1.1: Our results for the 3-D Knapsack problem

1.3.3 Generalized Multidimensional Knapsack

We also introduce the theoretical study of a variant of knapsack with both geometric and vector constraints. In practice, while packing objects, in addition to geometric constraints, we also need to consider vector constraints such as weight, volume etc. For instance, while packing items into a cargo, we need to ensure that the total weight of the items packed doesn't exceed the capacity of the cargo. We also need to ensure that the volume of the items that we intend to pack doesn't exceed the cargo's volume. Hence, it is natural to assume that each item, along with its geometric dimensions, also has some weights associated with it.

With this as the motivation, we study the Rectangle knapsack problem with vector constraints. In this problem, each item is a rectangle with d weights each lying in range $[0, 1]$, where d is a constant fixed beforehand. We will denote this problem as $(2, d)$-Knapsack. Each input item i has a width given by $w(i)$, height given by $h(i)$, and profit denoted as $p(i)$. Its weight in the j^{th} dimension is given by $v_j(i)$. The objective is to pack a maximum profit subset of items J into a unit square knapsack in an axis-parallel, non-overlapping manner such that for any $j \in [d], \sum_{i \in J} v_j(i) \leq 1$.

For this problem, we devise a $(2+\varepsilon)$-approximation algorithm. En route, we study a variant of the well-known Maximum Generalized Assignment Problem (Max-GAP). In the Max-GAP problem, we are provided with a set of machines with designated capacities, and a set of items; an item's size and value depends on the machine to which it is going to be assigned. The objective is to assign a subset of items to machines such that the obtained value is maximized while making sure that no machine's capacity is breached. We define a variant of the Max-GAP problem called Vector-Max-GAP. In this problem, we additionally have a d-dimensional weight vector associated with every item and a d-dimensional global weight constraint vector on the whole setup of machines. The objective is to find the maximum value obtainable so that no machine's capacity is breached and the overall weight of items does not cross the global weight constraint in any of the d dimensions. For the Vector-Max-GAP problem, we achieve a PTAS.

1.3.4 d-D Hypercube Knapsack

The d-D Hypercube Knapsack problem is a special case of the d-D Geometric Knapsack problem where each item, instead of being a d-D hypercuboid, is a d-D hypercube.

Note that for $d \geq 2$, this problem is not (necessarily) a generalization of the classical knapsack. To see this, note that an optimal algorithm for 2-D hypercube knapsack wouldn't directly imply an optimal algorithm for the classical knapsack problem. Hence, the NP-Hardness of the classical knapsack doesn't trivially imply that d-D Hypercube Knapsack ($d \geq 2$) is NP-Hard. Back in 1990, [LTW+90] proved that the Square Knapsack problem (the special case when $d = 2$) is NP-Hard. But the NP-Hardness for $d \geq 3$ has only been proved recently in 2015 by [LCC15].

We consider two special cases of this problem, namely the *cardinality* case and the *bounded-density* case. In the cardinality case, each input item has unit profit. Hence, the objective of the problem is to maximize the number of items packed. In the bounded-density case, the profit/volume value (which is also called *profit density*) of each item lies in the range $[1, r]$ where r is a constant. In the special case of $r = 1$, i.e., when the profit of an item equals its volume, the objective is to pack as much volume as possible. For both the cases, we devise PTASes.

Chapter 2

Notations and Preliminaries

In this chapter, we will introduce some notations and discuss some preliminaries that will be used throughout the thesis.

2.1 General Notations

For any positive integer n, we will denote the set $\{1, 2, \ldots, n\}$ by $[n]$.

Given an instance I for an optimization problem and an algorithm $\mathcal{A}$ for that problem, we denote by $\mathcal{A}(I)$ the value of the objective function obtained by running the algorithm $\mathcal{A}$ on the instance I.

Functions depending on ε. Generally, for any NP-Hard optimization problem, we assume that a constant accuracy parameter $\varepsilon \in (0, 1)$ is given as a part of the problem description. So, if an algorithm runs in time $n^{1/\varepsilon}$ for a problem, where n is the input size, we consider it to be polynomial runtime unless otherwise specified. Similarly, functions of ε like $1/\varepsilon, (1/\varepsilon)^{1/\varepsilon}$ are considered constants. However, at some places, for clarity, we make this dependence on ε explicit as follows. The notation $O_\varepsilon(f(n))$ (where n is the input size, and f is some function) is shorthand for a function of the form $g(\varepsilon)f(n)$. In the entire thesis, $g(\varepsilon)$ increases with decreasing ε (e.g., $g(\varepsilon) = 1/\varepsilon$ or $g(\varepsilon) = (1/\varepsilon)!$). Similarly, suppose $\varepsilon_1, \varepsilon_2$ are two parameters which in turn depend only on ε. Then the notation $O_{\varepsilon_1,\varepsilon_2}(f(n))$ is a shorthand for a function of the form $g_1(\varepsilon_1, \varepsilon_2)f(n)$.

2.2 Performance Measures of Algorithms

All the problems that we discuss in this thesis are NP-Hard. Hence, if we wish to design polynomial-time algorithms, we can only approximately solve these problems unless $\mathsf{P} = \mathsf{NP}$.

There are multiple ways to quantitatively measure the performance of approximation algorithms. Two of the most common measures are *Approximation Ratio* and *Asymptotic Approximation Ratio*. For online algorithms, we use the notion of *Competitive Ratio* to measure the performance. We will formally define these notions in the following subsections.

2.2.1 Approximation Ratio

Consider a combinatorial optimization problem $\mathcal{P}$. Let $\mathcal{I}$ denote the set of all the input instances of the problem. Let $\mathcal{A}$ denote an (approximation) algorithm to $\mathcal{P}$ and let Opt denote an optimal algorithm to $\mathcal{P}$.

If $\mathcal{P}$ is a maximization problem (e.g., Knapsack where we try to maximize the profit), then the *Approximation Ratio* (AR) of the algorithm $\mathcal{A}$ is defined as follows:

$$\mathrm{AR}(\mathcal{A}) := \sup_{I \in \mathcal{I}} \left\{ \frac{\mathrm{Opt}(I)}{\mathcal{A}(I)} \right\}$$

On the other hand, if $\mathcal{P}$ is a minimization problem (e.g., Bin packing where we try to minimize the number of bins), then the *Approximation Ratio* (AR) of the algorithm $\mathcal{A}$ is defined as follows:

$$\mathrm{AR}(\mathcal{A}) := \sup_{I \in \mathcal{I}} \left\{ \frac{\mathcal{A}(I)}{\mathrm{Opt}(I)} \right\}$$

An equivalent definition of AR is as follows: it is the minimum constant α for which for every instance $I \in \mathcal{I}$,

$$\mathcal{A}(I) \geq \frac{1}{\alpha}\mathrm{Opt}(I) \qquad \text{(in case of maximization problems)}$$

$$\mathcal{A}(I) \leq \alpha\mathrm{Opt}(I) \qquad \text{(in case of minimization problems)}$$

Remark 2.1. *Be it a minimization problem or a maximization problem, the* AR *of an algorithm is always at least* 1. *If we confine ourselves to polynomial-time algorithms, unless* $\mathsf{P} = \mathsf{NP}$*, the* AR *can't be equal to* 1. *The lesser the* AR *of an algorithm, the better it is.*

2.2.2 Asymptotic Approximation Ratio

Besides approximation ratio, another commonly used notion to measure the performance of an algorithm is asymptotic approximation ratio. Consider, for example, the problem of bin packing. Using a reduction from the partition problem, it can be proved that it is NP-Hard to decide if a given input set can be packed in at most two bins or not. Hence, no polynomial-time algorithm for bin packing can have an AR strictly better than 3/2. However, such hard instances typically have very small optimal value and don't occur in practice. Hence, we define measures that are only concerned about instances for which the optimal values are large. One such measure is asymptotic approximation ratio.

As before, let $\mathcal{P}$ be an optimization problem, $\mathcal{I}$ be the set of all input instances, $\mathcal{A}$ be an (approximation) algorithm, and Opt be an optimal algorithm.

If $\mathcal{P}$ is a maximization problem, then the *Asymptotic Approximation Ratio* (AAR) of the algorithm $\mathcal{A}$ is defined as follows:

$$\mathrm{AAR}(\mathcal{A}) := \limsup_{m\to\infty}\left(\sup_{I\in\mathcal{I}}\left\{\frac{\mathrm{Opt}(I)}{\mathcal{A}(I)}\middle|\mathrm{Opt}(I)=m\right\}\right)$$

On the other hand, if $\mathcal{P}$ is a minimization problem, then the *Asymptotic Approximation Ratio* (AAR) of the algorithm $\mathcal{A}$ is defined as follows:

$$\mathrm{AAR}(\mathcal{A}) := \limsup_{m\to\infty}\left(\sup_{I\in\mathcal{I}}\left\{\frac{\mathcal{A}(I)}{\mathrm{Opt}(I)}\middle|\mathrm{Opt}(I)=m\right\}\right)$$

Equivalently, AAR can be defined as the minimum α such that for every instance I whose optimal value is sufficiently large,

$$\mathcal{A}(I) \geq \frac{1}{\alpha}\mathrm{Opt}(I) - o(\mathrm{Opt}(I)) \qquad \text{(in case of maximization problems)}$$
$$\mathcal{A}(I) \leq \alpha\mathrm{Opt}(I) + o(\mathrm{Opt}(I)) \qquad \text{(in case of minimization problems)}$$

Remark 2.2. *Just like* AR*, the* AAR *of any algorithm is always at least* 1 *and the lesser the* AAR *of an algorithm, the better it is. However, unlike* AR*, the* AAR *of an algorithm can be exactly* 1*.*

2.2.3 Competitive Ratio

For an online algorithm, we use the term *Competitive Ratio* (CR) instead of *Asymptotic Approximation Ratio* (AAR). The definition is exactly the same, i.e., the CR of an online algorithm $\mathcal{A}$ is defined as

$$\mathrm{CR}(\mathcal{A}) := \limsup_{m\to\infty}\left(\sup_{I\in\mathcal{I}}\left\{\frac{\mathrm{Opt}(I)}{\mathcal{A}(I)}\middle|\mathrm{Opt}(I)=m\right\}\right)$$

for a maximization problem, and

$$\mathrm{CR}(\mathcal{A}) := \limsup_{m\to\infty}\left(\sup_{I\in\mathcal{I}}\left\{\frac{\mathcal{A}(I)}{\mathrm{Opt}(I)}\middle|\mathrm{Opt}(I)=m\right\}\right)$$

for a minimization problem.

Note that we defined competitive ratio only in the asymptotic case and not for all the instances. Also, here we compare the objective value of an online algorithm to that of the optimal algorithm which is offline, i.e., it has apriori knowledge of the input set.

2.2.4 Expected Competitive Ratio

In Section 1.1.3, we discussed that the online model is too pessimistic, and one way to counter it is to somehow include randomness into the model. Hence, in stochastic models, to measure the performance of an algorithm, we define the notion of *Expected Competitive Ratio* (ECR). We will confine ourselves to stochastic online bin packing while defining ECR as we do not use it elsewhere in this thesis. Let $\mathcal{A}$ be an online algorithm for bin packing and let Opt denote the optimal algorithm.

2.2.4.1 ECR in the i.i.d. Model

Consider the online bin packing problem under the i.i.d. model. Suppose the item sizes arrive from a distribution $\mathcal{D}$. Let I_n denote a random sequence of n items sampled from $\mathcal{D}$. Then the ECR of $\mathcal{A}$ is defined as

$$\mathrm{ECR}(\mathcal{A}) := \limsup_{n\to\infty}\left(\frac{\mathbb{E}\left[\mathcal{A}(I_n)\right]}{\mathbb{E}\left[\mathrm{Opt}(I_n)\right]}\right)$$

We can equivalently define ECR in the i.i.d. model as the least α such that for sufficiently large n,

$$\mathbb{E}\left[\mathcal{A}(I_n)\right] \le \alpha\mathbb{E}\left[\mathrm{Opt}(I_n)\right] + o(\mathbb{E}\left[\mathrm{Opt}(I_n)\right])$$

In the above definition we could use $o(\mathbb{E}\left[\mathrm{Opt}(I_n)\right])$ because $n \to \infty$ directly implies that $\mathbb{E}\left[\mathrm{Opt}(I_n)\right] \to \infty$ given that the support[1] of $\mathcal{D}$ is non-empty. This is because $\mathcal{D}$ is fixed beforehand and not a part of the input.

[1]Support of a distribution is defined as the set of values which occur with non-zero probability.

2.2.4.2 ECR in the Random-order Model

Let $\mathcal{I}$ denote the set of all input instances of bin packing. For an input list I, let I_σ denote the list obtained by permuting I according to a permutation σ chosen uniformly at random. Just to avoid the confusion with ECR in the i.i.d. model, we use the term *random-order ratio*. The random-order ratio of the algorithm $\mathcal{A}$ is denoted by $\mathrm{RR}_{\mathcal{A}}^{\infty}$ and is defined as

$$\mathrm{RR}_{\mathcal{A}}^{\infty} := \limsup_{m\to\infty} \left(\sup_{I\in\mathcal{I}} \left\{ \frac{\mathbb{E}\left[\mathcal{A}(I_\sigma)\right]}{\mathrm{Opt}(I)} \middle| \mathrm{Opt}(I) = m \right\} \right)$$

Remark 2.3. *For any permutation σ, $\mathrm{Opt}(I_\sigma) = \mathrm{Opt}(I)$ as for the optimal offline algorithm, the input order doesn't matter.*

Equivalently, $\mathrm{RR}_{\mathcal{A}}^{\infty}$ can be defined as the minimum α such that for every instance I whose optimal value is sufficiently large,

$$\mathbb{E}\left[\mathcal{A}(I_\sigma)\right] \le \alpha\mathrm{Opt}(I) + o(\mathrm{Opt}(I))$$

2.3 Special Approximation Schemes

Let ε be a constant accuracy parameter. Consider an optimization problem $\mathcal{P}$. As before, let $\mathcal{P}$ be an optimization problem, $\mathcal{I}$ be the set of all input instances, $\mathcal{A}$ be an (approximation) algorithm, and Opt be an optimal algorithm.

Approximation algorithms are given special names if they satisfy certain efficiency constraints.

Definition 2.1 (Polynomial Time Approximation Scheme (PTAS))**.** *The algorithm $\mathcal{A}$ is said to be a PTAS iff it runs in polynomial time, and for all input instances $I \in \mathcal{I}$,*

$$\mathcal{A}(I) \le (1+\varepsilon)\mathrm{Opt}(I) \qquad \text{(If } \mathcal{P} \text{ is a minimization problem)}$$
$$\mathcal{A}(I) \ge (1-\varepsilon)\mathrm{Opt}(I) \qquad \text{(If } \mathcal{P} \text{ is a maximization problem)}$$

In other words, PTAS is a polynomial-time approximation algorithm if its approximation factor can be made arbitrarily close to 1.

Definition 2.2 (Asymptotic Polynomial Time Approximation Scheme (APTAS))**.** *The algorithm $\mathcal{A}$ is said to be an APTAS iff it runs in polynomial time, and for all input instances $I \in \mathcal{I}$*

such that $\mathrm{Opt}(I)$ *is sufficiently large,*

$$\mathcal{A}(I) \leq (1+\varepsilon)\mathrm{Opt}(I) + o(\mathrm{Opt}(I)) \qquad \text{(If } \mathcal{P} \text{ is a minimization problem)}$$
$$\mathcal{A}(I) \geq (1-\varepsilon)\mathrm{Opt}(I) - o(\mathrm{Opt}(I)) \qquad \text{(If } \mathcal{P} \text{ is a maximization problem)}$$

Informally speaking, an APTAS is a polynomial-time approximation algorithm if its asymptotic approximation factor can be made arbitrarily close to 1.

Definition 2.3 (Fully Polynomial Time Approximation Scheme (FPTAS))**.** *An FPTAS is a PTAS whose runtime is polynomial in both the input size and* $1/\varepsilon$ *where* ε *is the accuracy parameter, i.e., its runtime must be of the form* $O\left(n^{c_1}\left(\frac{1}{\varepsilon}\right)^{c_2}\right)$ *for some absolute constants* c_1, c_2*, where* n *is the input size.*

For example, a PTAS which runs in time $O\left(\frac{n^2}{\varepsilon^3}\right)$ is an FPTAS, but not one which runs in time, say, $O\left(n^{\frac{1}{\varepsilon}}\right)$.

Definition 2.4 (Asymptotic Fully Polynomial Time Approximation Scheme (AFPTAS))**.** *An AFPTAS is an APTAS whose runtime is polynomial in both the input size and* $1/\varepsilon$ *where* ε *is the accuracy parameter.*

Definition 2.5 (Efficient Polynomial Time Approximation Scheme (EPTAS))**.** *An EPTAS is a PTAS whose runtime is of the form* $O\left(n^{c_1} f\left(\frac{1}{\varepsilon}\right)\right)$ *for some absolute constant* c_1 *and a real-valued function* f*.*

For example, a PTAS which runs in time $O\left(\frac{n^2}{\varepsilon^{1/\varepsilon}}\right)$ is also an EPTAS.

Definition 2.6 (Quasi Polynomial Time Approximation Scheme (QPTAS))**.** *The algorithm* $\mathcal{A}$ *is said to be a QPTAS iff for all input instances* $I \in \mathcal{I}$*,*

$$\mathcal{A}(I) \leq (1+\varepsilon)\mathrm{Opt}(I) \qquad \text{(If } \mathcal{P} \text{ is a minimization problem)}$$
$$\mathcal{A}(I) \geq (1-\varepsilon)\mathrm{Opt}(I) \qquad \text{(If } \mathcal{P} \text{ is a maximization problem)}$$

However, its runtime can be of the form $O\left(n^{(\log n)^{c_1} f(1/\varepsilon)}\right)$ *for some absolute constant* c_1 *and real valued function* f*.*

Note that a QPTAS need not be a PTAS.

2.4 Greedy Packing Algorithms

In this section, we will discuss some simple packing algorithms. First, we will review two well-known greedy online algorithms for bin packing, namely Next-Fit and Best-Fit. Then, we will discuss Next-Fit Decreasing Height (NFDH), a greedy geometric packing algorithm.

2.4.1 Next-Fit and Best-Fit

2.4.1.1 Next-Fit

Next-Fit is one of the simplest algorithms for bin packing. It works as follows: we keep exactly one bin in the working memory (we call this an *open* bin). When an item arrives, we check if it fits in the open bin. If it fits, we pack it there; otherwise, we *close* this bin, open a new bin, and pack it there. An illustration of Next-Fit is given in Fig. 2.1.

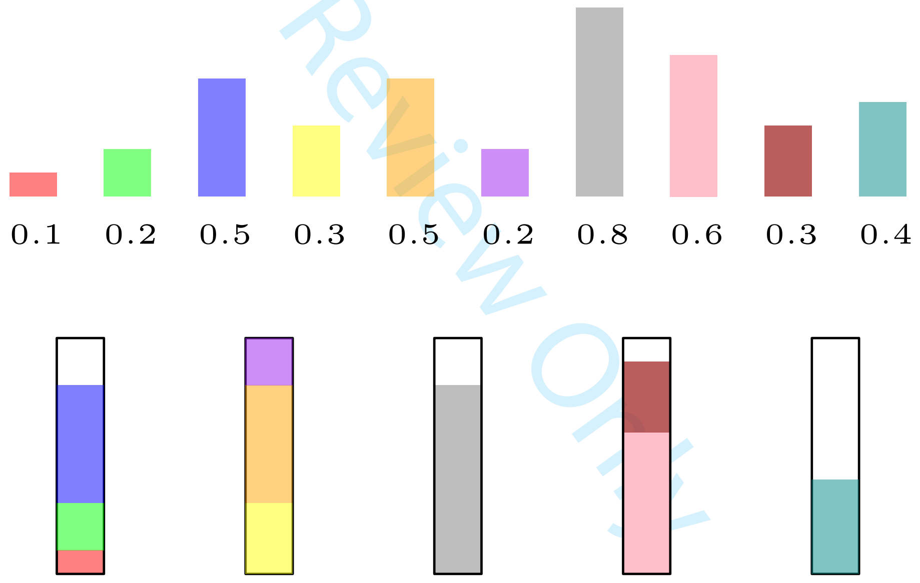


Figure 2.1: Example run of the Next-Fit algorithm into 5 bins. The items are processed from left to right.

It can be easily shown that the AAR of Next-Fit is 2.

Lemma 2.4. *For any bin packing instance I, Next-Fit uses at most* $2\text{Opt}(I) + 1$ *number of bins.*

Proof. Let $\mathcal{W}(I)$ denote the total size of items in I. Say $B_1, B_2, \ldots, B_m$ are the bins opened by Next-Fit. For any $i \in [m-1]$, B_{i+1} has been opened since B_i can't accommodate the first item packed in B_{i+1}. Therefore, $\mathcal{W}(B_i) + \mathcal{W}(B_{i+1}) > 1$. Noting that $\mathcal{W}(I)$ is a lower bound on

Opt(I),

$$\mathrm{Opt}(I) \geq \mathcal{W}(B_1) + \mathcal{W}(B_2) + \cdots + \mathcal{W}(B_m) > \frac{m-1}{2}$$

Hence, the number of bins used by Next-Fit is at most $2\mathrm{Opt}(I) + 1$. □

2.4.1.2 Best-Fit

Best-Fit is another greedy algorithm widely used in practice for bin packing. Unlike Next-Fit, we keep all the bins open in Best-Fit. Once an item arrives, we check if there are any of the bins can accommodate this item. If such a bin exists, we pack the item in the fullest feasible bin. Otherwise, we open a new bin and pack it there. More formally, suppose the current item has size x and the bins used by Best-Fit up till now are $B_1, B_2, \ldots, B_k$. If there exists $i \in [k]$ such that $\mathcal{W}(B_i) + x \leq 1$, then we pack x in the bin B_j where $j = \arg\max_i\{\mathcal{W}(B_i) | \mathcal{W}(B_i) + x \leq 1\}$. If no such i exists, we open a new bin B_{k+1} and pack x there. An illustration of Best-Fit is given in Fig. 2.2.

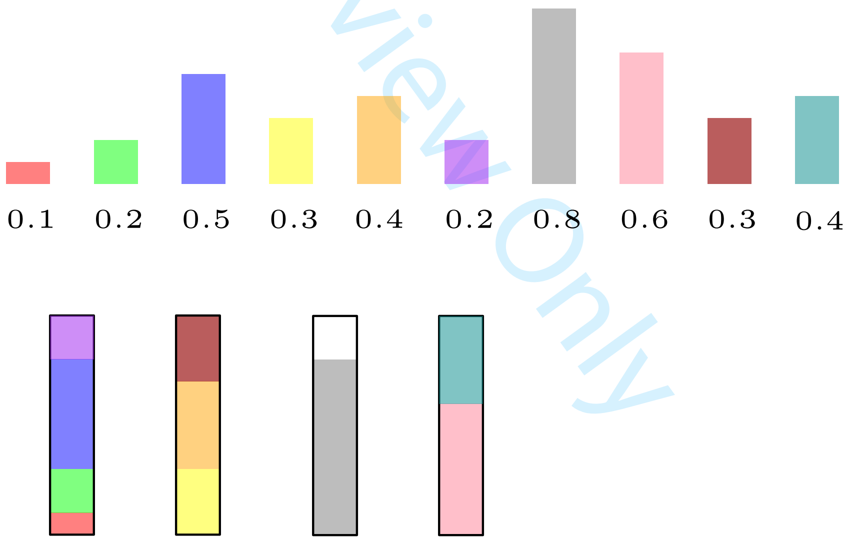


Figure 2.2: Example run of the Best-Fit algorithm into 4 bins. The difference between Next-Fit and Best-Fit can be seen while packing the sixth item. Both the first and the second bins can accommodate this item, but we will pack it in the first bin since it is the fullest.

2.4.2 Next-Fit Decreasing Height

Next-Fit Decreasing Height (NFDH) introduced in [CGJT80] is a widely used algorithm to pack rectangles in a bigger rectangular bin. It works as follows. First we order the rectangles in the decreasing order of their heights. We then place the rectangles greedily on the base of the bin until we do not cross the boundaries of the bin. At this point, we "close" the shelf and shift the base to top edge of the first rectangle and continue the process (see Fig. 2.3).

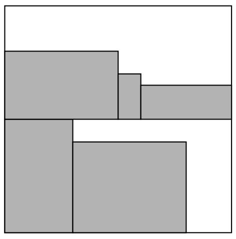


Figure 2.3: The NFDH Algorithm. After packing the first two items on the base of the bin, the third item can't be packed on the same level. Hence, we close the shelf and create a new shelf and continue.

Now, we state a well-known and important lemma about NFDH. The proof can be found in [BCJ+09].

Lemma 2.5. *Let S be a set of rectangles and let w, h respectively denote the largest width and largest height in S. Consider a rectangular bin of dimensions $W \times H$. Then if the area of rectangles in S is at most $(W - w)(H - h)$, NFDH packs all the rectangles into the bin.*

The above lemma informally says that NFDH works very well if all the rectangles are small in both the dimensions as compared to the dimensions of the bin.

2.4.2.1 NFDH in Higher Dimensions

We can also recursively define NFDH for packing hypercuboids. For simplicity, let's consider the three dimensional case. We initialize the current level of the bin to 0. We first ignore the third dimension (say, the height), and use NFDH (two-dimensional) to pack as many items as possible on the base of the bin. Then we close the shelf by updating the current level to the height of the tallest item packed in this shelf. We iterate with this current level as our new base.

In general, NFDH turns out to be an inefficient algorithm to pack hypercuboids, even if they are very small in volume when compared to the bin. But when we restrict ourselves to small hypercubes, NFDH again turns out to be a very good algorithm. This is formalized in the following lemma which is a restatement from [BCKS06].

Lemma 2.6. *Let S be a set of d-D hypercubes each with side length at most δ. Consider a d-D hypercuboidal region $\mathcal{R}$ with side lengths $r_1, r_2, \ldots, r_d$ (with each r_i at most 1). Suppose we try to pack S into $\mathcal{R}$ using NFDH. Then we either pack all the items, or the wasted space in $\mathcal{R}$ is at most $\delta(r_1 + r_2 + \cdots + r_d)$.*

Chapter 3

Near-optimal Algorithms for Stochastic Online Bin Packing

Recall the Bin Packing (BP) problem. In BP, we are given a set of items $I := (x_1, x_2, \ldots, x_n)$ with their associated weights (also called sizes) x_i's in $(0, 1]$ and the goal is to partition them into the minimum number of sets (bins) such that the total weight of each set is at most 1.

In this chapter, we will study online BP under two stochastic models, namely the i.i.d. model, and the random-order model and we will prove Theorems 1.1 to 1.3.

3.1 Related Work

Being one of the cornerstones of approximation and online algorithms, BP has been very well-studied.

Best-Fit (BF), First-Fit (FF), and Next-Fit (NF) are the three most commonly used simple algorithms for BP. Given x_i as the present item to be packed, they work as follows:

- NF: Pack x_i into the *most recently opened* bin; open a new bin if necessary.
- BF: Pack x_i into the *fullest possible* bin; open a new bin if necessary.
- FF: Pack x_i into the *first possible* bin; open a new bin if necessary.

The formal definitions of NF, BF have been given in Section 2.4.1. We also proved in Lemma 2.4 that the asymptotic approximation ratio of NF is 2. Johnson et al. [JDU+74] analyzed several heuristics like BF, FF, Best-Fit-Decreasing (BFD), First-Fit-Decreasing (FFD) and showed their asymptotic approximation guarantees to be $17/10, 17/10, 11/9, 11/9$, respectively. Bekesi et al. [BGK00] gave an $O(n)$ time $5/4$-asymptotic approximation algorithm. Another $O(n \log n)$ time algorithm is Modified-First-Fit-Decreasing (MFFD) [JG85] which attains an AAR of $71/60 \approx 1.1834$. Vega and Lueker [dlVL81] gave an asymptotic fully polynomial-time approximation scheme (AFPTAS) for BP: For any $1/2 > \varepsilon > 0$, it returns a solution with at most $(1+\varepsilon)\mathrm{Opt}(I) + O(1)$ [1] bins in time $C_\varepsilon + Cn \log 1/\varepsilon$, where C is an absolute constant and C_ε depends only on ε. Karmarkar and Karp [KK82] gave an algorithm that returns a solution using $\mathrm{Opt}(I) + O(\log^2 \mathrm{Opt}(I))$ bins. The present best approximation is due to Hoberg and Rothvoss [HR17a] which returns a solution using $\mathrm{Opt}(I) + O(\log \mathrm{Opt}(I))$ bins.

3-Partition problem is a notoriously hard special case of BP where all item sizes are larger than $1/4$. Eisenbrand et al. [EPR11] mentioned that "much of the hardness of bin packing seems to appear already in the special case of 3-Partition when all item sizes are in $(1/4, 1/2]$". This problem has deep connections with Beck's conjecture in discrepancy theory [Spe94, NNN12]. In fact, Rothvoss [HR17a] conjectured that these 3-Partition instances are indeed the hardest instances for bin packing and the additive integrality gap of the bin packing configuration LP for these 3-Parition instances is already $\Theta(\log n)$.

In online BP, items appear one by one and are required to be packed immediately and irrevocably. Lee and Lee [LL85b] presented the Harmonic algorithm with competitive ratio $T_\infty \approx 1.691$, which is optimal for $O(1)$ space algorithms. For general online BP, the present best upper and lower bounds for the competitive ratio (CR) are 1.57829 [BBD+18] and 1.54278 [BBD+19], respectively.

In the *i.i.d. model* (see Section 1.1.3.1), bin packing has mainly been studied under continuous uniform (denoted by $U[a,b], 0 \le a < b \le 1$, where item sizes are chosen uniformly from $[a,b]$) or discrete uniform distributions (denoted by $U\{j,k\}, 1 \le j \le k$, where item sizes are chosen uniformly from $\{1/k, 2/k, \ldots, j/k\}$). For $U[0,1]$, Coffman et al. [CJSHY80] showed that NF has an expected competitive ratio (ECR) of $4/3$ and Lee and Lee [LL87] showed that the Harmonic algorithm has an ECR of $\pi^2/3 - 2 \approx 1.2899$. Interestingly, Bentley et al. [BJL+84] showed that the ECR of FF as well as BF converges to 1 for $U[0,1]$. It was later shown that the expected wasted space (i.e., the number of needed bins minus the total size of items) is $\Theta(n^{2/3})$ for First-Fit [Sho86, CJJSW97] and $\Theta(\sqrt{n}\log^{3/4} n)$ for Best-Fit [Sho86, LS89]. Rhee and Talagrand [RT93b] exhibited an algorithm that w.h.p. achieves a packing in $\mathrm{Opt} + O(\sqrt{n}\log^{3/4} n)$ bins for any distribution F on $(0,1]$. However, note that their competitive ratio can be quite bad when $\mathrm{Opt} \ll n$. A distribution F is said to be *perfectly packable* if the expected wasted space in the optimal solution is $o(n)$ (i.e., nearly all bins in an optimal packing are almost fully packed). Csirik et al. [CJK+06] studied the Sum-of-Squares (SS) algorithm and showed that for any perfectly packable distribution, the expected wasted space is $O(\sqrt{n})$. However, for

[1] As mentioned in Section 2.1, in bin packing and related problems, the accuracy parameter ε is assumed to be a constant. Here, the term $O(1)$ hides some constants depending on ε.

distributions that are not perfectly packable, the SS algorithm has an ECR of at most 3 and can have an ECR of 3/2 in the worst-case [CJK+06]. For any discrete distribution, they gave an algorithm with an ECR of 1 that runs in pseudo-polynomial time in expectation. Gupta et al. [GS20] also obtained similar $o(n)$ expected wasted space guarantee by using an algorithm inspired by the interior-point (primal-dual) solution of the bin packing LP. However, it remains an open problem to obtain a polynomial-time $(1+\varepsilon)$-competitive algorithm for online bin packing under the i.i.d. model for arbitrary general distributions. In fact, the present best polynomial-time algorithm for bin packing under the i.i.d. model is BF which has an ECR of at most 3/2. However, Albers et al. [AKL21a] showed that BF has an ECR ≥ 1.1 even for a simple distribution: when each item has size 1/4 with probability 3/5 and size 1/3 with probability 2/5.

We also study bin packing under *random-order model* (see Section 1.1.3.2). Recall that the performance of an algorithm $\mathcal{A}$ is measured using random-order ratio ($\mathrm{RR}^{\infty}_{\mathcal{A}}$) in this model. Random-order model generalizes the i.i.d. model [AKL21a], thus the lower bounds in the random-order model can be obtained from the i.i.d. model. Kenyon in her seminal paper [Ken96] studied Best-Fit under random-order and showed that $1.08 \leq \mathrm{RR}^{\infty}_{\mathrm{BF}} \leq 3/2$. She conjectured that $\mathrm{RR}^{\infty}_{\mathrm{BF}} \approx 1.15$. The conjecture, if true, raises the possibility of a better alternate practical offline algorithm: first shuffle the items randomly, then apply Best-Fit. This then beats the AAR of 71/60 of the present best practical algorithm MFFD. The conjecture has received a lot of attention in the past two decades and yet, no other polynomial-time algorithm is known with a better random-order ratio than BF. Coffman et al. [JCRZ08] showed that $\mathrm{RR}^{\infty}_{\mathrm{NF}} = 2$. Fischer and Röglin [FR18] achieved analogous results for Worst-Fit [Joh74] and Smart-Next-Fit [Ram89]. Recently, Fischer [Car19] presented an exponential-time algorithm, claiming a random-order ratio of $(1+\varepsilon)$.

Monotonicity is a natural property of BP algorithms, which holds if the algorithm never uses fewer bins to pack $\hat{I}$ when compared I, where $\hat{I}$ is obtained from I by increasing the item sizes. Murgolo [Mur88] showed that while NF is monotone, BF and FF are not.

Several other problems have been studied under the i.i.d. model and the random-order model [DGV08, GKNS21, FMMM09, GKR12, Fer89, AKL21b, FR16, MY11, GS20].

3.2 Overview of Results and Techniques

Bin packing under the i.i.d. model: We achieve near-optimal performance guarantee for the bin packing problem under the i.i.d. model, thus settling the problem. For any arbitrary unknown distribution $\mathcal{D}$ on $(0,1]$, we give a meta-algorithm (see Algorithm 1) that takes an α-asymptotic approximation algorithm as input and provides a polynomial-time $(\alpha+\varepsilon)$-

competitive algorithm. Note that both the distribution $\mathcal{D}$ as well as the number of items n are unknown in this case.

Theorem 3.1. *Let $\varepsilon \in (0,1)$ be a constant parameter. For online bin packing under the i.i.d. model, where n items are sampled from an unknown distribution $\mathcal{D}$, given an offline algorithm $\mathcal{A}_\alpha$ with an AAR of α and runtime $\beta(n)$, there exists a meta-algorithm (Algorithm 1) which returns a solution with an ECR of $(\alpha+\varepsilon)$ and runtime $O(\beta(n))$.* [2]

Using an AFPTAS for bin packing (e.g. [dlVL81]) as $\mathcal{A}_\alpha$, we obtain the following corollary.

Corollary 3.1.1. *Using an AFPTAS for bin packing as $\mathcal{A}_\alpha$ in Theorem 3.1, we obtain an algorithm for online bin packing under the i.i.d. model with an ECR of $(1+\varepsilon)$ for any $\varepsilon \in (0, 1/2)$.*

Most algorithms for bin packing under the i.i.d. model are based on the following idea. Consider a sequence of $2k$ items where each item is independently drawn from an unknown distribution $\mathcal{D}$, and let $\mathcal{A}$ be a packing algorithm. Pack the first k items using $\mathcal{A}$; denote the packing by $\mathcal{P}'$. Similarly, let $\mathcal{P}''$ be the packing of the next k items using $\mathcal{A}$. Since each item is drawn independently from $\mathcal{D}$, both $\mathcal{P}'$ and $\mathcal{P}''$ have the same properties in expectation; in particular, the expected number of bins used in $\mathcal{P}'$ and $\mathcal{P}''$ are the same. Thus, intuitively, we want to use the packing $\mathcal{P}'$ as a proxy for the packing $\mathcal{P}''$. However, there are two problems. First, we do not know n, which means that there is no way to know what a good sample size is. Second, we need to show the stronger statement that w.h.p. $\mathcal{P}' \approx \mathcal{P}''$. Note that the items in $\mathcal{P}'$ and $\mathcal{P}''$ are expected to be similar, but they may not be the same. So, it is not clear which item in $\mathcal{P}'$ is to be used as a proxy for a newly arrived item in the second half. Due to the online nature, erroneous choice of proxy items can be quite costly. Different algorithms handle this problem in different ways. Some algorithms exploit the properties of particular distributions, some use exponential or pseudo-polynomial time, etc.

Rhee and Talagrand [RT88, RT93b] used *upright matching* to decide which item can be considered as a proxy for a newly arrived item.

They consider the model packing $\mathcal{P}_k$ of the first k items (let's call these the proxy items) using an offline algorithm. With the arrival of each of the next k items, they take a proxy item at random and pack it according to the model packing. Then, they try to fit in the real item using upright matching. They repeat this process until the last item is packed. However, they could only show a guarantee of $\text{Opt} + O(\sqrt{n}\log^{3/4} n)$. The main drawback of [RT93b] is that their ECR can be quite bad if $\text{Opt} \ll n$ (say, $\text{Opt} = \sqrt{n}$). One of the reasons for this drawback

[2] As mentioned in an earlier footnote, the $O(\cdot)$ notation hides some constants depending on ε here.

is that they don't distinguish between small and large items; when there are too many small items, the ECR blows up.

Using a similar approach, Fischer [Car19] obtained a $(1+\varepsilon)$-competitive randomized algorithm for the random-order model, but it takes exponential time, and the analysis is quite complicated. The exponential time was crucial in finding the optimal packing which was then used as a good proxy packing. However, prior to our work, no polynomial-time algorithm existed which achieves a $(1+\varepsilon)$ competitive ratio.

To circumvent these issues, we treat large and small items separately. However, a straightforward adaptation faces several technical obstacles. Thus our analysis required intricate applications of concentration inequalities and sophisticated use of upright matching. First, we consider the semi-random case when we know n. Our algorithm works in stages. For a small constant $\delta \in (0,1]$, the first stage contains only $\delta^2 n$ items. These items give us an estimate of the distribution. If the packing does not contain too many large items, we show that the simple Next-Fit algorithm suffices for the entire input. Otherwise, we use a proxy packing of the set of first $\delta^2 n$ items to pack the next $\delta^2 n$ items. In the process, the small and large items are packed in a different manner. The third set of $\delta^2 n$ number of items are packed using the proxy packing of the second set of $\delta^2 n$ number of items. This process continues until all the items arrive.

Finally, we get rid of the assumption that we know n by first guessing the value of n and then refining our guess if it is incorrect. First, we guess the value of n to be a constant n_0. If it is incorrect, we increase our guess by multiplying n_0 with a small factor greater than 1. We continue this process of improving our guess until all the items arrive.

Our algorithm is simple, polynomial-time (in fact, $O(n)$ time), and achieves essentially the best possible competitive ratio. It is relatively simpler to analyze when compared to Fischer's algorithm [Car19]. Also, unlike the algorithms of Rhee and Talagrand [RT93b] as well as Fischer [Car19], our algorithm is deterministic. This is because, unlike their algorithms, instead of taking proxy items at random, we pack all the proxy items before the start of a stage and try to fit in the real items as they come. This makes our algorithm deterministic. Our algorithm is explained in detail in Section 3.3.1. The nature of the meta-algorithm provides flexibility and ease of application. See Table 3.1 for the performance guarantees obtained using different offline algorithms.

$\mathcal{A}_\alpha$	**Time Complexity**	**Expected Competitive Ratio**
AFPTAS [dlVL81]	$O(C_\varepsilon + Cn \log 1/\varepsilon)$	$(1+\varepsilon)$
Modified-First-Fit-Decreasing [JG85]	$O(n \log n)$	$(71/60+\varepsilon)$
Best-Fit-Decreasing [Joh73]	$O(n \log n)$	$(11/9+\varepsilon)$
First-Fit-Decreasing [Joh73]	$O(n \log n)$	$(11/9+\varepsilon)$
Next-Fit-Decreasing [BC81]	$O(n \log n)$	$(T_\infty+\varepsilon)$
Harmonic [LL85b]	$O(n)$	$(T_\infty+\varepsilon)$
Next-Fit	$O(n)$	$(2+\varepsilon)$

Table 3.1: Analysis of Algorithm 1 depending on $\mathcal{A}_\alpha$. In the first row, C is an absolute constant and C_ε is a constant that depends on ε.

See Section 3.3 for the details of the proof and the description of our algorithm. In fact, our algorithm can easily be generalized to d-dimensional online vector packing [BEK16], a multidimensional generalization of bin packing. See Section 3.5 for a $d(\alpha+\varepsilon)$ competitive algorithm for d-dimensional online vector packing where the i^{th} item X_i can be seen as a tuple $\left(X_i^{(1)}, X_i^{(2)}, \ldots, X_i^{(d)}\right)$ where each $X_i^{(j)}$ is independently sampled from an unknown distribution $\mathcal{D}^{(j)}$.

Bin packing under the random-order model: Next, we study BP under the random-order model. Recently, Albers et al. [AKL21a] showed that BF is monotone if all the item sizes are greater than $1/3$. Using this result, they showed that in this special case, BF has a random-order ratio of at most $5/4$. We show that, somewhat surprisingly, in this case, BF actually has a random-order ratio of 1 (see Section 3.4.1 for the detailed proof).

Theorem 3.2. *For online bin packing under the random-order model, Best-Fit achieves a random-order ratio of* 1 *when all the item sizes are in* $(1/3, 1]$.

Next, we study the 3-partition problem, a special case of bin packing when all the item sizes are in $(1/4, 1/2]$. This is known to be an extremely hard case [HR17a]. Albers et al. [AKL21a] mentioned that "it is sufficient to have one item in $(1/4, 1/3]$ to force Best-Fit into anomalous behavior." E.g., BF is non-monotone in the presence of items of size less than $1/3$. Thus the techniques of [AKL21a] do not extend to the 3-Partition problem. We break the barrier of $3/2$ in this special case, by showing that BF attains a random-order ratio of ≈ 1.4941.

Theorem 3.3. *For online bin packing under the random-order model, Best-Fit achieves a random-order ratio of at most* ≈ 1.4941 *when all the item sizes are in* $(1/4, 1/2]$.

We prove Theorem 3.3 in Section 3.4.2. As 3-partition instances are believed to be the hardest instances for bin packing, our result gives a strong indication that the random-order ratio of BF might be strictly less than $3/2$.

3.3 Online Bin Packing Problem under the i.i.d. Model

In this section, we provide the meta algorithm as described in Theorem 3.1. For the ease of presentation, we split this into two parts. In Section 3.3.1, we assume a semi-random model, i.e., we assume that the number of items n is known beforehand and design an algorithm. Later, in Section 3.3.2, we get rid of this assumption.

Let the underlying distribution be $\mathcal{D}$. Without loss of generality, we assume that the support set of $\mathcal{D}$ is a subset of $(0, 1]$. For any set of items J, we define $\mathcal{W}(J)$ as the sum of weights of all the items in J. For any $k \in \mathbb{N}_+$, we denote the set $\{1, 2, \ldots, k\}$ by $[k]$. Let $\varepsilon \in (0, 1)$ be a constant parameter and let $0 < \delta < \varepsilon/8$ be a constant such that $1/\delta$ is an integer. Let $\mathcal{A}_\alpha$ be an offline algorithm for bin packing with an AAR of $\alpha > 1$ and let Opt denote the optimal algorithm. For any $i \in [n]$, we call x_i to be a *large* item if $x_i \geq \delta$ and a *small* item otherwise. Let I_ℓ and I_s denote the set of large and small items in I, respectively.

3.3.1 Algorithm Assuming that the Value of n is Known

We now describe our algorithm which assumes the knowledge of the number of items. For simplicity, in this section we assume that $1/\delta^2$ is a divisor of n; we will anyway get rid of the assumption on the knowledge of n in the next subsection. First, we give a high level idea of the algorithm. We divide the entire input into stages as follows: we partition the input set I into $m := 1/\delta^2$ stages $T_0, T_1, \ldots, T_{m-1}$. The zeroth stage T_0, called the *sampling stage*, contains the first $\delta^2 n$ items, i.e., $x_1, x_2, \ldots, x_{\delta^2 n}$. For $j \in [m-1]$, T_j contains the items with index starting from $j\delta^2 n + 1$ till $(j+1)\delta^2 n$. In essence, T_0 contains the first $\delta^2 n$ items, T_1 contains the next $\delta^2 n$ items, T_2 contains the next $\delta^2 n$ items, and so on. Note that the number of stages m is a constant. In any stage T_j, we denote the set of large items and small items by L_j and S_j, respectively. Note that for any $j \in [m-1]$, $|T_j| = |T_{j-1}|$ and since all the items are sampled independently from the same distribution, we know that with high probability, the optimal solutions of T_j and T_{j-1} are quite similar. Since T_{j-1} would have arrived before T_j, we compute an almost optimal packing (resp. a packing within α factor of the optimal packing) of T_{j-1} (in an offline manner) and use it as a blueprint to pack T_j almost optimally (resp. within α factor of the optimum).

The algorithm is as follows: first, we pack T_0, the sampling stage using Next-Fit. The sampling stage contains only a small but a constant fraction of the entire input set; hence it uses only a few number of bins when compared to the final packing but at the same time provides a good estimate of the underlying distribution. If the number of large items in the sampling stage is at most $\delta^3\mathcal{W}(T_0)$, then we continue using Next-Fit for the rest of the entire input too. Intuitively, NF performs well in this case as most of the items are small. Thus, from now on, let us assume otherwise. Now assume that we are at an intermediate point where T_{j-1} has arrived and T_j is about to arrive ($j \geq 1$). We create D_j, the set of *proxy* items, which is just a copy of T_{j-1}. We pack D_j using $\mathcal{A}_\alpha$. Let this packing be denoted by $\mathcal{P}_j$. Let $B_j^{(k)}$ denote the k^{th} bin in the packing $\mathcal{P}_j$. We iterate over k and remove all the small items in the bin $B_j^{(k)}$ and create a slot in the free space of $B_j^{(k)}$.

We call this slot to be an S-*slot*. When an item $x_i \in T_j$ arrives, we check if x_i is small or large

- If x_i is small, we pack it in one of the S-slots greedily, using Next-Fit. If it doesn't fit in any of the S-slots, then we create a new bin with only one S-slot spanning the entire bin (so, this bin will only be used to pack small items), and pack it there.

- If x_i is large, we remove the smallest proxy item with a size more than x_i in the packing $\mathcal{P}_j$ and pack it there. If no such proxy item exists, we open a new bin, pack x_i in there and close it, meaning that it will not be used to pack any further items.

After T_j is packed completely, we just discard the proxy items in the packing that haven't been replaced and move to the next stage. For more formal details and pseudocode for the algorithm, please refer to Algorithm 1.

We will proceed to analyze the algorithm. But first, we will discuss stochastic upright matching and a standard result on it. Using a standard probabilistic concentration inequality, we will formulate a few lemmas which are going to be very important for the analysis of the algorithm.

Stochastic Upright Matching. Rhee and Talagrand [RT93c] studied the stochastic upright matching problem in the context of analysis of bin packing algorithms. Consider a set $P = \{(x_i, y_i)\}_{i\in[2n]}$ of $2n$ points where each x_i is either $+1$ or -1 with equal probability and $y_1, y_2, \ldots, y_{2n}$ are sampled according to an i.i.d. distribution. We can define a bipartite graph $\mathcal{G}$ as follows: the vertex set is P, viewed as a set of points in $\mathbb{R} \times \mathbb{R}$. Two points $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ share an edge iff $x_1 = 1, x_2 = -1$ and $y_1 \geq y_2$.

The objective of the problem is to find a maximum matching in $\mathcal{G}$ or, in other words, minimize the number of unmatched points which we denote by $U(P)$.

We denote the stochastic upright matching problem by $\mathcal{M}$. The following lemma shows that w.h.p., the number of unmatched points is $O(\sqrt{n}(\log n)^{3/4})$. The proof of the lemma follows

Algorithm 1 `Alg`$(x_1, x_2, \dots, x_n)$: A nearly optimal algorithm for online bin packing assuming that the number of items n is known before-hand

```
1: Input: $I_n(\mathcal{D}) = \{x_1, x_2, ..., x_n\}$.
2: $m := \frac{1}{\delta^2}$                                   ▷ Number of stages
3: for $j$ in $\{0, 1, \dots, m-1\}$ do
4:     $T_j = \{x_{j\delta^2 n+1}, x_{j\delta^2 n+2}, \dots, x_{(j+1)\delta^2 n}\}$        ▷ The $j^{\text{th}}$ stage
5: end for
6: Pack the sampling stage $T_0$ using Next-Fit.
7: if $|L_0| \le \delta^3 \mathcal{W}(T_0)$ then     ▷ Very few large items in the sampling stage
8:     Use Next-Fit for all the remaining stages.
9: else
10:    for $j$= 1 to $m-1$ do
11:        $D_j \leftarrow T_{j-1}; L(D_j) \leftarrow$ set of large items in $D_j$.
12:        Pack $D_j$ using $\mathcal{A}_\alpha$.
13:        Let the packing be denoted by $\mathcal{P}_j$.          ▷ Packing of proxy items
14:        $\mathcal{S}_j \leftarrow \phi$.                           ▷ the set of $S$-slots
15:        for bin $B$ in $\mathcal{P}_j$ do
16:            Remove the small items in $B$.
17:            Create an $S$-slot $H$ of size equal to (1−weight of all the large items in $B$).
18:            $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup H$.
19:        end for
20:        for $x_i \in T_j$ do
21:            if $x_i$ is large then
22:                if $\exists d \in L(D_j)$ such that $d \ge x_i$ then
23:                    Find smallest such $d$.
24:                    $L(D_j) \leftarrow L(D_j) \setminus \{d\}$.
25:                    Pack $x_i$ in place of $d$ in the packing $\mathcal{P}_j$.
26:                else
27:                    Open a new bin and pack $x_i$ and close the bin.
28:                end if
29:            else
30:                Try packing $x_i$ in $\mathcal{S}_j$ using Next-Fit.
31:                if $x_i$ couldn't be packed then
32:                    Open a new bin $B'$ with a single $S$-slot of unit capacity.
33:                    $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup B'$.
34:                    Pack $x_i$ in $B'$.
35:                end if
36:            end if
37:        end for
38:    end for
39: end if
```

from Lemma 3.1 in [RT93c].

Lemma 3.4. *[RT93c] Let P be an instance for $\mathcal{M}$. Then there exist constants $a, C, K > 0$ such that,*

$$\mathbb{P}\left[U(P) \geq K\sqrt{n}(\log n)^{3/4}\right] \leq C\exp\left(-a(\log n)^{3/2}\right)$$

Concentration Inequalities. Now we state the concentration inequalities. The following observation will be used extensively.

Observation 3.5. *For any instance I, $\mathcal{W}(I) \leq \mathrm{Opt}(I) \leq 2\mathcal{W}(I)$.*

Lemma 3.6 (Bernstein's Inequality)**.** *Let $X_1, X_2, \ldots, X_n$ be independent random variables such that each $X_i \in [0,1]$. Then, for any $\lambda > 0$, the following inequality holds.*

$$\mathbb{P}\left[\left|\sum_{i=1}^{n} X_i - \sum_{i=1}^{n}\mathbb{E}\left[X_i\right]\right| \geq \lambda\right] \leq 2\exp\left(-\frac{\lambda^2}{2\left(\sum_{i=1}^{n}\mathbb{E}\left[X_i\right] + \lambda/3\right)}\right).$$

The following lemma is a direct implication of the results of [RT93b, Rhe94]. It intuitively states that the optimal solution for a part of the input is almost a linear function of the length of the part. This makes sense because each item comes from the same distribution.

Lemma 3.7. *For any $t \in [n]$, let $I(1,t)$ denote the first t items of the input set I. Then there exist constants $K, a > 0$ such that,*

$$\mathbb{P}\left[\mathrm{Opt}(I(1,t)) \geq \frac{t}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + K\sqrt{n}(\log n)^{3/4}\right] \leq \exp\left(-a(\log n)^{3/2}\right)$$

Proof. The following claim follows directly from Theorem 2.1 in [RT93b].

Claim 3.8. *Let I be a list of n items sampled independently from a distribution and let $I(1,t)$ denote the first t items. Then there exist constants $K_1, a_1 > 0$ such that*

$$\mathbb{P}\left[\mathrm{Opt}(I(1,t)) \geq \frac{t}{n}\mathrm{Opt}(I) + K_1\sqrt{n}(\log n)^{3/4}\right] \leq \exp\left(-a_1(\log n)^{3/2}\right)$$

The next claim is a direct implication of the main result of [Rhe94].

Claim 3.9. *Let I be a list of n items sampled independently from a distribution. Then there exist constants $K_2, a_2 > 0$ such that*

$$\mathbb{P}\left[\mathbb{E}\left[\mathrm{Opt}(I)\right] \geq \mathrm{Opt}(I) + K_2\sqrt{n}(\log n)^{3/4}\right] \leq \exp\left(-a_2(\log n)^{3/2}\right)$$

Combining both the claims, we obtain that there exist constants $K, a > 0$ such that

$$\mathbb{P}\left[\mathrm{Opt}(I(1,t)) \geq \frac{t}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + K\sqrt{n}(\log n)^{3/4}\right] \leq \exp\left(-a(\log n)^{3/2}\right)$$

□

The next lemma states that the weight of the items of a part of the input is almost a linear function of the length of the part.

Lemma 3.10. *For an input set I of n items drawn from a distribution independently, for any arbitrary but fixed* [3] *set $J \subseteq I$ we have,*

$$\mathbb{P}\left[\left|\mathcal{W}(J) - \frac{|J|}{n}\mathbb{E}\left[\mathcal{W}(I)\right]\right| \geq \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\right] \leq 2\exp\left(-\frac{1}{3}\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$$

Proof. We can assume that $\mathcal{W}(I)$ goes to infinity since we know that $\mathrm{Opt}(I) \leq 2\mathcal{W}(I) + 1$. Let $K := \mathbb{E}\left[\mathcal{W}(I)\right]$ and $\mathcal{W}(x)$ be the weight of item x. Using Bernstein's inequality (Lemma 3.6),

$$\begin{aligned}
&\mathbb{P}\left[\left|\sum_{x\in J}\mathcal{W}(x) - \sum_{x\in J}\mathbb{E}\left[\mathcal{W}(x)\right]\right| \geq K^{2/3}\right] \\
&\leq 2\exp\left(-\frac{K^{4/3}}{2\left(\sum_{x\in J}\mathbb{E}\left[\mathcal{W}(x)\right] + K^{2/3}/3\right)}\right) \\
&\leq 2\exp\left(-\frac{K^{4/3}}{2\left(K + K^{2/3}/3\right)}\right) \\
&\leq 2\exp\left(-\frac{K^{4/3}}{2\left(K + K/3\right)}\right) && \text{(since } K \text{ goes to infinity, } K^{2/3} \leq K\text{)} \\
&\leq 2\exp\left(-\frac{1}{3}K^{1/3}\right).
\end{aligned}$$

Since $\sum_{x\in J}\mathcal{W}(x) = \mathcal{W}(J)$ and since $\mathbb{E}\left[\mathcal{W}(J)\right] = \frac{|J|}{n}\mathbb{E}\left[\mathcal{W}(I)\right]$, the lemma follows. □

The lemma below states that the number of large items in a part of the input is almost a linear function of the length of the part.

Lemma 3.11. *Let I be an input set of n items drawn from a distribution independently and let J be any arbitrary but fixed subset of I. Suppose J_ℓ (resp. I_ℓ) denote the set of large items*

[3]the indices of J must be fixed beforehand

in J (resp. I). Then we have,

$$\mathbb{P}\left[\left||J_\ell| - \frac{|J|}{n}\mathbb{E}\left[|I_\ell|\right]\right| \geq \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\right] \leq 2\exp\left(-\frac{\delta}{3}\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$$

Proof. We can assume that $\mathcal{W}(I)$ goes to infinity. Let $K := \mathbb{E}\left[\mathcal{W}(I)\right]$. For any item x, let L_x be the indicator random variable which denotes if the item x is a large item or not. Using Bernstein's inequality (Lemma 3.6),

$$\begin{aligned}
&\mathbb{P}\left[\left|\sum_{x\in J} L_x - \sum_{x\in J}\mathbb{E}\left[L_x\right]\right| \geq K^{2/3}\right] \\
&\leq 2\exp\left(-\frac{K^{4/3}}{2\left(\sum_{x\in J}\mathbb{E}\left[L_x\right] + K^{2/3}/3\right)}\right) \\
&= 2\exp\left(-\frac{K^{4/3}}{2\left(\mathbb{E}\left[|J_\ell|\right] + K^{2/3}/3\right)}\right) && \left(\text{since } \textstyle\sum_{x\in J} L_x = |J_\ell|\right) \\
&\leq 2\exp\left(-\frac{K^{4/3}}{2\left(K/\delta + K^{2/3}/3\right)}\right) && \left(\text{since } K = \mathbb{E}\left[\mathcal{W}(I)\right] \geq \delta\mathbb{E}\left[|J_\ell|\right]\right) \\
&\leq 2\exp\left(-\frac{\delta}{3}K^{1/3}\right).
\end{aligned}$$

Since $\sum_{x\in J} L_x = |J_\ell|$ and since $\mathbb{E}\left[|J_\ell|\right] = \frac{|J|}{n}\mathbb{E}\left[|I_\ell|\right]$, the lemma follows. $\square$

Lemma 3.12 further tries to improve on Lemma 3.7 by bounding the lower order terms in terms of $\mathbb{E}\left[\mathrm{Opt}(I)\right]$ instead of n while losing only a small factor of $1+2\delta$. This is crucial since we need to bound the number of additional bins used by our algorithm in terms of $\mathbb{E}\left[\mathrm{Opt}(I)\right]$ and not in terms of n.

Lemma 3.12. *For any $t \in \{1, 2, \ldots, n\}$, Suppose $I(1,t)$ denote the first t items of the input set I. Then there exist constants $C, a > 0$ such that with probability at least $1 - \exp\left(-a\left(\log\mathbb{E}\left[\mathrm{Opt}(I)\right]\right)^{1/3}\right)$, we have*

$$\mathrm{Opt}(I(1,t)) \leq (1+2\delta)\frac{t}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + C\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3}$$

Proof. Let's denote the set of large items in $I(1,t)$ by $I_\ell(1,t)$ and denote the set of small items by $I_s(1,t)$. Consider any optimal packing of $I_\ell(1,t)$. Start packing $I_s(1,t)$ in the spaces left in the bins greedily using Next-Fit while opening new bins whenever necessary. This gives us a valid packing of $I(1,t)$.

If we don't open any new bins to pack $I_s(1,t)$, then by Lemma 3.7 for some constants $C_1, C_2 > 0$,

$$\begin{aligned}
\mathrm{Opt}(I(1,t)) &\le \mathrm{Opt}\left(I_\ell(1,t)\right) \\
&\le \frac{|I_\ell(1,t)|}{|I_\ell|}\mathbb{E}\left[\mathrm{Opt}(I_\ell)\right] + C_1\sqrt{\mathbb{E}\left[|I_\ell|\right]}(\log \mathbb{E}\left[|I_\ell|\right])^{3/4} && \text{(using Lemma 3.7)} \\
&\le \frac{|I_\ell(1,t)|}{|I_\ell|}\mathbb{E}\left[\mathrm{Opt}(I_\ell)\right] + \mathbb{E}\left[|I_\ell|\right]^{2/3} \\
&\le \frac{\frac{t}{n}\left|\mathbb{E}\left[|I_\ell|\right]\right| + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}}{|I_\ell|}\mathbb{E}\left[\mathrm{Opt}(I_\ell)\right] + \mathbb{E}\left[|I_\ell|\right]^{2/3} && \text{(using Lemma 3.11)} \\
&\le \frac{\frac{t}{n}\left(|I_\ell| + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\right) + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}}{|I_\ell|}\mathbb{E}\left[\mathrm{Opt}(I_\ell)\right] + \mathbb{E}\left[|I_\ell|\right]^{2/3} \\
& && \text{(using Lemma 3.11)} \\
&\le \frac{t}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_1\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} && \text{(using } \mathcal{W}(I) \ge \delta\,|I_\ell|\text{)}
\end{aligned}$$

with probability at least $1 - \exp\left(-a(\log \mathbb{E}\left[\mathrm{Opt}(I)\right])^{1/3}\right)$ for some constants $a, C_1 > 0$.

On the other hand, if we open new bins while packing $I_s(1,t)$, then after the final packing, every bin (except possibly one) is filled to a level of at least $1-\delta$. Hence, in this case,

$$\begin{aligned}
\mathrm{Opt}(I(1,t)) &\le \frac{1}{1-\delta}\mathcal{W}(I(1,t)) + 1 \\
&\le (1+2\delta)\frac{t}{n}\mathbb{E}\left[\mathcal{W}(I)\right] + (1+2\delta)\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} + 1 \\
&\le (1+2\delta)\frac{t}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_2\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} \\
&\qquad \text{(using } \mathrm{Opt}(I) \ge \mathcal{W}(I) \text{ and } (1-\delta)(1+2\delta) \ge 1 \text{ for } \delta < 1/2\text{)}
\end{aligned}$$

with probability at least $1 - \exp\left(-b\mathbb{E}\left[\mathrm{Opt}(I)\right]^{1/3}\right)$ for some constants $b, C_2 > 0$. This completes the proof. □

With these helpful lemmas, we now proceed to analyze the algorithm. We split the analysis into the following two cases: when $|L_0| \le \delta^3 \cdot \mathcal{W}(T_0)$ and when $|L_0| > \delta^3 \cdot \mathcal{W}(T_0)$.

3.3.1.1 Analysis of Case 1: $|L_0| \le \delta^3 \cdot \mathcal{W}(T_0)$

Recall that in this case, we just continue with Next-Fit for all the remaining items. To bound the Next-Fit solution, we first consider the number of bins that contain at least one large item. For this, we bound the value of $|I_\ell|$. Then we consider the bins that contain only small items and bound this value in terms of weight of all items $\mathcal{W}(I)$.

Claim 3.13. *Let $K := \mathbb{E}\left[\mathrm{Opt}(I)\right]$. For some positive constants C_1, C_2, a, we have that*

$$\mathbb{P}\left[|I_\ell| \leq \delta \cdot \mathcal{W}(T_0) + C_1 K^{2/3}\right] \geq 1 - C_2 \exp\left(-aK^{1/3}\right)$$

Proof. As the sampling stage contains $\delta^2 n$ items, $\mathbb{E}\left[|L_0|\right] = \delta^2 \mathbb{E}\left[|I_\ell|\right]$. From Lemma 3.11, we have

$$\mathbb{P}\left[|L_0| \leq \delta^2 \mathbb{E}\left[|I_\ell|\right] - \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\right] \leq 2\exp\left(-(\delta/3)\cdot(\mathbb{E}\left[\mathcal{W}(I)\right])^{1/3}\right), \text{ and}$$
$$\mathbb{P}\left[|I_\ell| \geq \mathbb{E}\left[|I_\ell|\right] + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\right] \leq 2\exp\left(-(\delta/3)\cdot\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$$

From the above inequalities we have,

$$|I_\ell| \leq \frac{1}{\delta^2}|L_0| + \left(1 + \frac{1}{\delta^2}\right)\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$$

with probability at least $1 - 4\exp\left(-\frac{\delta}{3}\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$. We can use the inequalities $\mathbb{E}\left[\mathrm{Opt}(I)\right] \geq \mathbb{E}\left[\mathcal{W}(I)\right]$ and $|L_0| \leq \delta^3 \cdot \mathcal{W}(T_0)$ to conclude the proof of this claim. □

Now we bound the number of bins that are closed by small items. Note that Next-Fit fills each such bin up to a capacity at least $(1-\delta)$. So, the number of such bins is at most $\frac{1}{1-\delta}\mathcal{W}(I)$ when all items are packed by Next-Fit. Also, there can be at most one bin that can be open. Thus combining all these results, (and using inequality $\frac{1}{1-\delta} \leq 1 + 2\delta$ for $\delta < \frac{1}{2}$) with high probability,

$$\mathrm{NF}(I) \leq |I_\ell| + (1+2\delta)\mathcal{W}(I) + 1 \leq \delta \cdot \mathcal{W}(T_0) + (1+2\delta)\mathcal{W}(I) + K\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3}$$

for some constant K.

Using Lemma 3.10, we get that with high probability, $\mathcal{W}(I) \leq \mathbb{E}\left[\mathcal{W}(I)\right] + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$. Using the facts $\mathcal{W}(I) \geq \mathcal{W}(T_0)$ and $\mathbb{E}\left[\mathrm{Opt}(I)\right]/2 \leq \mathbb{E}\left[\mathcal{W}(I)\right] \leq \mathbb{E}\left[\mathrm{Opt}(I)\right]$, we get,

$$\mathrm{NF}(I) \leq (1+3\delta)\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_3\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} \tag{3.1}$$

with probability of at least $1 - C_4\exp\left(-a_1\mathbb{E}\left[\mathrm{Opt}(I)\right]^{1/3}\right)$ for some constants $C_3, C_4, a_1 > 0$. When the low probability event occurs, we can use the upper bound of $\mathrm{NF}(I) \leq 2\mathrm{Opt}(I) - 1$

to obtain the competitive ratio. Let $p = C_4 \exp\left(-a_1 \mathbb{E}\left[\mathrm{Opt}(I)\right]^{1/3}\right)$.

$$\begin{aligned}\mathbb{E}\left[\mathrm{NF}(I)\right] &\le (1-p)\left((1+3\delta)\mathbb{E}\left[\mathrm{Opt}(I)\right] + K\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + 1\right) + p(2\mathbb{E}\left[\mathrm{Opt}(I)\right] - 1)\\ &= (1+3\delta+2p)\mathbb{E}\left[\mathrm{Opt}(I)\right] + o(\mathbb{E}\left[\mathrm{Opt}(I)\right])\end{aligned}$$

Since $p = o(1)$ when $\mathbb{E}\left[\mathrm{Opt}(I)\right]$ tends to infinity, we obtain that the expected competitive ratio tends to at most $1 + 3\delta < 1 + \varepsilon$.

3.3.1.2 Analysis of Case 2: $|L_0| > \delta^3 \cdot \mathcal{W}(T_0)$

We split our analysis in this case into two parts. We first analyze the number of bins used in the sampling stage T_0 and then analyze the number of bins used in the remaining stages.

Using Lemma 3.11, we obtain w.h.p. that $|L_0| \le \delta^2\mathbb{E}\left[|I_\ell|\right] + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$. Hence,

$$\begin{aligned}\mathbb{E}\left[|I_\ell|\right] &\ge \frac{1}{\delta^2}|L_0| - \frac{1}{\delta^2}\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\\ &\ge \delta\mathcal{W}(T_0) - \frac{1}{\delta^2}\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} && (\text{since } |L_0| > \delta^3 \cdot \mathcal{W}(T_0))\\ &\ge \delta^3\mathbb{E}\left[\mathcal{W}(I)\right] - \left(\delta + \frac{1}{\delta^2}\right)\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} && (3.2)\end{aligned}$$

The last inequality follows from Lemma 3.10. For any $j \ge 1$, using $|T_j|/n \ge \delta^2$ and using Lemma 3.11, we get,

$$|L_j| \ge \delta^5\mathbb{E}\left[\mathcal{W}(I)\right] - (2+\delta^3)\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} \tag{3.3}$$

Each of the Eqs. (3.2),(3.3) holds with a probability of at least $1 - C\exp\left(-a\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$ for some constants $C, a > 0$.

Note that $\mathcal{W}(I) \ge \mathrm{Opt}(I)/2$. So from now on, we assume that there exist constants $C_1, C_2 > 0$ which depend on δ such that w.h.p. both the following inequalities hold.

$$\mathbb{E}\left[|I_\ell|\right] \ge C_1 \cdot \mathbb{E}\left[\mathrm{Opt}(I)\right] \tag{3.4}$$

$$|L_j| \ge C_2 \cdot \mathbb{E}\left[\mathrm{Opt}(I)\right] \tag{3.5}$$

- **Analysis of the Sampling Stage:** Recall that the number of items considered in the sampling stage is $\delta^2 n$. We will bound the number of large items and the weight of items in this stage using Bernstein's inequality.

  1. Since sampling stage has $\delta^2 n$ items, $\mathbb{E}\left[|L_0|\right] = \delta^2\mathbb{E}\left[|I_\ell|\right]$. By applying Bernstein's

inequality for $X_1, X_2, \ldots, X_{|T_0|}$ where X_i takes value 1 is x_i is large and 0 otherwise, we get,

$$\begin{aligned}\mathbb{P}\left[|L_0| \geq 2\delta^2\mathbb{E}\left[|I_\ell|\right]\right] &= \mathbb{P}\left[|L_0| \geq \mathbb{E}\left[|L_0|\right] + \delta^2\mathbb{E}\left[|I_\ell|\right]\right] \\ &\leq 2\exp\left(-\frac{\delta^4\mathbb{E}\left[|I_\ell|\right]^2}{2\mathbb{E}\left[|L_0|\right] + \frac{2}{3}\delta^2\mathbb{E}\left[|I_\ell|\right]}\right) \leq 2\exp\left(-\frac{1}{3}\delta^2\mathbb{E}\left[|I_\ell|\right]\right) \\ &\leq 2\exp\left(-a_1 \cdot \mathbb{E}\left[\mathrm{Opt}(I)\right]\right) \qquad \text{(from Eq. (3.4))}\end{aligned}$$

for some constant $a_1 > 0$. So, with high probability, $|L_0| \leq 2\delta^2\mathbb{E}\left[|I_\ell|\right] \leq 2\delta\mathbb{E}\left[\mathrm{Opt}(I)\right]$.

2. Similarly, $\mathbb{E}\left[\mathcal{W}(T_0)\right] = \delta^2\mathbb{E}\left[\mathcal{W}(I)\right]$. By applying Bernstein's inequality for $X_1, X_2, \ldots, X_{|T_0|}$ where X_i takes value x_i, we get,

$$\begin{aligned}\mathbb{P}\left[\mathcal{W}(T_0) \geq 2\delta^2\mathbb{E}\left[\mathcal{W}(I)\right]\right] &= \mathbb{P}\left[\mathcal{W}(T_0) \geq \mathbb{E}\left[\mathcal{W}(S_0)\right] + \delta^2\mathbb{E}\left[\mathcal{W}(I)\right]\right] \\ &\leq 2\exp\left(\frac{-\delta^4\mathbb{E}\left[\mathcal{W}(I)\right]^2}{2\delta^2\mathbb{E}\left[\mathcal{W}(T_0)\right] + \frac{2}{3}\delta^2\mathbb{E}\left[\mathcal{W}(I)\right]}\right) \\ &\leq 2\exp\left(\frac{-\delta^2}{3}\mathbb{E}\left[\mathcal{W}(I)\right]\right) \leq 2\exp\left(\frac{-\delta^2\mathbb{E}\left[\mathrm{Opt}(I)\right]}{6}\right)\end{aligned}$$

So, with high probability we have, $\mathcal{W}(T_0) \leq 2\delta^2\mathbb{E}\left[\mathcal{W}(I)\right] \leq 2\delta^2\mathbb{E}\left[\mathrm{Opt}(I)\right]$.

Since the number of bins opened by Next-Fit $\mathrm{NF}(T_0)$ is at most $|L_0| + \frac{1}{1-\delta}\mathcal{W}(T_0) + 1$, using the bounds on the number of large items and weight of small items in sampling stage, w.h.p. we have,

$$\mathrm{NF}(T_0) \leq 2\delta\mathbb{E}\left[\mathrm{Opt}(I)\right] + \frac{2\delta^2}{1-\delta}\mathbb{E}\left[\mathrm{Opt}(I)\right] \leq 4\delta\mathbb{E}\left[\mathrm{Opt}(I)\right] \tag{3.6}$$

- **Analysis of the Remaining Stages:** Consider any stage T_j $(j > 0)$ after the sampling stage. Note that $|T_{j-1}| = |T_j|$. A bin can be opened in three different ways.

  1. When a new bin is opened while packing the set of proxy items D_j using $\mathcal{A}_\alpha$ (see the algorithm description in Section 3.3.1, and line 12 in Algorithm 1).
  2. When a large item can't replace a proxy item and hence a new bin is opened for it (line 27 in Algorithm 1).

3. When a small item can't fit in the set of S-slots and hence a new bin is opened with a single S-slot spanning the entire bin (line 32 in Algorithm 1).

In our analysis, first we bound the number of bins opened by $\mathcal{A}_\alpha$ for proxy items; we use Lemma 3.12 to obtain this bound. Then we show that the number of large items which cannot replace a proxy item would be very small by using upright matching arguments (Lemma 3.4). For small items, we bound the number of new bins opened by using the fact that $\mathcal{W}(S_j)$ and $\mathcal{W}(S_{j-1})$ are very close which will be proved using Bernstein's inequality.

Now we analyze the number of bins opened in the *first way* (say A^j_{proxy}). This is nothing but the number of bins opened by $\mathcal{A}_\alpha$ to pack D_j (which is nothing but T_{j-1}). Since, $\mathcal{A}_\alpha$ has an AAR of α, by using Lemma 3.12, we have,

$$\begin{aligned} A^j_{\text{proxy}} &= \mathcal{A}_\alpha(T_{j-1}) \\ &\le \alpha \text{Opt}(T_{j-1}) + o(\text{Opt}(T_{j-1})) \\ &\le \alpha(1+2\delta)\frac{|T_{j-1}|}{n}\mathbb{E}\left[\text{Opt}(I)\right] + C_3\mathbb{E}\left[\text{Opt}(I)\right]^{2/3} + o(\text{Opt}(I)) \end{aligned} \tag{3.7}$$

with probability at least $1 - \exp\left(-a_2 \log(\mathbb{E}\left[\text{Opt}(I)\right])^{1/3}\right)$, for some constants $C_3, a_2 > 0$.

We now bound the number of bins opened in the *second way*. Using Lemma 3.11, we get that w.h.p., $|L_{j-1}| \ge \frac{|T_{j-1}|}{n}\mathbb{E}\left[|I_\ell|\right] - \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$, and $|L_{j-1}| \le \frac{|T_{j-1}|}{n}\mathbb{E}\left[|I_\ell|\right] + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$. Since $|T_{j-1}| = |T_j|$ we have w.h.p,

$$|L_j| \le |L_{j-1}| + 2\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}. \tag{3.8}$$

So, w.h.p. the number of large items in stage T_j doesn't exceed that of those in stage T_{j-1} by a large number. Now consider the number of bins opened because there is no feasible proxy item that can be replaced i.e., when the if condition at line 22 of Algorithm 1 fails and hence we execute line 27. Let this number be A^j_{unmatch}. We can interpret this number as the number of unmatched items when we use the stochastic matching variant $\mathcal{M}$ from [RT93c] as follows. We can interpret each item $\bar{t} \in L_{j-1}$ as a point $P_{\bar{t}} := (+1, \bar{t})$ and each point $t \in L_j$ as a point $P_t := (-1, t)$. For simplicity, let's call the points with $+1$ as their first coordinate as *plus* points and the points with -1 as their first coordinate as *minus* points. We match a point $P_{\bar{t}}$ with P_t iff t replaced $\bar{t}$ in our algorithm. It is shown in [Sho86] that such matching is always maximum. Hence the number of items that open new bins is at most the number of unmatched points in this maximum matching. There are two differences though. First, $|L_{j-1}|$ may not be greater than $|L_j|$; but as we have shown, w.h.p, the difference can at most be $2\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$. Secondly, in the matching variant $\mathcal{M}$, every point has equal chance to be a plus point or minus point. However, this is also inconsequential, since using concentration bounds for binomial random variables, we can show that the number of plus points/minus points lie in the range $\left(\mathbb{E}\left[|L_{j-1}|\right] \pm \mathbb{E}\left[|L_{j-1}|\right]^{2/3}\right)$ w.h.p. Hence by Lemma 3.4, we obtain that there exist constants a_3, C_4, K_1 s.t.

$$\mathbb{P}\left[A^j_{\text{unmatch}} \geq K_1\sqrt{|L_{j-1}|}\left(\log|L_{j-1}|\right)^{3/4} + 2\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} + 2\mathbb{E}\left[|L_{j-1}|\right]^{2/3}\right] \\ \leq C_4\exp\left(-a_3(\log|L_{j-1}|)^{3/2}\right)$$

We can simplify the above inequality using Eqs. (3.5) and (3.8) and the fact that $\text{Opt}(I) \leq 2\mathcal{W}(I)$ to obtain that there exist constants $a_4, C_4, K_2 > 0$ such that,

$$\mathbb{P}\left[A^j_{\text{unmatch}} \geq K_2\mathbb{E}\left[\text{Opt}(I)\right]^{2/3}\right] \leq C_4\exp\left(-a_4(\log\mathbb{E}\left[\text{Opt}(I)\right])^{3/2}\right) \tag{3.9}$$

The only part left is to bound the number of bins opened by small items in *third way*. Let this number be A^j_{small}. We will bound this by using the concentration of weights of small items in T_{j-1} and T_j. Consider the random variables $X_1, X_2 \ldots X_n$ where $X_i = 0$ if x_i is large, and $X_i = x_i$ otherwise. We have that

$$\mathcal{W}(S_j) = \sum_{X_i : x_i \in T_j} X_i \text{ and } \mathcal{W}(S_{j-1}) = \sum_{X_i : x_i \in T_{j-1}} X_i$$

By applying Bernstein's inequality (similar to Lemma 3.10) we get,

$$\mathcal{W}(S_j) \leq \frac{|T_j|}{n}\mathcal{W}(I_s) + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}, \text{ and}$$

$$\mathcal{W}(S_{j-1}) \geq \frac{|T_{j-1}|}{n}\mathcal{W}(I_s) - \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$$

with a probability of at least $1 - C_5\exp\left(-a_5\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$ for some constants $C_5, a_5 > 0$. Combining both, we get,

$$\mathcal{W}(S_j) \leq \mathcal{W}(S_{j-1}) + 2\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} \tag{3.10}$$

The initial allocated space for small items at the start of stage j in the proxy packing $\mathcal{P}_j$ (i.e., the total size of all the S-slots in $\mathcal{P}_j$) is $\mathcal{W}(S_{j-1})$. Recall that $B_j^{(k)}$ denotes the k^{th} bin in the packing of $\mathcal{P}_j$. While packing the small items, if the S-slot in $B_j^{(k)}$ cannot accommodate a small item, this means that the remaining space in this S-slot is at most δ. So, the weight of small items which overflow the space allocated in packing $\mathcal{P}_j$ is at most $\mathcal{W}(S_j)-\mathcal{W}(S_{j-1})+\delta\,|\mathcal{P}_j|$ and this entire weight is packed in new bins opened exclusively for small items. Each of these bins (except possibly one) have an occupancy of at least $(1-\delta)$. Since $\mathcal{A}_\alpha$ is α-approximation algorithm, $|\mathcal{P}_j| = \mathcal{A}_\alpha(T_{j-1}) \le \alpha\mathrm{Opt}(T_{j-1}) + o(\mathrm{Opt}(I))$. Using Eqs. (3.7) and (3.10) we get,

$$\begin{aligned} A^j_{\text{small}} &\le \frac{1}{1-\delta}\left(\mathcal{W}(S_j) - \mathcal{W}(S_{j-1}) + \delta\cdot\alpha\mathrm{Opt}(T_{j-1}) + o(\mathrm{Opt}(I))\right) + 1 \\ &\le 2\delta\cdot\alpha\frac{|T_{j-1}|}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_3\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I)) \end{aligned} \tag{3.11}$$

with high probability. Combining Eqs. (3.7), (3.9) and (3.11), the number of bins, A_j, opened in the stage j is bounded as,

$$\begin{aligned} A_j &= A^j_{\text{proxy}} + A^j_{\text{unmatch}} + A^j_{\text{small}} \\ &\le \alpha(1+4\delta)\frac{|T_{j-1}|}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_6\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I)) \\ &\le \alpha(1+4\delta)\frac{|T_j|}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_6\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I)) \end{aligned} \tag{3.12}$$

w.h.p., for some constant C_6. To bound the sum of all A_js, first note that the number of "remaining stages" is $m-1$ which is a constant dependent on δ. Hence, with high probability,

$$\begin{aligned} \sum_{j=1}^{m-1} A_j &\le \alpha(1+4\delta)\sum_{j=1}^{m-1}\frac{|T_{j-1}|}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + (m-1)\cdot C_6\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I)) \\ &\le \alpha(1+4\delta)\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_7\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I)) \end{aligned} \tag{3.13}$$

for some constant $C_7 > 0$ dependent on δ.

For the sampling stage, from Eq. (3.6), we have $\mathrm{NF}(T_0) \le 4\delta\mathbb{E}\left[\mathrm{Opt}(I)\right]$ with high probability and in all the remaining phases we have

$$\sum_{j=1}^{m-1} A_j \le \alpha(1+4\delta)\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_7\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I))$$

Combining both the results we get that w.h.p. the number of bins opened by `Alg` is,

$$\begin{aligned}\texttt{Alg}(I) &\le \alpha(1+8\delta)\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_7\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I)) \\ &\le \alpha(1+\varepsilon)\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_7\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I)) \qquad (3.14)\end{aligned}$$

In the low probability event when Eq. (3.14) may not hold, we can bound $\texttt{Alg}(I)$ as follows. In the sampling stage, we have that $\mathrm{NF}(T_0) \le 2\mathrm{Opt}(I)-1$. For the remaining stages, we bound the number of bins containing at least one large item and the number of bins containing only small items. Each large item is considered at most twice in the packing: first - when it arrives in the input list, second - when it takes the role of a proxy item. So, the number of bins containing at least one large item is at most $2\left|I_\ell\right|$. In each stage, with one possible exception, every bin opened which has only small items has an occupancy of at least $(1-\delta)$. Combining over all the stages, the number of bins which contain only small items is at most $\frac{1}{1-\delta}\mathcal{W}(I_s)+m$. Thus, we can bound the total number of bins used by `Alg` to be at most $2\mathrm{Opt}(I)+2\left|I_\ell\right|+\frac{1}{1-\delta}\mathcal{W}(I_s)+m$. On the other hand, we know that $\mathrm{Opt}(I) \ge \mathcal{W}(I) \ge \delta\left|I_\ell\right| + \mathcal{W}(I_s)$. Hence, we obtain that $2\left|I_\ell\right| + \frac{1}{1-\delta}\mathcal{W}(I_s) \le \frac{2}{\delta(1-\delta)}\mathrm{Opt}(I)$. Combining all these, we obtain that

$$\texttt{Alg}(I) \le \left(2+\frac{2}{\delta(1-\delta)}\right)\mathrm{Opt}(I)+m \qquad (3.15)$$

Now, to obtain the competitive ratio, suppose Eq. (3.14) holds with probability p $(=1-o(1))$. We combine Eqs. (3.14) and (3.15) similar to the case when $|L_0| \le \delta^3 \cdot \mathcal{W}(T_0)$.

$$\begin{aligned}\mathbb{E}\left[\texttt{Alg}(I)\right] &\le p(\alpha(1+8\delta)\mathbb{E}\left[\mathrm{Opt}(I)\right] + o(\mathbb{E}\left[\mathrm{Opt}(I)\right])) \\ &\quad + (1-p)\left(\left(2+\frac{2}{\delta(1-\delta)}\right)\mathbb{E}\left[\mathrm{Opt}(I)\right]+m\right) \\ &\le \alpha(1+\varepsilon)\mathbb{E}\left[\mathrm{Opt}(I)\right] + o(\mathbb{E}\left[\mathrm{Opt}(I)\right]) \qquad (\text{since } 1-p=o(1) \text{ and } \delta \le \varepsilon/8)\end{aligned}$$

Scaling the initial value ε to ε/α before the start of the algorithm, we obtain a competitive ratio of $\alpha+\varepsilon$.

3.3.2 Getting Rid of the Assumption on the Knowledge of the Input Size

In this subsection, we will extend `Alg` to devise an algorithm for online bin packing with i.i.d. items that guarantees essentially the same competitive ratio without knowing the value of n. We denote this algorithm by `ImpAlg`.

Let $\mu := \delta^2$. We first guess the value of n to be a constant $n_0 := 1/\delta^3$. Then, we run `Alg` until $\min\{n, n_0\}$ items arrive (here, if $\min\{n, n_0\} = n$, then it means that the input stream has ended before n_0 items have arrived). If $n > n_0$, i.e., if there are more items to arrive, then we revise our estimate of n by multiplying it with $(1+\mu)$, i.e., the new value of n is set as $n_1 := (1+\mu)n_0$. We start `Alg` afresh on the next $\min\{n, n_1\} - n_0$ items. If $n > (1+\mu)n_0$, then we set the new guess of n to be $n_2 := (1+\mu)n_1 = (1+\mu)^2 n_0$ and start `Alg` afresh on the next $\min\{n, n_2\} - n_1$ number of items. We continue this process of multiplying our estimate of n with $(1+\mu)$ until all the items arrive. See Fig. 3.1 for an illustration. The pseudocode is provided in Algorithm 2.

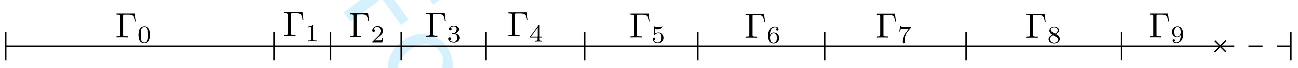


Figure 3.1: The division of input into super-stages to get rid of the assumption on the knowledge of n. The $(j+1)^{\text{th}}$ super-stage is denoted by Γ_j. Stage Γ_0 contains $n_0 = 1/\delta^3$ items, $\Gamma_0 \cup \Gamma_1$ contains $(1+\mu)n_0$ items, $\Gamma_0 \cup \Gamma_1 \cup \Gamma_2$ contains $(1+\mu)^2 n_0$ items and so on. The last super-stage may not be full, but since it is very small in size compared to the entire input, it doesn't affect the performance of the algorithm.

We consider the following partition of the entire input into super-stages as follows: The first super-stage, Γ_0, contains the first n_0 items. The second super-stage, Γ_1, contains the next $n_1 - n_0$ items. In general, for $i > 0$, the $(i+1)^{\text{th}}$ super-stage, Γ_i, contains $\min\{n_i, n\} - n_{i-1}$ items which are given by $x_{n_{i-1}+1}, x_{n_{i-1}+2}, \dots, x_{\min\{n,n_i\}}$. So, essentially, `ImpAlg` can be thought of running `Alg` on each super-stage separately. The number of super-stages is given by $\kappa := \lceil \log_{(1+\mu)}(n/n_0) \rceil$.

Note that $|\Gamma_0| = n_0$, $|\Gamma_1| = \mu n_0$, $|\Gamma_2| = \mu(1+\mu)n_0$ and so on. In general, for $j \in [\kappa - 2]$, $|\Gamma_j| = \mu(1+\mu)^{j-1} n_0$. Now consider the last super-stage $\Gamma_{\kappa-1}$ and note that it may not be full, i.e., it can be the case that $|\Gamma_{\kappa-1}| < \mu(1+\mu)^{\kappa-2} n_0$. So, `Alg` may pack the last super-stage inefficiently. However, note that $|\Gamma_{\kappa-1}|$ can be at most $\mu(1+\mu)^{\kappa-2} n_0 \le \mu n$. Hence the last super-stage contains only a tiny fraction of the input and so, this will have very little effect on the packing of the entire input.

3.3.2.1 Analysis

When n was known we only had $O(1)$ number of stages. However, now we can have $\kappa = \lceil \log_{(1+\mu)}(n/n_0) \rceil$ number of super-stages. There can arise two problems:

- We can not analyze each super-stage individually and then sum up the performance guar-

Algorithm 2 `ImpAlg`: Improving `Alg` to get rid of the assumption on the knowledge of the number of items

```
 1: Input: $I_n(\mathcal{D}) = \{x_1, x_2, ..., x_n\}$.
 2: $n_{-1} \leftarrow 0; n_0 \leftarrow \frac{1}{\delta^3}$; for $j \geq 1, n_j = (1+\mu)n_{j-1}$
 3: $i \leftarrow 0$
 4: while true do
 5:     Run Alg($x_{n_{i-1}+1}, x_{n_{i-1}+2}, \dots, x_{\min\{n_i,n\}}$)
 6:     if the input stream has ended then
 7:         return the packing
 8:     else
 9:         $i \leftarrow i+1$
10:     end if
11: end while
```

antees, as we can not use union bound for a super-constant number of events. Moreover, we can't even use the analysis of `Alg` for the first few super-stages since they might only have a constant number of items. So, we consider the $\kappa_1 := \lceil \log_{(1+\mu)}(\delta^7 n) \rceil$ number of the initial super-stages at a time. We show that these initial super-stages contain only a small fraction of the entire input. Each of the final $(\kappa - \kappa_1)$ super-stages can be individually analyzed using the analysis of `Alg`.

- For each super-stage, we can have a constant number of S-bins (bins which contain only small items) with less occupancy. However, since the number of super-stages itself is a super-constant, this can result in a lot of wasted space. For this, we exploit the monotonicity of Next-Fit to ensure that we can pack small items from a super-stage into empty slots for small items from the previous stages.

We will now proceed to analyze `ImpAlg`. Recall that the number of super-stages is given by $\kappa = \lceil \log_{(1+\mu)}(n/n_0) \rceil$ where n_0 was defined to be $1/\delta^3$. We will split the analysis into two parts. First, we will analyze the number of bins used by our algorithm in the first $\kappa_1 := \lceil \log_{(1+\mu)}(\delta^7 n) \rceil$ super-stages as a whole. Then, we will analyze the final $\kappa_2 := \kappa - \kappa_1$ super-stages considering each one at a time. We call the first κ_1 super-stages as *initial super-stages* and the remaining κ_2 super-stages as *final super-stages.*

Analysis of the initial super-stages: The basic intuition of the analysis of our algorithm in the initial super-stages is as follows: Since only a small fraction of the entire input is present in these κ_1 super-stages, our algorithm uses only a small fraction of bins compared to the optimal packing of the entire input. So, we bound the number of large items and the weight of small items and thus bound the number of bins used.

Lemma 3.14. *Let $\mathcal{A}$ be an algorithm for online bin packing such that in the packing output by $\mathcal{A}$, every bin (except possibly a constant number of bins τ) which contains only small items has an occupancy of at least $(1-\delta)$. Let I be a sequence of n i.i.d. items and let J be any contiguous subsequence of I having size βn $(0 < \beta \leq 1)$. Suppose $\mathcal{A}$ works in a way such that it packs at most ν copies of any large item where ν is a constant*[4]*. Then the following inequality holds with high probability.*

$$\mathcal{A}(J) \leq \frac{2\beta\nu}{\delta(1-\delta)}\mathbb{E}\left[\mathrm{Opt}(I)\right] + o(\mathbb{E}\left[\mathrm{Opt}(I)\right])$$

Proof. Let J_ℓ, I_ℓ respectively denote the set of large items in J, I and let J_s, I_s respectively denote the set of small items in J, I. We prove the lemma considering two cases.

First, we consider the case when $|J_\ell| \leq \delta \mathrm{Opt}(J)$. The number of bins in the packing of J by $\mathcal{A}$ which contain at least one large item is upper bounded by $\nu\,|J_\ell|$. The number of bins which contain only small items is upper bounded by $\mathcal{W}(J_s)/(1-\delta) + \tau$. Hence,

$$\begin{aligned}
\mathcal{A}(J) &\leq \nu\,|J_\ell| + \frac{\mathcal{W}(J_s)}{1-\delta} + \tau \\
&\leq \nu\delta\mathrm{Opt}(J) + \frac{\mathrm{Opt}(J)}{1-\delta} + \tau \\
&= \frac{1+\nu(1-\delta)\delta}{1-\delta}\mathrm{Opt}(J) + \tau \\
&\leq 2\nu(1+\delta)\mathrm{Opt}(J) + \tau && \text{(since } 0 < \delta < 1/2 \text{ and } \nu \geq 1) \\
&\leq 2\beta\nu(1+\delta)(1+2\delta)\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_0\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} && \text{w.h.p. for some constant } C_0 \\
& && \text{(using Lemma 3.12)} \\
&\leq \frac{2\beta\nu}{\delta(1-\delta)}\mathbb{E}\left[\mathrm{Opt}(I)\right] + o(\mathbb{E}\left[\mathrm{Opt}(I)\right]) && \text{(since } \delta < 1/2,\ 1+\delta < \tfrac{1}{1-\delta} \text{ and } 1+2\delta < \tfrac{1}{\delta})
\end{aligned}$$

Next, we consider the case when $|J_\ell| > \delta\mathrm{Opt}(J)$. We will again bound the quantity $\nu\,|J_\ell| + \frac{\mathcal{W}(J_s)}{1-\delta} + \tau$. To bound $|J_\ell|$, we use Bernstein's inequality (Lemma 3.6). Let $X_1, X_2, \ldots, X_{|J|}$ be random variables where X_i takes value 1 if the i^{th} item in J is large and 0 otherwise. Then, clearly, $|J_\ell| = \sum_{i=1}^{|J|} X_j$. Also, note that $\mathbb{E}\left[|J_\ell|\right] = \beta\mathbb{E}\left[|I_\ell|\right]$. Moreover, we can derive the

[4]For example, `Alg` does this. For each stage, it computes the proxy packing of the previous stage. Hence, every large item is potentially packed twice. In the worst case, these copies may not get replaced, thus resulting in wasted space.

following inequalities that hold with high probability.

$$\begin{aligned}
\mathbb{E}\left[|I_\ell|\right] &\geq \frac{1}{\beta}|J_\ell| - \frac{1}{\beta}\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} && \text{(using Lemma 3.11)}\\
&\geq \frac{\delta}{\beta}\mathrm{Opt}(J) - \frac{1}{\beta}\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\\
&\geq \frac{\delta}{\beta}\mathcal{W}(J) - \frac{1}{\beta}\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\\
&\geq \frac{\delta}{\beta}\left(\beta\mathbb{E}\left[\mathcal{W}(I)\right] - \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\right) - \frac{1}{\beta}\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} && \text{(using Lemma 3.10)}\\
&\geq \frac{\delta}{2}\mathbb{E}\left[\mathrm{Opt}(I)\right] - o(\mathbb{E}\left[\mathrm{Opt}(I)\right]) && \text{(since } \mathcal{W}(I) \leq \mathrm{Opt}(I) \leq 2\mathcal{W}(I))
\end{aligned}$$

Hence, from now on, we will assume that there exists a constant $a_1 > 0$ such that

$$\mathbb{E}\left[|I_\ell|\right] \geq a_1\mathbb{E}\left[\mathrm{Opt}(I)\right] \tag{3.16}$$

holds with high probability. Using Bernstein's inequality (Lemma 3.6),

$$\begin{aligned}
\mathbb{P}\left[|J_\ell| \geq 2\beta\mathbb{E}\left[|I_\ell|\right]\right] &= \mathbb{P}\left[|J_\ell| \geq \mathbb{E}\left[|J_\ell|\right] + \beta\mathbb{E}\left[I_\ell\right]\right]\\
&\leq 2\exp\left(-\frac{\beta^2\mathbb{E}\left[|I_\ell|\right]^2}{2\mathbb{E}\left[|J_\ell|\right] + \frac{2}{3}\beta\mathbb{E}\left[|I_\ell|\right]}\right)\\
&= 2\exp\left(-\frac{3}{8}\beta\mathbb{E}\left[|I_\ell|\right]\right)\\
&\leq 2\exp\left(-\frac{3}{8}\beta a_1\cdot\mathbb{E}\left[\mathrm{Opt}(I)\right]\right) && \text{(from Eq. (3.16))}
\end{aligned}$$

Now to bound $\mathcal{W}(J_s)$, we will again use Bernstein's inequality on a new set of random variables $X_1, X_2, \ldots, X_{|J|}$ where X_i equals the weight of the i^{th} item in J if it is small and 0 otherwise. Clearly, $\sum_{i=1}^{|J|} X_i = \mathcal{W}(J_s)$ and $\mathbb{E}\left[\mathcal{W}(J_s)\right] = \beta\mathbb{E}\left[\mathcal{W}(I_s)\right] \leq \beta\mathbb{E}\left[\mathcal{W}(I)\right]$. Thus,

$$\begin{aligned}
\mathbb{P}\left[\mathcal{W}(J_s) \geq 2\beta\mathbb{E}\left[\mathcal{W}(I)\right]\right] &\leq \mathbb{P}\left[\mathcal{W}(J_s) \geq \mathbb{E}\left[\mathcal{W}(J_s)\right] + \beta\mathbb{E}\left[\mathcal{W}(I)\right]\right]\\
&\leq 2\exp\left(\frac{-\beta^2\mathbb{E}\left[\mathcal{W}(I)\right]^2}{2\mathbb{E}\left[\mathcal{W}(J_s)\right] + \frac{2}{3}\beta\mathbb{E}\left[\mathcal{W}(I)\right]}\right)\\
&= 2\exp\left(\frac{-3\beta}{8}\mathbb{E}\left[\mathcal{W}(I)\right]\right)\\
&\leq 2\exp\left(\frac{-3\beta\mathbb{E}\left[\mathrm{Opt}(I)\right]}{16}\right) && \text{(since } 2\mathcal{W}(I) \geq \mathrm{Opt}(I))
\end{aligned}$$

Thus, with high probability, we have that $|J_\ell| \le 2\beta\mathbb{E}[|I_\ell|]$ and $\mathcal{W}(J_s) \le 2\beta\mathbb{E}[\mathcal{W}(I)]$. Hence,

$$\begin{aligned}\mathcal{A}(J) &\le \nu|J_\ell| + \frac{\mathcal{W}(J_s)}{1-\delta} + \tau \\ &\le 2\nu\beta\mathbb{E}[|I_\ell|] + \frac{2\beta\mathbb{E}[\mathcal{W}(I)]}{1-\delta} + \tau \\ &\le \frac{2\nu\beta}{\delta}\mathbb{E}[\mathrm{Opt}(I)] + \frac{2\beta\mathbb{E}[\mathrm{Opt}(I)]}{1-\delta} + \tau \\ &= \frac{2\nu\beta}{\delta(1-\delta)}\mathbb{E}[\mathrm{Opt}(I)] + o(\mathbb{E}[\mathrm{Opt}(I)])\end{aligned}$$

This completes the proof. □

With the help of the above lemma, we proceed to analyze `ImpAlg` for the initial (first κ_1) super-stages. Note that the number of items in the initial super-stages is given by $(1+\varepsilon)^{\kappa_1-1}n_0 \le (\delta^7 n)n_0 = \delta^4 n$. Since `Alg` satisfies the properties of $\mathcal{A}$ in Lemma 3.14 and `ImpAlg` just applies `Alg` multiple times, we can use Lemma 3.14 to analyze `ImpAlg`.

There is one difficulty though. Let's call a bin which contains only small items to be an S-bin. Although in a super-stage the number of S-bins not filled up to a level of at least $(1-\delta)$ is a constant, the number of initial super-stages itself is not a constant. We can work around this problem by continuing (Next-Fit) NF to pack the small items in the S-slots created during the previous stages and the previous super-stages as well. In other words, instead of packing the small items of a stage in S-slots created only during that stage, we keep a global set of S-slots and pack the small items in them using NF. We now have to show that our algorithm doesn't increase the number of bins with this change. Since the way in which the large items are packed hasn't changed, the number of bins that contain large items does not change. Since NF is monotone (even with varying bin sizes, [Mur88]), the number of bins containing only the small items will either decrease or stays the same.

Using Lemma 3.14 with $\beta = \delta^4$ and $\nu = 2$ (since in each super-stage, a large item is packed at most twice – once when it arrives as a real item, and once when it is used as a proxy item), we obtain that with high probability,

$$\begin{aligned}\texttt{Alg}(\Gamma_0) + \texttt{Alg}(\Gamma_1) + \cdots + \texttt{Alg}(\Gamma_{\kappa_1-1}) &\le \frac{2(2)\delta^4}{\delta(1-\delta)}\mathbb{E}[\mathrm{Opt}(I)] + o(\mathbb{E}[\mathrm{Opt}(I)]) \\ &\le 8\delta^3\mathbb{E}[\mathrm{Opt}(I)] + o(\mathbb{E}[\mathrm{Opt}(I)]) \\ &\le 8\delta\mathbb{E}[\mathrm{Opt}(I)] + o(\mathbb{E}[\mathrm{Opt}(I)]) \qquad (3.17)\end{aligned}$$

Analysis of the final super-stages: For this part, the important thing to note is that

$\kappa_2 \leq \lceil \log_{(1+\mu)}(1/(\delta^7 n_0)) \rceil$ is a constant. Moreover, since the number of items in the initial super-stages is at least $\delta^4 n/(1+\mu)$, each of the final super-stages has at least $\mu\delta^4 n/(1+\mu)$ items (which tends to infinity in the limiting case). Thus, we can use the analysis of `Alg` for each of these final super-stages. As mentioned, since the last super-stage might not be full, we analyze the last super-stage differently. All the other super-stages are full. So, we can directly use the analysis of `Alg`. For all $\kappa_1 \leq i < \kappa - 1$, from the analysis of `Alg` (Eq. (3.14)), we have that with high probability

$$\texttt{Alg}(\Gamma_i) \leq \alpha(1+8\delta)\mathbb{E}\left[\mathrm{Opt}(\Gamma_i)\right] + C\mathbb{E}\left[\mathrm{Opt}(\Gamma_i)\right]^{2/3} + o(\mathrm{Opt}(\Gamma_i))$$

for some constant C.

Now, to bound $\mathrm{Opt}(\Gamma_i)$ in terms of $\mathrm{Opt}(I)$, we can use Lemma 3.12. Thus, the above inequality transforms into

$$\begin{aligned}\texttt{Alg}(\Gamma_i) &\leq \alpha(1+8\delta)(1+2\delta)\frac{|\Gamma_i|}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_1\mathbb{E}\left[\mathrm{Opt}(\Gamma_i)\right]^{2/3} + o(\mathrm{Opt}(\Gamma_i)) \\ &\leq \alpha(1+8\delta)(1+2\delta)\frac{|\Gamma_i|}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_1\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I))\end{aligned}$$

for some constant C_1. Summing over all i ($\kappa_1 \leq i < \kappa - 1$), and observing that $\kappa - \kappa_1$ is a constant, we obtain that with high probability,

$$\begin{aligned}&\texttt{Alg}(\Gamma_{\kappa_1}) + \texttt{Alg}(\Gamma_{\kappa_1+1}) + \cdots + \texttt{Alg}(\Gamma_{\kappa-2}) \\ \leq&\alpha(1+8\delta)(1+2\delta)\sum_{i=\kappa_1}^{\kappa-2}\frac{|\Gamma_i|}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_2\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I)) \\ \leq&\alpha(1+26\delta)\sum_{i=\kappa_1}^{\kappa-2}\frac{|\Gamma_i|}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_2\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I)) \\ \leq&\alpha(1+26\delta)\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_2\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I)) \qquad (3.18)\end{aligned}$$

for some constant C_2.

Now consider the last super-stage $\Gamma_{\kappa-1}$. If $|\Gamma_{\kappa-1}|$ is exactly equal to $n_{\kappa-1} - n_{\kappa-2}$, then we can obtain the same bound as Eq. (3.18). However, this might not be the case. The last super-stage contains $n - n_{\kappa-2}$ items.

$$\begin{aligned} n - n_{\kappa-2} \le n_{\kappa-1} - n_{\kappa-2} = (1+\mu)^{\kappa-1} n_0 - (1+\mu)^{\kappa-2} n_0 &= \mu(1+\mu)^{\kappa-2} n_0 \\ &= \mu n_{\kappa-2} \\ &\le \mu n \end{aligned}$$

Hence, the size of the last super-stage is at most μ fraction of the entire input size. We will again use Lemma 3.14 with $\beta = \mu$ and $\nu = 2$ for the last super-stage. We thus obtain that

$$\begin{aligned} \texttt{Alg}(\Gamma_{\kappa-1}) &\le \frac{(2)(2)(|\Gamma_{\kappa-1}|/n)}{n\delta(1-\delta)} \mathbb{E}[\mathrm{Opt}(I)] + o(\mathbb{E}[\mathrm{Opt}(I)]) \\ &\le \frac{4\mu}{\delta(1-\delta)} \mathbb{E}[\mathrm{Opt}(I)] + o(\mathbb{E}[\mathrm{Opt}(I)]) \\ &\le 8\delta \mathbb{E}[\mathrm{Opt}(I)] + o(\mathbb{E}[\mathrm{Opt}(I)]) \end{aligned} \tag{3.19}$$

with high probability. The last inequality follows since $\mu = \delta^2$ and $\delta < 1/2$.

Summing Eqs. (3.17) to (3.19), we obtain that with high probability, for some constant C_4,

$$\begin{aligned} \texttt{ImpAlg}(I) &= \sum_{j=0}^{\kappa_1 - 1} \texttt{Alg}(\Gamma_j) + \sum_{j=\kappa_1}^{\kappa-2} \texttt{Alg}(\Gamma_j) + \texttt{Alg}(\Gamma_{\kappa-1}) \\ &\le (\alpha(1+26\delta) + 16\delta)\mathbb{E}[\mathrm{Opt}(I)] + C_2 \mathbb{E}[\mathrm{Opt}(I)]^{2/3} + o(\mathbb{E}[\mathrm{Opt}(I)]) + o(\mathrm{Opt}(I)) \\ &\le \alpha(1+42\delta)\mathbb{E}[\mathrm{Opt}(I)] + C_2 \mathbb{E}[\mathrm{Opt}(I)]^{2/3} + o(\mathbb{E}[\mathrm{Opt}(I)]) + o(\mathrm{Opt}(I)) \end{aligned} \tag{3.20}$$

Eq. (3.20) holds with high probability, say $p = 1 - o(1)$. In the scenario where the low probability event occurs, we can bound the number of bins used by `ImpAlg` using Lemma 3.14 with $\beta = 1$ and $\nu = 2$:

$$\texttt{ImpAlg}(I) \le \frac{2 \cdot 2}{\delta(1-\delta)} \mathbb{E}[\mathrm{Opt}(I)] + o(\mathbb{E}[\mathrm{Opt}(I)])$$

Let E be the event when Eq. (3.20) holds. Then

$$\begin{aligned}\mathbb{E}\left[\texttt{ImpAlg}(I)\right] &= \mathbb{E}\left[\texttt{ImpAlg}(I)|E\right]\mathbb{P}\left[E\right] + \mathbb{E}\left[\texttt{ImpAlg}(I)|\overline{E}\right]\mathbb{P}\left[\overline{E}\right]\\ &\le \Big(\left(\alpha\left(1+42\delta\right)\right)\mathbb{E}\left[\mathrm{Opt}(I)\right] + o(\mathbb{E}\left[\mathrm{Opt}(I)\right])\Big)(1-o(1))\\ &\quad + \left(\frac{4}{\delta(1-\delta)}\mathbb{E}\left[\mathrm{Opt}(I)\right]+1\right)o(1)\\ &= \left(\alpha\left(1+42\delta\right)\right)\mathbb{E}\left[\mathrm{Opt}(I)\right] + o(\mathbb{E}\left[\mathrm{Opt}(I)\right]\end{aligned}$$

Choosing $\delta = \frac{\varepsilon}{42\alpha}$ ensures that the competitive ratio of `ImpAlg` is $\alpha + \varepsilon$.

3.4 Best-Fit under the Random-Order Model

In this section, we will prove Theorems 3.2 and 3.3.

3.4.1 When Item Sizes are Larger than 1/3

First, let us recall upright matching and a related result that we will be using.

Upright Matching Problem. For a positive integer k, let S_k denote the set of all permutations of $[k]$. Consider a set of points P in the two-dimensional coordinate system. Suppose each item is marked as a *plus* point or a *minus* point. Let P^+ denote the set of plus points and let P^- denote the set of minus points. An edge exists between two points p^+, p^- iff $p^+ \in P^+, p^- \in P^-$ and iff p^+ lies above and to the right of p^-, i.e., both the coordinates of p^+ are greater than or equal to the corresponding coordinates of p^-. The objective is to find a maximum matching in this graph or, in other words, minimize the number of unmatched points. We denote the number of unmatched points by $U(P)$.

We will use the following variant of upright matching to prove the final result. Refer to [Car19] for the proof of the following lemma.

Lemma 3.15. *[Car19] Let $k \in \mathbb{N}$ and let $\mathcal{A} = \{a_1, a_2, \ldots, a_{2k}\}$ such that $a_i \ge a_{k+i}$ for all $i \in [k]$. Define a set of* plus *points $P^+ = \{(i, a_i) : i \in [k]\}$ and a set of* minus *points $P^- = \{(i, a_i) : k < i \le 2k\}$. Suppose we randomly permute the x-coordinates of $P^+ \cup P^-$, i.e., for a uniform random permutation $\pi \in S_{2k}$, we redefine P^+ and P^- as $P^+ = \{(\pi(i), a_i) : i \in [k]\}$ and $P^- = \{(\pi(i), a_i) : k < i \le 2k\}$. Let $P = P^+ \cup P^-$. Then, there exist universal constants $a, C, K > 0$ such that*

$$\mathbb{P}\left[U(P) \ge K\sqrt{k}(\log k)^{3/4}\right] \le C\exp(-a(\log k)^{3/2}) \tag{3.21}$$

Remark 3.16. *In the above lemma, if we change the definitions of P^+, P^- to be $P^+_{\text{new}} =$*

$\{(-\pi(i), a_i) : i \in [k]\}$, $P^-_{\text{new}} = \{(-\pi(i), a_i) : k < i \le 2k\}$, the guarantee given by Eq. (3.21) doesn't change since the new set $P^+_{\text{new}} \cup P^-_{\text{new}}$ can be constructed by taking a mirror image of the original set $P^+ \cup P^-$ with respect to the y-axis. Since we consider random permutations, the probability of a set and its mirror image is the same.

With the above lemma and remark at hand, we now proceed to prove Theorem 3.2. Albers et al. [AKL21a] showed that the random-order ratio of the Best-Fit algorithm is at most 1.25 when all the item sizes are more than 1/3. In this section, we improve it further and show that, Best-Fit for this special case under the random-order model is nearly optimal. We first show that the Modified Best-Fit algorithm [CJJLS93] is nearly optimal and we analyze this using the above variant of stochastic upright matching. The Modified Best-Fit (MBF) algorithm is the same as BF except that it closes a bin if it receives an item of size less than 1/2. Shor[Sho86] showed that MBF *dominates* BF, i.e., for any instance I, $\text{BF}(I) \le \text{MBF}(I)$. MBF can be easily reduced to upright matching as follows. Given an instance $I = \{x_1, \dots, x_n\}$, for any item $x_i \in I$, $x_i \in P^-$ if $x_i \le 1/2$ with x-coordinate as $-i$ and y-coordinate as x_i, and $x_i \in P^+$ if $x_i > 1/2$ with x-coordinate as $-i$ and y-coordinate as $1 - x_i$. So, any item x_s of size $\le 1/2$ can be matched with an item x_ℓ of size $> 1/2$ if and only if, x_ℓ arrives before x_s and the remaining space in the bin occupied by x_ℓ is more than the size of x_s.

Define an item x_i as a large item (L) if $x_i > 1/2$; otherwise, as a medium item (M) if $x_i \in (1/3, 1/2]$. We define a bin as LM-bin if it contains one large item and one medium item. We use the following lemma which was proved in [AKL21a] using the monotonicity property of BF when all item sizes are more than 1/3.

Lemma 3.17. *[AKL21a] Let I be any list that can be packed into $\text{Opt}(I)$ number of LM-bins. If Best-Fit has an AAR of α for I, then it has an AAR of α for any list of items larger than $1/3$ as well.*

Consider an input instance which has an optimal packing containing only LM-bins. Consider the number of bins opened by MBF for such instances. Each large item definitely opens a new bin, and a medium item opens a new bin if and only if it can not be placed along with a large item, i.e., it is “unmatched”. So, the number of bins opened by MBF equals (number of large items+number of unmatched medium items). Now, we will prove our result.

Theorem 3.18. *For any list I of items larger than $1/3$, the random-order ratio $\text{RR}^\infty_{\text{BF}} = 1$.*

Proof. From Lemma 3.17, it is enough to prove the theorem for any list I that can be packed in $\text{Opt}(I)$ LM-bins. So, we can assume that I has k large items and k medium items where $\text{Opt}(I) = k$. Now consider the packing of MBF for a randomly permuted list I_σ. We have,

$\mathrm{MBF}(I_\sigma) = (k + \text{number of unmatched medium items})$. Since the optimal packing has all the items matched, we can reduce the following case into the matching variant in Lemma 3.15: Let ℓ_i, m_i denote the sizes of the large item and the medium item respectively in the i^{th} bin of the optimal solution. For $i \in [k]$, we let $a_i = 1 - \ell_i$ and $a_{k+i} = m_i$ and let $\mathcal{A} = \{a_1, a_2, \dots, a_{2k}\}$. Note that the required condition in Lemma 3.15, i.e., $a_i \geq a_{k+i}$ is satisfied.

The arrival order is randomly sampled from S_{2k}. So, we have

$$\mathrm{MBF}(I_\sigma) = k + \frac{U(P)}{2} \leq k + K\sqrt{k}(\log k)^{3/4}$$

with probability of at least $1 - C\exp(-a(\log k)^{3/2})$ for some universal constants $a, C, K > 0$. Since MBF dominates BF we have

$$\mathbb{P}\left[\mathrm{BF}(I_\sigma) \leq k + K\sqrt{k}(\log k)^{3/4}\right] \geq 1 - C\exp(-a(\log k)^{3/2}).$$

In case the high probability event does not occur, we can use the bound of $\mathrm{BF}(I) \leq 1.7\mathrm{Opt}(I) + 2$. Let $p := C\exp(-a(\log k)^{3/2})$. Then

$$\begin{aligned}\mathbb{E}\left[\mathrm{BF}(I_\sigma)\right] &\leq p(1.7\mathbb{E}\left[k\right] + 2) + (1-p)(\mathbb{E}\left[k\right] + K\sqrt{\mathbb{E}\left[k\right]}(\log \mathbb{E}\left[k\right])^{3/4}) \\ &\leq \mathbb{E}\left[\mathrm{Opt}(I)\right] + o(\mathbb{E}\left[\mathrm{Opt}(I)\right]) \qquad \text{(since } p = o(1))\end{aligned}$$

So, we get: $\mathrm{RR}_{\mathrm{BF}}^{\infty} = \limsup_{k\to\infty}\left(\sup_{I:\mathrm{Opt}(I)=k}(\mathbb{E}[\mathrm{BF}(I_\sigma)]/\mathrm{Opt}(I))\right) = 1$. This completes the proof. □

3.4.2 The 3-Partition Problem under Random-Order Model

In this section, we analyze the Best-Fit algorithm under the random-order model given that the item sizes lie in the range $(1/4, 1/2]$, and thus prove Theorem 3.3. We call an item *small* if its size lies in the range $(1/4, 1/3]$ and *medium* if its size lies in the range $(1/3, 1/2]$. Let I be the input list of items and let $n := |I|$. Recall that given σ, a uniform random permutation of $[n]$, I_σ denotes the list I permuted according to σ. We denote by $\mathrm{Opt}(I_\sigma)$, the number of bins used in the optimal packing of I_σ and by $\mathrm{BF}(I_\sigma)$, the number of bins used by Best-Fit to pack I_σ. Note that $\mathrm{Opt}(I_\sigma) = \mathrm{Opt}(I)$.

If there exists a set of three small items in I_σ such that they arrive as three consecutive items, we call that set to be an S-*triplet*. We call a bin to be a k-bin if it contains exactly k items, for $k \in \{1, 2, 3\}$. We sometimes refer to a bin by mentioning its contents more specifically as follows: An MS-bin is a 2-bin which contains a medium item and a small item. Similarly, an SSS-bin is a 3-bin which contains three small items. Likewise, we can define an M-bin, S-bin,

MM-bin, SS-bin, MMS-bin, and MSS-bin.

Since the item sizes lie in $(1/4, 1/2]$, any bin in the optimal packing contains at most three items. For the same reason, in the packing by Best-Fit, every bin (with one possible exception) contains at least two items. This trivially shows that the ECR of Best-Fit is at most 3/2. To break the barrier of 3/2, we use the following observations.

- Any 3-bin must contain a small item.
- So, if the optimal solution contains a lot of 3-bins, then it means that the input set contains a lot of small items.

We will prove that if there exist many small items in the input, then with high probability, in a random permutation of the input, there exist many disjoint S-triplets.

Claim 3.19. *Let m be the number of small items in the input set I, and let X_σ denote the maximum number of mutually disjoint S-triplets in I_σ. Suppose $m \geq cn$ where $c = 0.00033$, then the following statements hold true:*

1. $\mathbb{E}\left[X_\sigma\right] \geq m^3/(3n^2) \geq c^3 n/3$.
2. $X_\sigma \geq c^3 n/3 - o(n)$ *with high probability.*

We defer the proof of the above claim to Section 3.4.2.1. Then, we prove that Best-Fit packs at least one small item from an S-triplet in a 3-bin or in an SS-bin.

Claim 3.20. *Let $\{S_1, S_2, S_3\}$ be an S-triplet in I such that S_3 follows S_2 which in turn follows S_1. Then, in the final packing of Best-Fit of I, at least one of S_1, S_2, S_3 is packed in a 3-bin or in an SS-bin.* □

Proof. There are three cases we need to consider.

- If S_1 is going to be packed in a 2-bin. Then after packing S_1, this bin becomes a 3-bin and hence the claim holds.
- Suppose S_1 is going to be packed in a 1-bin B. Then just before the arrival of S_1, B must have been the only 1-bin and hence, after packing S_1, each bin is either a 2-bin or a 3-bin.
  - Now, if one of S_2, S_3 is packed into a 2-bin, then the bin becomes a 3-bin and the claim follows.
  - Otherwise, both S_2, S_3 are packed into a single new SS-bin and hence the claim follows.

- Suppose S_1 is packed in a new bin. Then just prior to the arrival of S_2, every bin is either a 2-bin or 3-bin except the bin containing S_1. Thus, S_2 is either packed in a 2-bin (thus becoming a 3-bin) or packed in the bin containing S_1, resulting in an SS-bin.

□

But the number of SS-bins in the final packing of Best-Fit can be at most one. So, we obtain that the number of 3-bins in the Best-Fit packing is significant. With these arguments, we can prove Theorem 3.3 follows.

Proof of Theorem 3.3. Consider the final Best-Fit packing of I_σ. Let X_3 be the number of 3-bins. The remaining $(n - 3X_3)$ items are packed in 2-bins and at most one 1-bin. Therefore,

$$\mathrm{BF}(I_\sigma) \le X_3 + \frac{n-3X_3}{2} + 1 = \frac{n-X_3}{2} + 1 \le \frac{3}{2}\left(1 - \frac{X_3}{n}\right)\mathrm{Opt}(I_\sigma) + 1 \tag{3.22}$$

The last inequality follows from the fact that any bin in the optimal solution can accommodate at most three items. Let $z_1(\le 1)$, z_2 and z_3 be the number of 1-bins, 2-bins and 3-bins in the optimal packing of I_σ, respectively. Then, $\mathrm{Opt}(I_\sigma) = z_1 + z_2 + z_3$ and $n = z_1 + 2z_2 + 3z_3$. Note that any two items can fit in a bin. Therefore,

$$\mathrm{BF}(I_\sigma) \le \frac{n+1}{2} = \frac{z_1 + 2z_2 + 3z_3 + 1}{2} \le \frac{3\mathrm{Opt}(I_\sigma) - z_2 - 2z_1 + 1}{2} \le \left(\frac{3}{2} - \frac{\mu}{2}\right)\mathrm{Opt}(I_\sigma) + 1 \tag{3.23}$$

where $\mu := z_2/\mathrm{Opt}(I_\sigma)$ is the fraction of 2-bins in the optimal solution.

When μ is close to one (say $\mu > 0.999$), we already obtain a competitive ratio very close to 1 due to the above inequality. When $\mu \le 0.999$, then m, the number of small items in the input list is at least $0.00033n$. This is because z_3, the number of 3-bins in the optimal solution, is at least $0.001\mathrm{Opt}(I_\sigma) - 1 \ge 0.001(n/3) - 1 \ge 0.00033n$ and any 3-bin must contain at least one small item.

Hence, assuming $\mu \le 0.999$, let us analyze the Best-Fit packing of I_σ in a different way. Let n_s be the number of small items. Since a 3-bin contains at least one small item, we know that, $n_s \ge z_3 \ge (1-\mu)\mathrm{Opt}(I_\sigma) - z_1$. Let X_σ be the random variable which denotes the maximum number of S-triplets in the input sequence. By Claim 3.19,

$$X_\sigma \ge \frac{((1-\mu)\mathrm{Opt}(I_\sigma) - z_1)^3}{3n^2} - o(n) \approx \frac{(1-\mu)^3\mathrm{Opt}(I_\sigma)^3}{3n^2} - o(n) \text{ w.h.p., as } z_1 \le 1.$$

In the Best-Fit solution, the number of SS-bins can at most be one. Hence, by Claim 3.20,

$$X_3 \geq \frac{X_\sigma}{3} - 1 \geq \frac{(1-\mu)^3 \mathrm{Opt}(I_\sigma)^3}{9n^2} - o(n) \geq \frac{(1-\mu)^3 n}{243} - o(n) \text{ w.h.p.}$$

The last inequality follows from the fact that $\mathrm{Opt}(I_\sigma) \geq n/3$. Using Eq. (3.22), we get,

$$\begin{aligned} \mathrm{BF}(I_\sigma) &\leq \frac{3}{2}\left(1 + \frac{o(n)}{n} - \frac{(1-\mu)^3}{243}\right)\mathrm{Opt}(I_\sigma) + 1 \\ &\leq \frac{3}{2}\left(1 - \frac{(1-\mu)^3}{243}\right)\mathrm{Opt}(I_\sigma) + o(\mathrm{Opt}(I_\sigma)) \text{ w.h.p.} \end{aligned} \tag{3.24}$$

Eqs. (3.23) and (3.24) are the result of two different ways of analyzing the same algorithm. Hence, the expected competitive ratio is given by $\min\left\{\frac{3}{2} - \frac{\mu}{2}, \frac{3}{2} - \frac{(1-\mu)^3}{162}\right\}$. When both the above quantities are equal, we obtain the worst-case competitive ratio. This happens when $81\mu = (1-\mu)^3$, i.e., $\mu \approx 0.01191$; the approximation ratio, in this case, is roughly 1.494045.

We just proved that with high probability, say $p = 1 - o(1)$, $\mathrm{BF}(I) \leq 1.494045\mathrm{Opt}(I) + o(\mathrm{Opt}(I))$. The fact that $\mathrm{BF}(I) \leq 1.7\mathrm{Opt}(I) + 2$ always holds due to [JDU+74]. Combining both these, $\mathbb{E}[\mathrm{BF}(I)] \leq p(1.494045\mathbb{E}[\mathrm{Opt}(I)] + o(\mathbb{E}[\mathrm{Opt}(I)])) + (1-p)(1.7\mathbb{E}[\mathrm{Opt}(I)] + 2)$. Thus we obtain that $\mathbb{E}[\mathrm{Opt}(I)] \leq 1.494045\mathbb{E}[\mathrm{Opt}(I)] + o(\mathbb{E}[\mathrm{Opt}(I)])$.

This completes the proof of Theorem 3.3. □

3.4.2.1 Deferred Proof of Claim 3.19

One can construct a random permutation of the input list I by first placing the small items in a random order and then inserting the remaining items among the small items randomly.

After placing the small items, we have $m+1$ gaps to place the remaining items as shown below in the form of empty squares.

$$\square \underbrace{S\square S\square S}_{\text{Triplet } T_1} \square \underbrace{S\square S\square S}_{\text{Triplet } T_2} \square \underbrace{S\square S\square S}_{\text{Triplet } T_3} \cdots \underbrace{S\square S\square S}_{\text{Triplet } T_{m/3}} \square$$

As shown above, we name the S-triplets as $T_1, T_2, \ldots, T_{m/3}$ (Strictly speaking, the number of S-triplets should be $\lfloor m/3 \rfloor$, but we relax this since we are only interested in the asymptotic case and $m \geq cn$). Let us start inserting the non-small items into these empty squares. Consider any S-triplet T_i. The probability that T_i continues to be an S-triplet after inserting the first non-small item is $(m-1)/(m+1)$. The first non-small item thus occupies one of the squares, but in this process, it creates two new squares on either side of itself, thereby increasing the net number of squares by one. Hence, the probability that T_i continues to be an S-triplet after inserting the second non-small item (given it remains to be one after inserting the first non-small item) is $m/(m+2)$. We continue this process of inserting the non-small items and after they all have been inserted, let $Y_\sigma^{(i)}$ be the indicator random variable denoting whether T_i is consecutive or not. Then

$$\mathbb{E}\left[Y_\sigma^{(i)}\right] = \frac{m-1}{m+1}\frac{m}{m+2}\cdots\frac{n-2}{n} = \frac{m(m-1)}{n(n-1)}.$$

Let $Y_\sigma = Y_\sigma^{(1)} + Y_\sigma^{(2)} + \cdots + Y_\sigma^{(m/3)}$, by linearity of expectations,

$$\lim_{n,m\to\infty} \mathbb{E}\left[Y_\sigma\right] = \lim_{n,m\to\infty} \frac{m}{3}\frac{m(m-1)}{n(n-1)} \approx \frac{m^3}{3n^2} \geq \frac{c^3 n}{3}.$$

Since $X_\sigma \geq Y_\sigma$, the first part of the claim follows.

To prove the second part of the claim, we will compute $\text{Var}\left[Y_\sigma\right]$ and use Chebyshev's inequality. For any i,

$$\begin{aligned}\text{Var}\left[Y_\sigma^{(i)}\right] &= \mathbb{E}\left[Y_\sigma^{(i)}\right] - \mathbb{E}\left[Y_\sigma^{(i)}\right]^2 \\ &= \frac{m(m-1)}{n(n-1)} - \frac{m^2(m-1)^2}{n^2(n-1)^2} \\ &= \frac{m(m-1)(n-m)(n+m-1)}{n^2(n-1)^2} \\ &\leq 2 \qquad (3.25)\end{aligned}$$

Now, consider any two S-triplets T_j, T_k. The probability that both T_j and T_k remain to be S-triplets after inserting the first non-small item is $(m-3)/(m+1)$. Continuing in this manner, after all the non-small items have been inserted,

$$\mathbb{E}\left[Y_\sigma^{(j)}Y_\sigma^{(k)}\right] = \mathbb{P}\left[Y_\sigma^{(j)} = 1 \wedge Y_\sigma^{(k)} = 1\right] = \frac{m-3}{m+1}\frac{m}{m+2}\cdots\frac{n-4}{n} = \frac{m(m-1)(m-2)(m-3)}{n(n-1)(n-2)(n-3)}$$

The covariance of $Y_\sigma^{(j)}, Y_\sigma^{(k)}$ is given by

$$\begin{aligned}
\operatorname{Cov}\left[Y_\sigma^{(j)}, Y_\sigma^{(k)}\right] &= \mathbb{E}\left[Y_\sigma^{(j)} Y_\sigma^{(k)}\right] - \mathbb{E}\left[Y_\sigma^{(j)}\right]\mathbb{E}\left[Y_\sigma^{(k)}\right] \\
&= \frac{m(m-1)(m-2)(m-3)}{n(n-1)(n-2)(n-3)} - \frac{m^2(m-1)^2}{n^2(n-1)^2} \\
&= \frac{m(n-m)(m-1)(4nm-6m-6n+6)}{n^2(n-1)^2(n-2)(n-3)} \\
&\le \frac{4n^3(n-1)^2}{n^2(n-1)^2(n-2)(n-3)} \\
&= \frac{4n}{(n-2)(n-3)} \\
&\le \frac{6}{n} \qquad (3.26)
\end{aligned}$$

Combining all these, the variance of Y_σ can be calculated as follows

$$\begin{aligned}
\operatorname{Var}\left[Y_\sigma\right] &= \sum_{i=1}^{m/3} \operatorname{Var}\left[Y_\sigma^{(i)}\right] + 2\sum_{1\le j<k\le m/3} \operatorname{Cov}\left[Y_\sigma^{(j)}, Y_\sigma^{(k)}\right] \\
&\le \frac{2m}{3} + \frac{m}{3}\left(\frac{m}{3}-1\right)\frac{6}{n} \qquad \text{(from Eqs. (3.25) and (3.26))} \\
&\le 2n
\end{aligned}$$

Now using Chebyshev's inequality,

$$\begin{aligned}
\mathbb{P}\left[Y_\sigma \le \mathbb{E}\left[Y_\sigma\right] - \left(\mathbb{E}\left[Y_\sigma\right]\right)^{2/3}\right] &\le \mathbb{P}\left[\left|Y_\sigma - \mathbb{E}\left[Y_\sigma\right]\right| \ge \left(\mathbb{E}\left[Y_\sigma\right]\right)^{2/3}\right] \\
&\le \frac{\operatorname{Var}\left[Y_\sigma\right]}{\left(\mathbb{E}\left[Y_\sigma\right]\right)^{4/3}} \\
&\le \frac{2n}{\frac{c^4}{3^{4/3}} n^{4/3}} \\
&= O\left(\frac{1}{n^{1/3}}\right), \text{ as } c \text{ is a constant.}
\end{aligned}$$

Since X_σ is the maximum number of S-triplets, $X_\sigma \ge Y_\sigma \ge c^3 n/3 - o(n)$ with high probability.

3.5 Online Vector Packing Problem under i.i.d. model

In this section, we design an algorithm for d-dimensional online vector packing problem (d-OVP) where d is a positive integer constant. In this entire section, we abbreviate a tuple with d entries as just *tuple*. A tuple Y is represented as $\left(Y^{(1)}, Y^{(2)}, \dots, Y^{(d)}\right)$. We define $Y^{\max}$ to be $\max\left\{Y^{(1)}, Y^{(2)}, \dots, Y^{(d)}\right\}$ In d-OVP, the input set I consists of n tuples $X_1, X_2, \dots, X_n$ which arrive in online fashion; we assume that n is known beforehand. Each $X_i^{(j)}$ is sampled independently from a distribution $\mathcal{D}^{(j)}$. The objective is to partition the n tuples into a minimum number of bins such that for any bin B and any $j \in [d]$, the sum of all the j^{th} entries of all the tuples in B does not exceed one. Formally, we require that for any bin B and any $j \in [d]$,$\sum_{x \in B} x^{(j)} \leq 1$.

3.5.1 Algorithm

Given an offline α-asymptotic approximation algorithm $\mathcal{A}_\alpha$ for bin packing, we obtain a $(d\alpha+\varepsilon)$-competitive algorithm for d-OVP as follows: For every input tuple X_i, we round each $X_i^{(j)}, j \in [d]$ to $X_i^{\max}$. After rounding, since all the tuples have same values in each of the d entries, we can treat each tuple X_i as an one-dimensional item of size $X_i^{\max}$.

It is easy to see that each $X_i^{\max}$ is independently sampled from the same distribution: Let $F^{(j)}$ be the cumulative distribution function (CDF) of $\mathcal{D}^{(j)}$. Then the CDF, F, of $X_i^{\max}$ (for any $i \in [n]$) is given by

$$\begin{aligned} F(y) = \mathbb{P}\left[X_i^{\max} \leq y\right] &= \prod_{j=1}^{d} \mathbb{P}\left[X_i^{(j)} \leq y\right] && \text{(By independence of } X_i^{(j)}\text{s)} \\ &= \prod_{j=1}^{d} F^{(j)}(y) \end{aligned}$$

Hence, the problem at hand reduces to solving an online bin packing problem where items are independently sampled from a distribution whose CDF is given by the function $\prod_{i=1}^{d} F^{(j)}$. So, we can use the algorithm from Section 3.3.

3.5.2 Analysis

Let I denote the vector packing input instance and let $\overline{I}$ denote the rounded up one-dimensional bin packing instance.

Lemma 3.21. *Let* $\mathrm{Opt}_v(I)$ *denote the optimal number of bins used to pack* I*. Then* $\mathrm{Opt}(\overline{I}) \leq d\mathrm{Opt}_v(I)$*.*

Proof. Consider any optimal packing of I. We show how to construct a feasible packing of $\overline{I}$ starting from the optimal packing of I. Consider any bin in the optimal packing of I and let B denote the set of tuples packed inside it. Let $\overline{B}$ denote the rounded-up instance of B. Also, for $j \in [d]$, let $B^{(j)}$ denote the set of tuples whose j^{th} dimension has the largest weight, i.e.,

$$B^{(j)} = \left\{Y \in B : Y^{(j)} = Y^{\max}\right\}$$

If any tuple belongs to $B^{(j)}$ as well as $B^{(k)}$ for some $j \neq k$, we break these ties arbitrarily and assign it to one of $B^{(j)}, B^{(k)}$. For $j \in [d]$, let $\overline{B^{(j)}}$ denote the rounded instance of $B^{(j)}$. We can pack $\overline{B}$ in at most d bins as follows: For any $j \in [d]$, we know that

$$\sum_{Y \in B^{(j)}} Y^{(j)} \leq 1$$

and since for all $Y \in B^{(j)}$, $Y^{\max} = Y^{(j)}$, it follows that

$$\sum_{Z \in \overline{B^{(j)}}} Z \leq 1$$

Hence, we can pack every $\overline{B^{(j)}}$ in one bin and the lemma follows. □

Now we are ready to prove our main theorem of this section.

Theorem 3.22. *For any $\varepsilon > 0$ and a given polynomial-time α-approximation algorithm for online bin packing, we can obtain a polynomial-time algorithm for d-OVP with an asymptotic approximation ratio of $(\alpha d + \varepsilon d)$.*

Proof. Let us denote our present algorithm by $\texttt{Alg}_v$. Then we get:

$$\begin{aligned} \texttt{Alg}_v(I) &= \texttt{Alg}(\overline{I}) && (3.27)\\ &\leq (\alpha + \varepsilon)\text{Opt}(\overline{I}) + o\left(\text{Opt}(\overline{I})\right) && (3.28)\\ &\leq (\alpha d + \varepsilon d)\text{Opt}_v(I) + o(\text{Opt}_v(I)). && (3.29)\end{aligned}$$

Here, Equation (3.27) follows from the property of $\texttt{Alg}_v$, (3.28) follows from the results of Section 3.3, and (3.29) follows from Lemma 3.21. This concludes the proof. □

Chapter 4

3-D Knapsack Problem and its Variants

In this chapter, we will discuss some improved approximation algorithms for the 3-D Knapsack problem. The main results are:

A simpler $(7 + \varepsilon)$ approximation algorithm (without rotations). One of the main results of [DHJ+07] is a $(7 + \varepsilon)$ approximation algorithm for the case without rotations. Here, we present an alternate and much simpler algorithm which achieves the same approximation ratio.

A $(31/7+\varepsilon)$ approximation algorithm (with rotations). Another main result of [DHJ+07] is a $(5+\varepsilon)$ approximate algorithm for the case with rotations. We improve upon their result and give an algorithm with approximation factor $(31/7 + \varepsilon) < 4.5$. First, we give an alternate and simpler $(5 + \varepsilon)$ approximation algorithm. Then we use some technical adaptations to improve this ratio to $(31/7 + \varepsilon)$.

A $(3+\varepsilon)$ approximation algorithm (with rotations) to maximize the volume packed. Finally, we consider the special case where each item has a profit equal to its volume. When rotations are allowed, we give an algorithm which has approximation ratio $\approx (3 + \varepsilon)$.

While proving the first two results, we obtain the following intermediate results.

- A $(6 + \varepsilon)$ approximation algorithm for the cardinality case without rotations.
- A $(24/7 + \varepsilon)$ approximation algorithm for the cardinality case with rotations.

4.1 Notations and Preliminaries

In the entire chapter, a "rectangle" is a cuboid in two dimensions and an "item" is a cuboid in three dimensions. We assume that the knapsack given is just a unit cube. Note that this assumption is without loss of generality if rotations are not allowed.

Let I be the input set of n items where each item $i \in I$ can be represented as (ℓ_i, b_i, h_i). We call the length of an item i in the first (resp. second, third) dimension to be the length (resp. breadth, height) of the item. We denote the profit of i by $p(i)$, and its volume as $\mathrm{vol}(i) := \ell_i b_i h_i$. We define the profit density of i as $p(i)/\mathrm{vol}(i)$ (assume without loss of generality that every item has non-zero volume).

We define *base area* of an item i as $\ell_i b_i$. For any subset of items $J \subseteq I$, denote the total profit of J by $p(J)$ and total volume of J by $\mathrm{vol}(J)$.

4.1.1 Steinberg's Algorithm

Steinberg [Ste97] gave the following lemma to pack a set of rectangles in a square. It has been used extensively in many important packing papers like [JZ04, HJPvS14]. We too will use this lemma in this chapter.

Lemma 4.1 (Steinberg's Lemma without Rotations). *Let J be a set of rectangles where each rectangle $j \in J$ is represented as (ℓ_j, b_j) where ℓ_j is the length of the rectangle and b_j is the breadth of the rectangle. Suppose the total area of rectangles in J is at most $1/2$ and the following condition holds true:*

$$\left(\forall j \in J, \ell_j \leq \frac{1}{2}\right) \vee \left(\forall j \in J, b_j \leq \frac{1}{2}\right)$$

Then the entire set J can be packed into a unit square.

The following is similar to the above lemma, except that rotations are not allowed.

Lemma 4.2 (Steinberg's Lemma with Rotations). *Let J be a set of rectangles such that their total area is at most $1/2$ and such that for no item $j \in J$, both length and breadth are greater than $1/2$. Then, if rotations are allowed, the entire set J can be packed in a unit square.*

Note that Lemma 4.2 follows from Lemma 4.1 since none of the items have both breadth and length more than $1/2$. Hence, by rotating, we can make sure that either the length of every item is at most $1/2$ or the breadth of every item is at most $1/2$.

4.1.2 Packing Large Items

Here, we will discuss how to pack items that are larger than a constant δ in all the three dimensions. Suppose each input item i has $\ell_i \geq \delta, b_i \geq \delta, h_i \geq \delta$. Then we can solve the knapsack problem (exactly) optimally. Clearly, the number of items in the optimal solution can be at most $1/\delta^3$. Hence, we can guess this optimal set in time at most

$$\frac{1}{\delta^3}\binom{n}{\lfloor 1/\delta^3 \rfloor} = O_\delta\left(n^{1/\delta^3}\right)$$

which is just polynomial time in n.

Once we guess the optimal set, we can find a packing of the optimal set in $O_\delta(1)$ time. This is because the optimal set has constant number of items.

Lemma 4.3. *If every item has each of its sides to be at least δ, where $\delta > 0$ is a constant, then we can find an optimal packing in polynomial time (irrespective of whether rotations are allowed or not).*

4.2 A simpler $(7+\varepsilon)$-approximation Algorithm (without Rotations)

In this section we will discuss a simpler $(7+\varepsilon)$-approximation (compared to [DHJ+07]) algorithm for 3-D Knapsack where rotations are not allowed.

First we will discuss a $(6+\varepsilon)$ approximation algorithm for the cardinality case, where each item has unit profit. We will see that this algorithm can be generalized to a $(7+\varepsilon)$ approximate algorithm for the general case.

4.2.1 Algorithm for the Cardinality case

Let $\varepsilon \in (0, 1)$ be an accuracy parameter. We first partition the input set into four categories as follows. Let $\delta := \varepsilon/40$.

- L: Items with all the three dimensions more than δ.
- A: Items in $I - L$ with height at most δ.
- B: Items in $I - (L \cup A)$ with length at most δ.
- C: Items in $I - (L \cup A \cup B)$; note that these have breadth at most δ.

We call an item from set L to be *large*. Let Opt denote an optimal packing. We can safely assume that the number of items in Opt is at least $1/\delta^4$. Otherwise, we can find an optimal packing in polynomial time similar to Section 4.1.2. Since the number of large items that can fit in the knapsack is at most $(1/\delta)^3$, by losing at most $\delta p(\text{Opt})$ profit, we can assume that there are no large items in the optimal packing. Let A_{Opt} (resp. $B_{\text{Opt}}, C_{\text{Opt}}$), denote the items of A (resp. B, C) that are in the optimal packing Opt. Therefore

$$p(A_{\text{Opt}} \cup B_{\text{Opt}} \cup C_{\text{Opt}}) \geq (1-\delta)p(\text{Opt})$$

Now we will describe our algorithm. The main idea is to pack subsets of each of A, B, C into three bins separately and pick the most profitable of the three bins. We will design an algorithm to pack a fairly profitable subset of A. The procedures to pack subsets of B, C are similar.

Let A_{big} be the set of items in A with both length and breadth greater than $1/2$. Let $A_{\text{long}} \subseteq A - A_{\text{big}}$ be the set of items whose length is greater than $1/2$ and let $A_{\text{rem}} = A - (A_{\text{big}} \cup A_{\text{long}})$. Note that every item in A_{long} has breadth at most $1/2$ and every item in A_{rem} has length at most $1/2$.

Suppose the list A is sorted in decreasing order of profit density (or, in other words, increasing order of volumes). Select the largest prefix A' of A such that $\text{vol}(A') \leq 1/4 - \delta$.

Remark 4.4. *If $A' \neq A$, then $\text{vol}(A') > 1/4 - 2\delta$ since each item in A has volume at most δ.*

We claim that the entire set A' can be packed in a knapsack and we give an algorithm to do this. Let $A'_{\text{big}} = A' \cap A_{\text{big}}$ and $A'_{\text{long}} = A' \cap A_{\text{long}}$ and $A'_{\text{rem}} = A' \cap A_{\text{rem}}$.

First we pack A'_{big} in the knapsack by just stacking them one on top of each other.

Then we pack A'_{long} on top of the packing of A'_{big} in the following way. Assume that the items in A'_{long} are sorted in decreasing order of their heights.

1. Select the largest prefix P of A'_{long} whose total base area is at most $1/2$.
2. Pack this set P in a layer using Steinberg's algorithm (Lemma 4.1) and redefine $A'_{\text{long}} \leftarrow A'_{\text{long}} - P$.
3. If $A'_{\text{long}} \neq \phi$ go to step 1.
4. Stack up all the layers on top of each other.

We make an important remark here.

Remark 4.5. *The total base area of items in any two consecutive layers is more than $1/2$.*

We pack A'_{rem} into layers using the same procedure as that used to pack A'_{long} and stack these layers on top of the packing of $A'_{\text{big}} \cup A'_{\text{long}}$. We will now prove that the layers thus stacked up do not cross the height of the knapsack.

Let k_{big} be the number of layers used to pack A'_{big} and let k_{long} be the number of layers used to pack A'_{long} and let k_{rem} be the number of layers used to pack A'_{rem}. Let $H_1, H_2, \ldots, H_{k_{\text{big}}}$ be the heights of the layers used to pack A'_{big} and let $X_1, X_2, \ldots, X_{k_{\text{long}}}$ be the heights of the layers used to pack A'_{long} and let $Y_1, Y_2, \ldots, Y_{k_{\text{rem}}}$ be the heights of the layers used to pack A'_{rem} (we will assume that $k_{\text{long}}, k_{\text{rem}}$ are even. It doesn't make much of a difference if they are odd as the height of a layer is at most δ). We know that in each layer used to pack A'_{big}, one item of type A is packed and it has a base area of at least $1/4$. So, we can bound the total height of the packing of A'_{big} as

$$H_1 + H_2 + \cdots + H_{k_{\text{big}}} < 4\,\text{vol}(A'_{\text{big}}) \tag{4.1}$$

Now let's bound the height of packing of A'_{long}. For the j^{th} layer in the packing of A'_{long}, let A_j denote the sum of base areas of all the items in the j^{th} layer and let x_j denote the height of the shortest item in the j^{th} layer. Note that X_j equals the height of the tallest item in the j^{th} layer. We have the following inequality.

$$\begin{aligned}
\text{vol}(A'_{\text{long}}) &\geq A_1x_1 + A_2x_2 + \cdots + A_{k_{\text{long}}}x_{k_{\text{long}}} \\
&\geq (A_1 + A_2)x_2 + (A_3 + A_4)x_4 + \cdots + (A_{k_{\text{long}}-1} + A_{k_{\text{long}}})x_{k_{\text{long}}} \\
&\qquad\qquad \text{(since } x_1, x_2, \ldots \text{ would be in decreasing order)} \\
&> \frac{1}{2}(x_2 + x_4 + \cdots + x_{k_{\text{long}}}) \\
&\qquad\qquad \text{(since the base area of any two consecutive layers is } > 1/2 \text{ (Remark 4.5))} \\
&\geq \frac{1}{2}(X_1 + X_3 + \cdots + X_{k_{\text{long}}-1} - X_1) \qquad \text{(since } x_j \geq X_{j+1} \text{ for all } j \in [k_{\text{long}} - 1]) \\
&\geq \frac{1}{2}\left(\frac{X_1 + X_2 + \cdots + X_{k_{\text{long}}}}{2} - X_1\right) \qquad \text{(since } X_j \geq X_{j+1} \text{ for all } j \in [k_{\text{long}} - 1]) \\
&> \frac{1}{4}(X_1 + X_2 + \cdots + X_{k_{\text{long}}}) - \frac{\delta}{2} \quad \text{(since height of each item in } A'_{\text{long}} \text{ is less than } \delta)
\end{aligned}$$

Therefore, we have that

$$X_1 + X_2 + \cdots + X_{k_{\text{long}}} < 4\,\text{vol}(A'_{\text{long}}) + 2\delta \tag{4.2}$$

Similarly, we also obtain that

$$Y_1 + Y_2 + \cdots + Y_{k_{\text{rem}}} < 4\,\text{vol}(A'_{\text{rem}}) + 2\delta \tag{4.3}$$

Combining Eqs. (4.1) to (4.3), we have

$$H_1 + \cdots + H_{k_{\text{big}}} + X_1 + \cdots + X_{k_{\text{long}}} + Y_1 + \cdots + Y_{k_{\text{rem}}} < 4\,\text{vol}(A') + 4\delta \leq 1$$

Therefore, the entire packing of A' fits in the knapsack.

Since A' is the most profitable subset of A whose volume is equal to vol(A'), we have that

$$p(A') \geq \min\left\{p(A_{\text{Opt}}), \frac{\text{vol}(A')}{\text{vol}(A_{\text{Opt}})}p(A_{\text{Opt}})\right\} \tag{4.4}$$

Using similar ideas, we obtain algorithms to pack $B' \subseteq B, C' \subseteq C$ into separate bins such that

$$p(B') \geq \min\left\{p(B_{\text{Opt}}), \frac{\text{vol}(B')}{\text{vol}(B_{\text{Opt}})}p(B_{\text{Opt}})\right\} \tag{4.5}$$

$$p(C') \geq \min\left\{p(C_{\text{Opt}}), \frac{\text{vol}(C')}{\text{vol}(C_{\text{Opt}})}p(C_{\text{Opt}})\right\} \tag{4.6}$$

4.2.1.1 Analysis

For the purpose of analysis, by appropriate scaling of the profit of items, let's assume that $p(A_{\text{Opt}} \cup B_{\text{Opt}} \cup C_{\text{Opt}}) = 1$. We will show that our algorithm is an $6 + \varepsilon$ approximation algorithm. We will assume without loss of generality that

$$\frac{p(A_{\text{Opt}})}{\text{vol}(A_{\text{Opt}})} \geq \frac{p(B_{\text{Opt}})}{\text{vol}(B_{\text{Opt}})} \geq \frac{p(C_{\text{Opt}})}{\text{vol}(C_{\text{Opt}})}$$

We hence have that

$$\frac{p(A_{\text{Opt}})}{\text{vol}(A_{\text{Opt}})} \geq \frac{p(A_{\text{Opt}}) + p(B_{\text{Opt}}) + p(C_{\text{Opt}})}{\text{vol}(A_{\text{Opt}}) + \text{vol}(B_{\text{Opt}}) + \text{vol}(C_{\text{Opt}})} \geq 1$$

If $p(A_{\text{Opt}}) \geq 1/6$, then from Eq. (4.4), we obtain that

$$\begin{aligned} p(A') \geq 1/6 &= \frac{p(A_{\text{Opt}}) + p(B_{\text{Opt}}) + p(C_{\text{Opt}})}{6} \\ &\geq \left(\frac{1}{6} - \frac{\delta}{6}\right) p(\text{Opt}) \\ &\geq \left(\frac{1}{6} - \varepsilon\right) p(\text{Opt}) \end{aligned}$$

So, assume from now on that $p(A_{\text{Opt}}) < 1/6$. This implies that

$$\frac{p(B_{\text{Opt}})}{\text{vol}(B_{\text{Opt}})} \geq \frac{p(B_{\text{Opt}}) + p(C_{\text{Opt}})}{\text{vol}(B_{\text{Opt}}) + \text{vol}(C_{\text{Opt}})} \geq 5/6 \qquad (\text{since } \text{vol}(B_{\text{Opt}}) + \text{vol}(C_{\text{Opt}}) \leq 1)$$

If $p(B_{\text{Opt}}) \geq 1/6$ and $\text{vol}(B_{\text{Opt}}) \geq 1/4 - 2\delta$, then by Remark 4.4 we know that $\text{vol}(B')$ will at least be $1/4 - 2\delta$. Then from Eq. (4.5),

$$\begin{aligned} p(B') &\geq \min\left\{p(B_{\text{Opt}}), \text{vol}(B')\frac{p(B_{\text{Opt}})}{\text{vol}(B_{\text{Opt}})}\right\} \\ &\geq \min\{1/6, (1/4 - 2\delta)(5/6)\} \\ &= 1/6 \qquad\qquad (\text{for } \delta < 1/40) \\ &\geq \left(\frac{1}{6} - \varepsilon\right) p(\text{Opt}) \end{aligned}$$

If $p(B_{\text{Opt}}) \geq 1/6$ and $\text{vol}(B_{\text{Opt}}) < 1/4 - 2\delta$, then we know that $\text{vol}(B') \geq \text{vol}(B_{\text{Opt}})$. Hence

$$\begin{aligned} p(B') &\geq \min\left\{p(B_{\text{Opt}}), \frac{\text{vol}(B')}{\text{vol}(B_{\text{Opt}})}p(B_{\text{Opt}})\right\} \\ &= p(B_{\text{Opt}}) \\ &\geq 1/6 \\ &\geq \left(\frac{1}{6} - \varepsilon\right) p(\text{Opt}) \end{aligned}$$

So, now assume that $p(B_{\text{Opt}})$ is also less than $1/6$. Then we have that

$$\frac{p(C_{\text{Opt}})}{\text{vol}(C_{\text{Opt}})} = \frac{1 - p(A_{\text{Opt}}) - p(B_{\text{Opt}})}{\text{vol}(C_{\text{Opt}})} > 2/3$$

If $\text{vol}(C_{\text{Opt}}) \geq 1/4 - 2\delta$, then using similar arguments as above,

$$\begin{aligned} p(C') &\geq \min\left\{\frac{2}{3}, \frac{2}{3}\left(\frac{1}{4} - 2\delta\right)\right\} \\ &= \frac{1}{6} - \frac{4\delta}{3} \\ &\geq \left(\frac{1}{6} - \frac{4\delta}{3}\right)(1-\delta)p(\text{Opt}) \\ &\geq \left(\frac{1}{6} - \frac{3\delta}{2}\right)p(\text{Opt}) \\ &\geq \left(\frac{1}{6} - \varepsilon\right)p(\text{Opt}) \end{aligned}$$

On the other hand, if $\text{vol}(C_{\text{Opt}}) < 1/4 - 2\delta$, we similarly obtain that $p(C') \geq 2/3$.

Combining all the above arguments, we obtain that

$$\max\{p(A'), p(B'), p(C')\} \geq \left(\frac{1}{6} - \varepsilon\right)p(\text{Opt})$$

Here, we make a useful remark that helps us extend this algorithm to the general case.

Remark 4.6. *In the entire algorithm for packing A', B', C' we never used the fact that we are in the cardinality case! We only used volume arguments. Hence, this algorithm also serves as a $(6+\varepsilon)$ approximate algorithm in the case when none of the items are large (but can have arbitrary profits.)*

4.2.2 Extending to General Case

The above algorithm for the cardinality case can be easily extended to a $(7+\varepsilon)$ approximation algorithm in the general case. Note that we can not ignore the set L, the set of large items, as they can have a huge profit.

Again, let Opt denote the optimal packing. Let $L_{\text{Opt}} := L \cap \text{Opt}$. We can pack a profit of at least $p(L_{\text{Opt}})$ in a knapsack using Lemma 4.3.

If $p(L_{\text{Opt}}) \geq (1/7)p(\text{Opt})$, then the packing of large items suffices for a $(7+\varepsilon)$ approximation algorithm. So, let's assume that $p(L_{\text{Opt}}) < (1/7)p(\text{Opt})$. Then

$$p(A_{\text{Opt}} \cup B_{\text{Opt}} \cup C_{\text{Opt}}) \geq \frac{6}{7}p(\text{Opt})$$

Using the above algorithm, (in Remark 4.6 we noted that it works for non-large items with general profits), we can get a packing with profit at least

$$\left(\frac{1}{6}-\varepsilon\right)p(A_{\text{Opt}}\cup B_{\text{Opt}}\cup C_{\text{Opt}})\geq\left(\frac{1}{7}-\frac{6\varepsilon}{7}\right)p(\text{Opt})$$

Scaling ε appropriately before the start of the algorithm, we obtain the desired approximation ratio of $(7+\varepsilon)$.

4.3 A simpler $(5+\varepsilon)$-approximation Algorithm (with Rotations)

In this section we will discuss a simpler $(5+\varepsilon)$-approximation (compared to [DHJ+07]) algorithm for 3-D Knapsack where rotations are allowed.

Similar to the section on the $(7+\varepsilon)$ approximate algorithm for the case without rotations, we first devise a $(4+\varepsilon)$ approximation algorithm for the cardinality case, where each item has unit profit. Then, we will extend it to a $(5+\varepsilon)$ approximate algorithm for the general case.

Since rotations are allowed, we can assume that for any item i, $\ell_i \geq b_i \geq h_i$.

4.3.1 Algorithm for the Cardinality Case

We first partition the input set into three sets as follows. Let $\delta := \varepsilon/10$.

- (L) Large items: Items of the form $(\geq \delta, \geq \delta, \geq \delta)$.
- (A) Items of the form $(\geq 1/2, \geq 1/2, < \delta)$.
- (B) All other items (these items have height less than δ and at least one of length and breadth less than $1/2$).

Let Opt denote the optimal packing. Like in the case without rotations, we can again assume that the number of items in Opt is at least $(1/\delta)^4$. Let A_{Opt} (resp. B_{Opt}), denote the items in A (resp. B) that are in the optimal solution. Hence

$$p(A_{\text{Opt}}\cup B_{\text{Opt}})\geq(1-\delta)p(\text{Opt})$$

Let $C = A \cup B$ and suppose the list C is sorted in decreasing order of profit density (or, in other words, increasing order of volumes). Select the largest prefix C' of C such that $\text{vol}(C') \leq 1/4 - \delta/2$.

Remark 4.7. *If $C' \neq C$, then $\text{vol}(C') > \frac{1}{4} - \frac{3\delta}{2}$ since each item in C has volume at most δ.*

We claim that the entire set C' can be packed in a knapsack and we give an algorithm to do this. Let $C'_A := C' \cap A$ and $C'_B = C' \cap B$.

First we pack C'_A in the knapsack by just stacking them one on top of each other.

Then we pack C'_B on top of the packing of C'_A in the following way. Assume that the items in C'_B are sorted in decreasing order of their heights.

1. Select the largest prefix P of C'_B whose total base area is at most $1/2$.
2. Pack this set P in a layer using Steinberg's algorithm (Lemma 4.2) and redefine $C'_B \leftarrow C'_B - P$.
3. If $C'_B \neq \phi$ go to step 1.
4. Stack up all the layers on top of each other.

We will now prove that the layers thus stacked up do not cross the height of the knapsack.

Let k_A be the number of layers used to pack C'_A and let k_B be the number of layers used to pack C'_B. Let $X_1, X_2, \ldots, X_{k_A}$ be the heights of the layers used to pack C'_A and let $H_1, H_2, \ldots, H_{k_B}$ be the heights of the layers used to pack C'_B (we will assume that k_B is even). We know that in each layer used to pack C'_A, one item of type A is packed and it has a base area of at least $1/4$. So, we can bound the total height of the packing of C'_A as

$$X_1 + X_2 + \cdots + X_{k_A} < 4\,\mathrm{vol}(C'_A) \tag{4.7}$$

Now let's bound the height of packing of C'_B. For the j^{th} layer in the packing of C'_B, we need the following notations:

- A_j denotes the base area of all the items in the j^{th} layer.
- h_j denotes the height of the shortest item in the j^{th} layer.

Note that H_j denotes the height of the tallest item in the j^{th} layer. We have the following inequalities.

$$\begin{aligned}
\mathrm{vol}(C'_B) &\geq A_1h_1 + A_2h_2 + \cdots + A_{k_B}h_{k_B} \\
&\geq (A_1 + A_2)h_2 + (A_3 + A_4)h_4 + \cdots + (A_{k_B-1} + A_{k_B})h_{K_B} \\
&\qquad\qquad \text{(since } h_1, h_2, \ldots \text{ would be in decreasing order)} \\
&> \frac{1}{2}(h_2 + h_4 + \cdots + h_{k_B}) \\
&\qquad\qquad \text{(since the base area of any two consecutive layers is } > 1/2) \\
&\geq \frac{1}{2}(H_1 + H_3 + H_5 + \cdots + H_{k_B-1} - H_1) \qquad \text{(since } h_j \geq H_{j+1} \text{ for all } j \in [k_B - 1]) \\
&\geq \frac{1}{2}\left(\frac{H_1 + H_2 + \cdots + H_{k_B}}{2} - H_1\right) \qquad \text{(since } H_j \geq H_{j+1} \text{ for all } j \in [k_B - 1]) \\
&> \frac{1}{4}(H_1 + H_2 + \cdots + H_{k_B}) - \frac{\delta}{2} \qquad \text{(since height of each item in } C'_B \text{ is less than } \delta)
\end{aligned}$$

Therefore, we have that

$$H_1 + H_2 + \cdots + H_{k_B} < 4\,\mathrm{vol}(C'_B) + 2\delta \tag{4.8}$$

Combining Eqs. (4.7) and (4.8), we have

$$X_1 + X_2 + \cdots + X_{k_A} + H_1 + H_2 + \cdots + H_{k_B} < 4\,\mathrm{vol}(C') + 2\delta \leq 1$$

Therefore, the entire packing of C' fits in the knapsack.

Since C' is the most profitable subset of C whose volume is equal to $\mathrm{vol}(C')$, we have that

$$p(C') \geq \frac{\mathrm{vol}(C')}{\mathrm{vol}(A_{\mathrm{Opt}} \cup B_{\mathrm{Opt}})} p(A_{\mathrm{Opt}} \cup B_{\mathrm{Opt}})$$

If $\mathrm{vol}(A_{\mathrm{Opt}} \cup B_{\mathrm{Opt}}) \leq 1/4 - 3\delta/2$, then our algorithm gives the exact optimal profit (minus the profit of large items in the optimal solution which is very small). Otherwise, since $\mathrm{vol}(A_{\mathrm{Opt}} \cup$

$B_{\text{Opt}}) \leq 1$, by Remark 4.7, we obtain that

$$\begin{aligned} p(C') &\geq \left(\frac{1}{4} - \frac{3\delta}{2}\right) p(A_{\text{Opt}} \cup B_{\text{Opt}}) \\ &\geq \left(\frac{1}{4} - \frac{3\delta}{2}\right) (1-\delta) p(\text{Opt}) \\ &\geq \left(\frac{1}{4} - \frac{7}{4}\delta\right) p(\text{Opt}) \\ &\geq \left(\frac{1}{4} - \varepsilon\right) p(\text{Opt}) \end{aligned}$$

4.3.2 Extending to General Case

Similar to the case without rotations, the above algorithm for the cardinality case can be extended to a $(5+\varepsilon)$ approximation algorithm in the general case.

Let Opt denote the optimal packing. Let $L_{\text{Opt}} := L \cap \text{Opt}$. We can pack a profit of at least $p(L_{\text{Opt}})$ in a knapsack using Lemma 4.3.

If $p(L_{\text{Opt}}) \geq (1/5)p(\text{Opt})$, then the packing of large items suffices for a $(5+\varepsilon)$ approximation algorithm. So, let's assume that $p(L_{\text{Opt}}) < (1/5)p(\text{Opt})$. Then

$$p(A_{\text{Opt}} \cup B_{\text{Opt}}) \geq \frac{4}{5} p(\text{Opt})$$

Using the above algorithm (similar to the "no rotations" case, it works for non-large items with general profits since we only use volume arguments), we can get a packing with profit at least

$$\left(\frac{1}{4} - \varepsilon\right) p(A_{\text{Opt}} \cup B_{\text{Opt}}) \geq \left(\frac{1}{5} - \frac{4\varepsilon}{5}\right) p(\text{Opt})$$

Scaling ε appropriately before the start of the algorithm, we obtain the desired approximation ratio of $(5+\varepsilon)$.

4.4 A $(31/7+\varepsilon)$-approximation Algorithm (with Rotations)

In this section, we will discuss a $(31/7+\varepsilon)$-approximation algorithm for the 3-D Knapsack problem with rotations allowed. This beats the current best $(5+\varepsilon)$-approximation algorithm by [DHJ+07].

Since rotations are allowed, we can assume that for any item i, $\ell_i \geq b_i \geq h_i$.

We first partition the input set into three sets as follows. Let $\delta := \varepsilon/40$.

- (L) Large items: Items of the form $(\geq \delta, \geq \delta, \geq \delta)$.
- (A) Items of the form $(\geq 1/2, \geq 1/2, < \delta)$.
- (B) All other items (these items have height less than δ and at least one of length and breadth less than $1/2$). This category is further divided into two sub-categories.
  - (B_ℓ) Items in B with base area at least $1/6$.
  - (B_s) Items in B with base area less than $1/6$.

Before jumping to the general case, we will first discuss a $(24/7 + \varepsilon)$ approximation algorithm for the cardinality case. Then, we will extend this to the general case.

4.4.1 Algorithm for the Cardinality Case

Let $\varepsilon \in (0, 1)$ be an accuracy parameter. Similar to the previous sections, we will assume that the number of items in Opt is at least $1/\delta^4$. Let A_{Opt} (resp. B_{Opt}), denote the items in A (resp. B) that are in the optimal solution. Similar to the previous section, we have that

$$p(A_{\text{Opt}} \cup B_{\text{Opt}}) \geq (1 - \delta)p(\text{Opt})$$

Let $C = A \cup B$ and suppose the list C is sorted in decreasing order of profit density (or, in other words, increasing order of volumes). Select the largest prefix C' of C such that $\text{vol}(C') \leq 7/24 - 2\delta$. Note that if $C' \neq C$, then $\text{vol}(C') > 7/24 - 3\delta$ since each item in C has volume at most δ.

We claim that the entire set C' can be packed in a knapsack and we give an algorithm to do this. Let $A' := C' \cap A$ and $B' = C' \cap B$. Further, let $B'_\ell = C' \cap B_\ell$ and $B'_s = C' \cap B_s$.

Packing the Items in B'. We first pack B'_ℓ in layers and then we pack B'_s in layers and finally, we pack these layers on top of each other.

Note that since each item in B'_ℓ has a base area of at least $1/6$ and has breadth at most $1/2$, we can pack two such items side by side in a layer making sure a base area guarantee of at least $1/3$ in a layer. Thus, the algorithm to pack B'_ℓ is the following. Sort B'_ℓ in the non-increasing order of heights. Pick the first two items and pack them in a layer. Then pick the next two items and pack them in a layer. Stack these layers on top of each other.

On the other hand, each item in B_s has a base area less than $1/6$ and has breadth at most $1/2$. We can devise the following algorithm to pack B_s in layers. First, we order the items in B_s in non-increasing order of heights. Pick the largest prefix whose total base area doesn't exceed 1/2. This prefix can be packed in a layer using Steinberg's algorithm. We then pack the remaining items in layers following the same procedure. We stack these layers on top of each other on the stack obtained in the procedure for packing B'_ℓ. Since each item has base area at most 1/6 we obtain an area guarantee of at least 1/3 in each layer.

Let $v := \mathrm{vol}(B')$. Using analysis similar to Eqs. (4.2) and (4.3) in Section 4.2,, we can prove that the total height h of the thus stacked layers is at most $3v + 2\delta$.

Packing the Items in A'. We now want to pack the items in A' in the remaining space left above the stacked up layers of items in B'. See Fig. 4.1 below on how to pack these items.

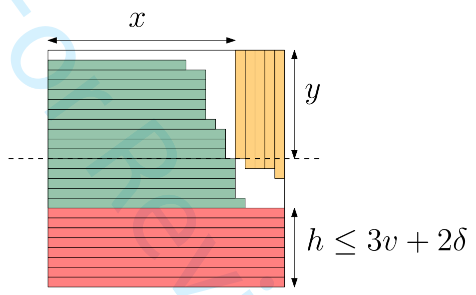


Figure 4.1: (The second dimension which is into the paper or the screen is ignored) The layers of items in B' are indicated in red. After packing B', the items in A' are packed as follows: First, they are ordered in non-increasing order of lengths. Then they are stacked up one on top of each other as much as possible (shown in green). Then they are rotated and packed side-by-side as shown in orange.

4.4.1.1 Analysis of the Algorithm

We will prove that the entire set $A' \cup B'$ can be packed in the manner as shown in Fig. 4.1.

Recall that $v = \mathrm{vol}(B')$ and the height h of the layers of B' is at most $3v + 2\delta$. Suppose $v \geq 1/6 - 2\delta/3$. Then, h can be close to 1/2 and so, we may not be able to pack any orange items as in Fig. 4.1 since each item in A' has length at least 1/2. However, we claim that there will be no orange items left as follows. Assume contradiction. Then the total height of the stack of green items is at least $1 - h - \delta \geq 1 - 3v - 3\delta$. Further, the stack has a length (horizontal dimension) of at least 1/2 and a depth (ignored dimension into the paper/screen) of at least

1/2. Hence, the total volume thus packed in the knapsack is at least

$$\begin{aligned}
v + \frac{1}{4}(1-3v-3\delta) &= \frac{v}{4} + \frac{1}{4}(1-3\delta) \\
&\geq \frac{1}{4}\left(\frac{1}{6} - \frac{2\delta}{3}\right) + \frac{1}{4}(1-3\delta) \\
&= \frac{7}{24} - \frac{11\delta}{12} \\
&> \frac{7}{24} - \delta \\
&= \mathrm{vol}(A' \cup B')
\end{aligned}$$

which is a contradiction.

So, from now on, lets assume that $v < 1/6 - 2\delta/3$. We again need to prove that we can exhaustively pack all the items in $A' \cup B'$ as shown in Fig. 4.1. Suppose not, i.e., we are not able to pack some items because the stack of green items blocks the space when packing the orange items. Let x, y be as shown in Fig. 4.1. We would like to calculate the total area of the green and orange items (not the base area but the area as shown in the figure). The stack of green items below the dotted line has length at least x and height $(1-h-y)$. This amounts to an area of at least $(1-h-y)x \geq (1-3v-y-2\delta)x$. The stack of green items above the dotted line has length at least y and height at least $(y-\delta)$. This amounts to an area of $y(y-\delta)$. The bundle of orange items has area at least $(1-x-\delta)y$. Thus the total area is at least $(1-3u-y-2\delta)x + y(y-\delta) + (1-x-\delta)y$.

Lemma 4.8. *For a very small constant δ and for $x, y \in (1/2, 1]$ and for $v \in (0, 1/6 - 2\delta/3)$,*

$$(1-3v-y-2\delta)x + y(y-\delta) + (1-x-\delta)y > \frac{7}{12} - 2v - 4\delta$$

Proof. Let's first look at the terms containing δ on both the sides. We have that $-2\delta x - \delta y - \delta y > -4\delta$ because both x, y are at most 1. Now, we have to prove that

$$(1-3v-y)x + y^2 + (1-x)y > \frac{7}{12} - 2v$$

Let $f(x,y) = (1-3v-y)x + y^2 + (1-x)y$. We have that

$$\frac{\partial f(x,y)}{\partial y} = 1 + 2(y-x) > 0 \text{ for } (x,y) \in (1/2,1] \times (1/2,1]$$

So, for a fixed x, the value of $f(x,y)$ increases with y. Hence, the minimum of $f(x,y)$ occurs when $y = 1/2$. But when $y = 1/2$, the function simplifies to

$$f(x, y) = \frac{3}{4} - 3vx$$

which attains it minimum value at $x = 1$. Finally, observe that $3/4 - 3v > 7/12 - 2v$ for $v \leq 1/6$ to complete the proof. $\square$

Then, using the above lemma, since each green and orange item has breadth more than 1/2, their total volume is strictly more than $(7/24 - v - 2\delta)$. Thus the total volume packed in Fig. 4.1 is strictly greater than $7/24 - 2\delta$ which is a contradiction.

This proves that our algorithm packs all the items in $A' \cup B'$. Since we selected the most profitable subset of volume at least $(7/24 - 3\delta)$, it follows that our algorithm is a $(7/24 - 3\delta)$-approximation algorithm.

Remark 4.9. *In the entire algorithm for packing A', B' we never used the fact that we are in the cardinality case! Hence, this algorithm also serves as a $(24/7 + \varepsilon)$ approximate algorithm in the case when none of the items are large (but can have arbitrary profits.)*

4.4.2 Extending to the General Case

The above algorithm for the cardinality case can be extended to a $(31/7 + \varepsilon)$ approximation algorithm in the general case.

Let Opt denote the optimal packing. Let $L_{\text{Opt}} := L \cap \text{Opt}$. We can pack a profit of at least $p(L_{\text{Opt}})$ in a knapsack using Lemma 4.3.

If $p(L_{\text{Opt}}) \geq (7/31)p(\text{Opt})$, then the packing of large items suffices for a $(31/7 + \varepsilon)$ approximation algorithm. So, let's assume that $p(L_{\text{Opt}}) < (7/31)p(\text{Opt})$. Then

$$p(A_{\text{Opt}} \cup B_{\text{Opt}}) \geq \frac{24}{31} p(\text{Opt})$$

Using the above algorithm (and by Remark 4.9), we can get a packing with profit at least

$$\left(\frac{7}{24} - \varepsilon\right) p(A_{\text{Opt}} \cup B_{\text{Opt}}) \geq \left(\frac{7}{31} - \frac{24\varepsilon}{31}\right) p(\text{Opt})$$

Scaling ε appropriately before the start of the algorithm, we obtain the desired approximation ratio of $(31/7 + \varepsilon)$.

4.5 A $(3+\varepsilon)$-approximation Algorithm for Maximizing the Packed Volume (with Rotations)

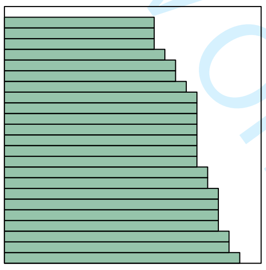
In this section, we will design a $(3+\varepsilon)$-approximation algorithm for the 3-D Knapsack problem with rotations allowed in the special case when the profit of each item is given by its volume.

4.5.1 The Algorithm

We need the following important lemma.

Lemma 4.10. *Let S be a set of rectangles such that each rectangle has one dimension more than $1/2$ and the other dimension at most ε. Suppose that the total area of rectangles in S is at most $3/4 - 3\varepsilon$. Then, if rotations are allowed, we can pack the entire set S in a square of unit dimensions.*

Proof. Let's call the size of a rectangle in the horizontal dimension as length, and the size in the vertical dimension as height. Without loss of generality, we assume that each rectangle in S has height at most ε and length more than $1/2$. We first sort them in decreasing order of lengths. Then we start stacking them one on top of each other to the maximum possible extent as shown in the following figure.



Then we rotate the remaining items and align them vertically in the remaining space to the maximum extent possible. These items are denoted in orange color in the figure below.

We claim that we exhaust the entire set S in this manner. Suppose not. Let the blue item in the below diagram denote the first item that we couldn't pack in this manner. The green rectangle intersecting the dashed line and all the green rectangles below it have a length of at least x. So, the total area of these rectangles is at least $x(1-y)$. Every green rectangle above the blue dashed line have length at least y (since they have larger length compared to the orange rectangles) and their total height is at least $y - 2\varepsilon$. Hence, this amounts to an area of at least $y(y-2\varepsilon)$. Finally, the total area of orange rectangles is at least $y(1-x-\varepsilon)$. Therefore,

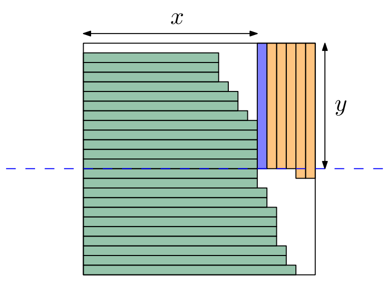


Figure 4.2: Packing rectangles long in one dimension and short in other dimension. The blue rectangle indicates the first rectangle that couldn't be packed without overlapping.

the total packed area of all the rectangles is at least

$$x(1-y) + y(y-2\varepsilon) + y(1-x-\varepsilon) = x + y + y^2 - 2xy - 3y\varepsilon$$

Claim 4.11. *For $x \in (1/2, 1]$ and $y \in (1/2, 1]$, $x + y + y^2 - 2xy > 3/4$*

Proof. When $x = 1$, $x + y + y^2 - 2xy = 1 + y^2 - y > 3/4$ since $y > 1/2$.

Now suppose $x < 1$. Let $f(x, y) = x + y + y^2 - 2xy$. Then

$$\frac{\partial f(x,y)}{\partial x} = 1 - 2y \text{ and } \frac{\partial f(x,y)}{\partial y} = 1 + 2y - 2x$$

In the range $(x, y) \in [1/2, 1) \times [1/2, 1]$, $1 + 2y - 2x > 0$. Hence, for a fixed x and varying y, the function is strictly increasing since the partial derivative of f with respect to y is always positive. So, the minimum of the function $f(x, y)$ in range $[1/2, 1) \times [1/2, 1]$ certainly occurs when $y = 1/2$. But when $y = 1/2$, $f(x, y) = 3/4$. Hence $x + y + y^2 - 2xy > 3/4$ in range $(1/2, 1] \times (1/2, 1]$. □

Thus the total area packed is strictly more than $3/4 - 3\varepsilon$ but this is a contradiction since the total area of rectangles in S is at most $3/4 - 3\varepsilon$. □

Lemma 4.12 ([DHJ+07])**.** *Let J be a set of items such that each has height less than ε and at least one of length, breadth at most $1/2$. Assume rotations are allowed. Then we can pack J in multiple layers such that every layer (except possibly two of them) has a base area of at least $1/3$.*

Proof. Assume without loss of generality that each item has length $\geq$ breadth. Partition J into the following sets.

- J_1: Items with base area less than $1/6$.

- J_2: Items with base area at least $1/6$.

On a square layer of unit dimensions, we can pack a maximal prefix of J_1 whose total area doesn't exceed $1/2$ using Steinberg's algorithm (Lemma 4.2). The base area of this prefix would at least be $1/2 - 1/6 = 1/3$. Thus, using this method, we can pack J_1 in layers such that each layer, except possibly the last, has a base area of at least $1/3$.

Packing J_2 is simple: In each layer we can pack two of them since both have breadth at most $1/2$. Thus the base area of each layer except possibly the last is at least $1/3$. □

Using both the above lemmas, we obtain a $(3 + \varepsilon)$-approximation algorithm for the 3-D knapsack problem with rotations allowed when profit of each item is equal to its volume.

Theorem 4.13. *There exists a* $(3+\varepsilon)$*-approximation algorithm for* 3*-D Knapsack with rotations allowed when profit of each item is equal to its volume.*

Proof. Let I denote the set of input items. Without loss of generality, lets assume that for each item $i \in I$, $\ell_i \geq b_i \geq h_i$. Let us partition I into three sets I_1, I_2, I_3 as follows.

$$I_1 = \{i \in I : \ell_i > \varepsilon, b_i > \varepsilon, h_i > \varepsilon\}$$
$$I_2 = \{i \in I : \ell_i > 1/2, b_i > 1/2, h_i \leq \varepsilon\}$$
$$I_3 = I - \{I_1 \cup I_2\}$$

Let Opt denote an optimal packing of I. We also use the notation Opt to denote the set of items in the optimal packing Opt. Note that $p(\text{Opt}) \leq 1$. Let

$$\text{Opt}_1 = I_1 \cap \text{Opt}, \text{Opt}_2 = I_2 \cap \text{Opt}, \text{Opt}_3 = I_3 \cap \text{Opt}$$

Our algorithm `Alg` packs subsets of each of the sets I_1, I_2, I_3 separately into three bins and takes the maximum profitable bin of these three.

- **Bin 1:** We pack the maximum profitable subset (that can be validly packed) of I_1 since there are at most $1/\varepsilon^3$ items in any valid packing. This gives us a profit of at least $p(\text{Opt}_1)$.
- **Bin 2:** We pack a subset of I_2 using Lemma 4.10 as follows. Ignoring the second dimension, we obtain a set of rectangles with one dimension at most ε and the other dimension more than $1/2$. Thus, these rectangles can be packed either exhaustively or to an extent such that the total area packed is at least $3/4 - 3\varepsilon$. Then, since the length in the first

dimension of each item is more than $1/2$, we obtain that the total volume packed is at least $\min\{3/8 - 3\varepsilon/2, p(I_2)\} \geq \min\{3/8 - 3\varepsilon/2, p(\mathrm{Opt}_2)\}$.

- **Bin 3:** We pack a subset of I_3 using Lemma 4.12 as follows. We first partition the set I_3 into two sets $I_3^{(l)}$ and $I_3^{(s)}$ where $I_3^{(l)}$ contains the items with base area at least $1/6$ and $I_3^{(s)}$ contains the items with base area less than $1/6$. First, we sort the items in $I_3^{(l)}$ in decreasing order of heights and pack them into layers. Then we similarly order $I_3^{(s)}$ in decreasing order of heights and pack them into layers. We first stack up the layers of $I_3^{(l)}$ to the maximum possible extent. If space is still available, we stack the layers of $I_3^{(s)}$ to the maximum possible extent. If we exhaust all the layers, then we must have packed the entire set I_3. Otherwise, using a similar analysis to Eqs. (4.2) and (4.3) in Section 4.2, we can prove that the total volume packed is at least $1/3 - 2\varepsilon$. Thus we pack a profit of at least $\min\{1/3 - 2\varepsilon, p(I_3)\} \geq \min\{1/3 - 2\varepsilon, p(\mathrm{Opt}_3)\}$.

For a small value of ε, it follows that the maximum profitable bin among these three bins has a profit of at least

$$\min\left\{\frac{1}{3} - 2\varepsilon, \frac{p(\mathrm{Opt}_1) + p(\mathrm{Opt}_2) + p(\mathrm{Opt}_3)}{3}\right\}$$

We finally scale the value of ε before the start of the algorithm to obtain a $(3+\varepsilon)$ approximation ratio. □

Chapter 5

Generalized Multidimensional Knapsack

In Sections 1.2.4 and 1.2.5, we introduced the geometric and vector variants of the classical knapsack problem. Both the variants are very well studied. In this chapter, we study a generalization of the knapsack problem with geometric and vector constraints. The input is a set of items which are rectangles (two-dimensional) with vector constraints in d dimensions along with an associated profit. The goal is to pack a subset of items into a given knapsack in a non-overlapping fashion so that the vector constraints are not violated i.e., the sum of vector constraints of all the packed items in any of the d dimensions doesn't exceed one.

We give a $(2 + \varepsilon)$-approximation algorithm for this problem for both the cases of 'with rotations' and 'without rotations' using a technique called corridor decomposition. Note that rotations are allowed only for the geometric dimensions. It isn't all that meaningful to rotate vector dimensions (swapping the weight and the volume of an item doesn't make much sense).

In the process, we also introduce a variant of the Maximum Generalized Assignment Problem (Max-GAP) called Vector-MAX-GAP and design a PTAS for it.

5.1 Preliminaries and Notations

5.1.1 Prior Work

For the knapsack problem, fully polynomial time approximation scheme (FPTAS) exists [IK75, Law77]. The 2-D geometric knapsack problem is known to be strongly NP-Hard i.e., no FPTAS can exist (under the assumption that $\mathsf{P} \neq \mathsf{NP}$). However, it is not known whether the problem admits a PTAS or not. The current best approximation algorithm [GGH+17] for this problem without rotations achieves an approximation ratio of $17/9 + \varepsilon$. If rotations are allowed, the approximation ratio improves to $3/2+\varepsilon$. The d-D vector knapsack problem on the other hand admits a PTAS [FC84], and it is known that for $d \geq 2$, it doesn't admit an FPTAS assuming that $\mathsf{P} \neq \mathsf{NP}$.

5.1.2 Notations and Definitions

We denote the 2-D geometric knapsack problem with d-D vector constraints by $(2,d)$ Knapsack or $(2,d)$ KS in short. Let $\text{poly}(n)$ denote the set of all polynomial and sub-polynomial functions of n.

In the $(2,d)$ Knapsack problem, the input set I consists of n items. Each item is called a $(2,d)$-dimensional item. For any item $i \in I$, let $w(i), h(i), p(i)$ denote the width, height and the profit of the item respectively and let $a(i) = w(i)h(i)$ denote the area of the item. Also, for any item $i \in I$, let $v_j(i)$ denote the weight of the item in the j^{th} dimension. The objective is to pack a maximum profit subset of items $J \subseteq I$ into the knapsack in an axis-parallel, non-overlapping manner such that for any $j \in [d]$,

$$\sum_{i \in J} v_j(i) \leq 1$$

In the whole paper, we will assume without loss of generality that the knapsack is a unit square and all items have width, height and weight constraints not exceeding one. If S is a set of items, let $a(S), p(S)$ denote the total profit and area of the items in S respectively.

Recall the Next-Fit Decreasing Height (NFDH) algorithm introduced in Section 2.4.2. The following lemma will be crucial for our purposes.

Lemma 5.1. *Consider a set of rectangles S with designated profits and a bin of dimensions $W \times H$ and assume that that each item in S has width at most εW and height at most εH. Suppose Opt denotes the optimal profit that can be packed in the bin. Then there exists a polynomial time algorithm that packs at least $(1-2\varepsilon)\text{Opt}$ profit into the bin.*

Proof. Lets first order the rectangles in the decreasing order of their profit/area ratio. Then pick the largest prefix of rectangles T such that $a(T) \leq (1-\varepsilon)^2 WH$. By Lemma 2.5, NFDH must be able to pack all the items in T into the bin. On the other hand, since each rectangle has area at most $\varepsilon^2 WH$, it follows that $a(T) \geq (1-2\varepsilon)WH$. Furthermore, since T contains the highest profit density items, it follows that $p(T) \geq (1-2\varepsilon)\text{Opt}$. □

5.1.3 Organization of the Chapter

In Section 5.2, we will formally define and design a PTAS for the Vector-Max-GAP problem. Section 5.3 is dedicated to designing our $(2+\varepsilon)$ approximation algorithm for the $(2,d)$ Knapsack problem. Sections 5.3.1 and 5.3.2 are dedicated to proving a structural result using corridor decomposition scheme and reduce our problem to what is called a *container packing problem*. In Sections 5.3.3 and 5.3.4 we will discuss how to model the container packing problem as an instance of the Vector-Max-GAP problem. Finally, in Section 5.3.5, we put everything together to obtain the $(2+\varepsilon)$ approximation algorithm for the $(2, d)$ Knapsack problem.

5.2 Vector-Max-GAP problem and a PTAS

The Vector-Max-GAP is a generalization of the well known Maximum Generalized Assignment Problem (Max-GAP). In the Max-GAP problem, we are provided with a set of machines, with designated capacities, and a set of items; an item's size and value depends on the machine to which it is going to be assigned. The objective is to assign a subset of items to machines such that the obtained value is maximized while making sure that no machine's capacity is breached. In the Vector-Max-GAP problem, we additionally have a d-dimensional weight vector associated with every item and a d-dimensional global weight constraint on the whole setup of machines. The objective is to find the maximum value obtainable so that no machine's capacity is breached and the overall weight of items does not cross the global weight constraint.

Formally, let I be a set of n items numbered 1 to n and let M be a set of k machines, where k is a constant. The j^{th} machine has a capacity M_j. Each item $i \in I$ has a size of $s_j(i)$, value of $\text{val}_j(i)$ in the j^{th} machine ($j \in [k]$). Additionally, each item i also has a weight $w_q(i)$ in the q^{th} dimension ($q \in [d]$, d is a constant). Assume that for all $j \in [k]$, $q \in [d]$ and $i \in [n]$, $M_j, w_q(i), s_j(i), \text{val}_j(i)$ are all non-negative.

The objective is to assign a subset of items $J \subseteq I$ to the machines such that for any machine j, the size of all the items assigned to it does not exceed M_j. Further, the total weight of the set J in any dimension $q \in [d]$ must not exceed W_q, which is the global weight constraint of the whole setup in the q^{th} dimension. Respecting these constraints, we would like to maximize the total value of the items in J.

Formally, let J be the subset of items picked and J_j be the items assigned to the j^{th} machine ($j \in [k]$). The assignment is feasible iff the following constraints are satisfied:

$$\forall q \in [d], \sum_{i \in J} w_q(i) \le W_q \qquad \text{(weight constraints)}$$

$$\forall j \in [k], \sum_{i \in J_j} s_j(i) \le M_j \qquad \text{(size constraints)}$$

Let $\overrightarrow{M} = [M_1, M_2, \dots, M_k]$, $\overrightarrow{w}(i) = [w_1(i), w_2(i), \dots, w_d(i)]$, $\overrightarrow{s}(i) = [s_1(i), s_2(i), \dots, s_k(i)]$,

$\overrightarrow{\mathrm{val}}(i) = [\mathrm{val}_1(i), \mathrm{val}_2(i), \ldots, \mathrm{val}_k(i)]$.

Let $\overrightarrow{W} = [W_1, W_2, \ldots, W_d]$, $\overrightarrow{s} = [\overrightarrow{s}(1), \overrightarrow{s}(2), \ldots, \overrightarrow{s}(n)]$, $\overrightarrow{w} = [\overrightarrow{w}(1), \overrightarrow{w}(2), \ldots, \overrightarrow{w}(n)]$, $\overrightarrow{\mathrm{val}} = [\overrightarrow{\mathrm{val}}(1), \overrightarrow{\mathrm{val}}(2), \ldots, \overrightarrow{\mathrm{val}}(n)]$.

An instance of this problem is given by $(I, \overrightarrow{\mathrm{val}}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$. We say that the set of items J is feasible for $(\overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$ iff J can fit in machines of capacity given by $\overrightarrow{M}$ and satisfy the global weight constraint given by $\overrightarrow{W}$ where item sizes and weights are given by $\overrightarrow{s}$ and $\overrightarrow{w}$ respectively.

5.2.1 Dynamic-Programming Algorithm for Integral Input

Consider the case where item sizes and weights are integers. Without loss of generality, we can assume that the capacities of machines, M_j ($j \in [k]$) and weight constraints, W_q ($q \in [d]$), are also integers (otherwise, we can round them down to the closest integers).

Arbitrarily order the items and number them from 1 onwards. Let $\mathrm{VAL}(n, \overrightarrow{M}, \overrightarrow{W})$ be the maximum value obtainable by assigning a subset of the first n items to k machines with capacities given by $\overrightarrow{M}$ respecting the global weight constraint $\overrightarrow{W}$. We can express $\mathrm{VAL}(n, \overrightarrow{M}, \overrightarrow{W})$ as a recurrence.

$$\mathrm{VAL}(n, \overrightarrow{M}, \overrightarrow{W}) = \begin{cases} -\infty & \text{if } \neg(\overrightarrow{W} \geq 0 \wedge \overrightarrow{M} \geq 0) \\ 0 & \text{if } n = 0 \\ \max\left(\begin{array}{l} \mathrm{VAL}(n-1, \overrightarrow{M}, \overrightarrow{W}), \\ \max\limits_{j=1}^{k} \left(\mathrm{val}_j(n) + \mathrm{VAL}\left(n-1, \overrightarrow{M} - \overrightarrow{s}(n) \cdot \overrightarrow{e}_j, \overrightarrow{W} - \overrightarrow{w}(n)\right)\right) \end{array} \right) & \text{else} \end{cases}$$

$\mathrm{VAL}(n, \overrightarrow{M}, \overrightarrow{W})$ can be computed using dynamic programming. We can find the subset of items that gives this much value and it is also easy to ensure that no item assigned to a machine has value 0 in that machine. There are $n \prod_{j=1}^{k}(M_j + 1) \prod_{q=1}^{d}(W_q + 1)$ items in the state space and each iteration takes $\Theta(d + k)$ time. Therefore, time taken by the dynamic programming solution is $\Theta\left(n(d+k) \prod_{j=1}^{k}(M_j + 1) \prod_{q=1}^{d}(W_q + 1)\right)$.

5.2.2 Optimal Solution with Resource Augmentation

Let $\overrightarrow{\mu} = [\mu_1, \mu_2, \ldots, \mu_k]$ and $\overrightarrow{\delta} = [\delta_1, \delta_2, \ldots, \delta_d]$ be vectors whose values will be decided later. For $j \in [k]$, define $s'_j(i) = \lceil s_j(i)/\mu_j \rceil, M'_j = \lfloor M_j/\mu_j \rfloor + n$. For $q \in [d]$, define $w'_q(i) = \lceil w_q(i)/\delta_q \rceil, W'_q = \lfloor W_q/\delta_q \rfloor + n$.

Lemma 5.2. *Let $J \subseteq I$ be feasible for $(\overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$. Then J is also feasible for $(\overrightarrow{s'}, \overrightarrow{w'}, \overrightarrow{M'}, \overrightarrow{W'})$.*

Proof. For any dimension $q \in [d]$, $\sum_{i \in J} w'_q(i) = \sum_{i \in J} \lceil w_q(i)/\delta_q \rceil \le \sum_{i \in J} (\lfloor w_q(i)/\delta_q \rfloor + 1)$.

$$\sum_{i \in J} (\lfloor w_q(i)/\delta_q \rfloor + 1) \le |J| + \left\lfloor (1/\delta_q) \sum_{i \in J} w_q(i) \right\rfloor \le n + \lfloor W_q/\delta_q \rfloor = W'_q$$

Let J_j be the items in J assigned to the j^{th} machine. Then $\sum_{i \in J_j} s'_j(i) = \sum_{i \in J_j} \lceil s_j(i)/\mu_j \rceil$.

$$\sum_{i \in J_j} (\lfloor s_j(i)/\mu_j \rfloor + 1) \le |J_j| + \left\lfloor (1/\mu_j) \sum_{i \in J_j} s_j(i) \right\rfloor \le n + \lfloor M_j/\mu_j \rfloor = M'_j$$ □

Lemma 5.3. *Let J be feasible for $(\overrightarrow{s'}, \overrightarrow{w'}, \overrightarrow{M'}, \overrightarrow{W'})$. Then J is also feasible for $(\overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M} + n\overrightarrow{\mu}, \overrightarrow{W} + n\overrightarrow{\delta})$.*

Proof. For all $q \in [d]$

$$\sum_{i \in J} w_q(i) \le \sum_{i \in J} \delta_q w'_q(i) \le \delta_q W'_q = \delta_q (\lfloor W_q/\delta_q \rfloor + n) \le W_q + n\delta_q$$

Let J_j be the items in J assigned to the j^{th} machine.

$$\sum_{i \in J_j} s_j(i) \le \sum_{i \in J_j} \mu_j s'_j(i) \le \mu_j M'_j = \mu_j (\lfloor M_j/\mu_j \rfloor + n) \le M_j + n\mu_j$$ □

Let $\mu_j = \varepsilon M_j/n$ and $\delta_q = \varepsilon W_q/n$ for all $q \in [d]$ and $j \in [k]$. Let J^* be the optimal solution to $(I, \overrightarrow{\text{val}}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$. Let $\widehat{J}$ be the optimal solution to $(I, \overrightarrow{\text{val}}, \overrightarrow{s'}, \overrightarrow{w'}, \overrightarrow{M'}, \overrightarrow{W'})$. By Lemma 5.2, $\text{val}(\widehat{J}) \ge \text{val}(J^*)$. By Lemma 5.3, $\widehat{J}$ is feasible for $(\overrightarrow{s}, \overrightarrow{w}, (1+\varepsilon)\overrightarrow{M}, (1+\varepsilon)\overrightarrow{W})$. Also, observe that $|M'_j| \le n + M_j/\mu_j = n(1+1/\varepsilon)$ is a polynomial in n. Similarly, $|W'_q| \le n(1+1/\varepsilon)$ and hence the optimal solution to $(I, \overrightarrow{\text{val}}, \overrightarrow{s'}, \overrightarrow{w'}, \overrightarrow{M'}, \overrightarrow{W'})$ can be obtained using the dynamic-programming algorithm in polynomial time. Therefore, the optimal solution to $(I, \overrightarrow{\text{val}}, \overrightarrow{s'}, \overrightarrow{w'}, \overrightarrow{M'}, \overrightarrow{W'})$ can be obtained using the dynamic-programming algorithm in time $\Theta\left((d+k)n^{d+k+1}/\varepsilon^{d+k}\right)$.

Let us define a subroutine $\text{assign-res-aug}_\varepsilon(I, \overrightarrow{\text{val}}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$ which takes as input set I with associated values, $\overrightarrow{\text{val}}$, and gives as output the optimal feasible solution to $(\overrightarrow{s}, \overrightarrow{w}, (1+\varepsilon)\overrightarrow{M}, (1+\varepsilon)\overrightarrow{W})$.

5.2.3 Packing Small Items

5.2.3.1 Trimming

Consider a set I of items where each item has length $s(i)$ and profit $p(i)$ (in this subsection, I is an instance of the knapsack problem instead of the Vector-Max-GAP problem).

Suppose for all $i \in I, s(i) \in (0, \alpha]$ and $s(I) \leq 1 + \beta$ where $\alpha, \beta \in \mathbb{R}^+$ such that $\alpha \leq \beta$. We'll show that there exists an $R \subseteq I$ such that $s(I - R) \leq 1$ and $p(R) < (\beta + \alpha)p(I)$. We call this technique of removing a low-profit subset from I so that it fits in a bin of length 1 trimming.

Arbitrarily order the items and arrange them linearly in a bin of size $1 + \beta$. Let $k = \lfloor 1/(\beta + \alpha) \rfloor$. Create $k+1$ intervals of length β and k intervals of length α. Place β-intervals and α-intervals alternately. They will fit in the bin because $(k+1)\beta + k\alpha = \beta + (\beta+\alpha)\lfloor 1/(\beta + \alpha) \rfloor \leq 1 + \beta$

Number the β-intervals from 0 to k and let S_i be the set of items intersecting the i^{th} β-interval. Note that, all S_i are mutually disjoint. Let $i^* = \arg\min_{i=0}^{k} p(S_i)$.

$$p(S_{i^*}) = \min_{i=0}^{k} p(S_i) \leq \frac{1}{k+1} \sum_{i=0}^{k} p(S_i) \leq \frac{p(I)}{\lfloor 1/(\beta + \alpha) \rfloor + 1} < (\beta + \alpha)p(I)$$

Removing S_{i^*} will create an empty interval of length β in the bin, and the remaining items can be shifted so that they fit in a bin of length 1.

5.2.3.2 Near-optimal Packing of Small Items

Consider a Vector-Max-GAP instance $(I, \overrightarrow{\text{val}}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$. Item i is said to be ε-small for this instance iff $\overrightarrow{w}(i) \leq \varepsilon\overrightarrow{W}$ and for all $j \in [k], (s_j(i) \leq \varepsilon M_j \textit{ or } \text{val}_j(i) = 0)$. A set I of items is said to be ε-small iff each item in I is ε-small.

Suppose I is ε-small. Let $J \subseteq I$ be a feasible solution to $(I, \overrightarrow{\text{val}}, \overrightarrow{s}, \overrightarrow{w}, (1+\varepsilon)\overrightarrow{M}, (1+\varepsilon)\overrightarrow{W})$. Let J_j be the items assigned to the j^{th} machine.

For each $j \in [k]$, use trimming on J_j for sizes s_j and then for each $q \in [d]$, use trimming on J for weights w_q. In both cases, use $\alpha := \varepsilon$ and $\beta := \varepsilon$. Let R be the removed items and $J' = J - R$ be the remaining items. Total value lost is less than $2\varepsilon(d+1)\text{val}(J)$ and J' is feasible for $(\overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$.

Therefore, any resource-augmented solution J of small items can be transformed to get a feasible solution J' of value at least $(1 - 2(d+1)\varepsilon)\text{val}(J)$.

5.2.4 A Structural Result

Theorem 5.4. *Let J be a feasible solution to $(I, \overrightarrow{\text{val}}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$. Let $J_j \subseteq J$ be the items assigned to the j^{th} machine. Then for all $\varepsilon > 0$, there exist sets X and Y such that $|X| \leq (d+k)/(\varepsilon^2)$ and $\text{val}(Y) \leq \varepsilon \cdot \text{val}(J)$ and*

$$\forall j \in [k], \forall i \in J_j - X - Y,\, s_j(i) \leq \varepsilon \left(M_j - s_j\left(X \cap J_j\right)\right)$$
$$\forall i \in J - X - Y,\, \overrightarrow{w}(i) \leq \varepsilon \left(\overrightarrow{W} - \overrightarrow{w}(X)\right)$$

Proof. Let $P_{1,j} = \{i \in J_j : s_j(i) > \varepsilon M_j\}$, where $j \in [k]$, and $Q_{1,q} = \{i \in J : w_q(i) > \varepsilon W_q\}$, where $q \in [d]$. We know that $s_j(P_{1,j}) > \varepsilon M_j |P_{1,j}|$. Therefore, $|P_{1,j}| \leq \frac{1}{\varepsilon}$. Also, $w_q(Q_{1,q}) > \varepsilon W_q |Q_{1,q}|$. Therefore, $|W_{1,q}| \leq \frac{1}{\varepsilon}$. Let $R_1 = \left(\bigcup_{j=1}^{k} P_{1,j}\right) \cup \left(\bigcup_{q=1}^{d} Q_{1,q}\right)$. R_1 is therefore the set of items in J that are in some sense 'big'. Note that $|R_1| \leq (d+k)/\varepsilon$.

If $\text{val}(R_1) \leq \varepsilon \cdot \text{val}(J)$, set $Y = R_1$ and $X = \{\}$ and we're done. Otherwise, set $P_{2,j} = \{i \in J_j - R_1 : s_j(i) > \varepsilon(M_j - s_j(R_1 \cap J_j))\}$, $Q_{2,q} = \{i \in J - R_1 : w_q(i) > \varepsilon(W_q - w_q(R_1))\}$, and $R_2 = \left(\bigcup_{j=1}^{k} P_{2,j}\right) \cup \left(\bigcup_{q=1}^{d} Q_{2,q}\right)$. If $\text{val}(R_2) \leq \varepsilon \cdot \text{val}(J)$, set $Y = R_2$ and $X = R_1$ and we're done. Otherwise, set $P_{3,j} = \{i \in J_j - R_1 - R_2 : s_j(i) > \varepsilon(M_j - s_j((R_1 \cup R_2) \cap J_j))\}$, $Q_{3,q} = \{i \in J - R_1 - R_2 : w_q(i) > \varepsilon(W_q - w_q(R_1) - w_q(R_2))\}$, and $R_3 = \left(\bigcup_{j=1}^{k} P_{3,j}\right) \cup \left(\bigcup_{q=1}^{d} Q_{3,q}\right)$. If $\text{val}(R_3) \leq \varepsilon \cdot \text{val}(J)$, set $Y = R_3$ and $X = R_1 \cup R_2$ and we're done. Otherwise, similarly compute R_4 and check if $\text{val}(R_4) \leq \varepsilon \cdot \text{val}(J)$, and so on. Extending the similar arguments about $|R_1|$, it follows that for all $t > 0$, $|R_t| \leq (d+k)/\varepsilon$.

Since every R_t $(t > 0)$ is disjoint, there will be some $T \leq 1/\varepsilon$ such that $\text{val}(R_T) \leq \varepsilon \cdot \text{val}(J)$.

Now set $Y = R_T$ and $X = R_1 \cup \ldots \cup R_{T-1}$. We can see that $|X| \leq \sum_{t=1}^{T} |R_T| \leq T(d+k)/\varepsilon \leq (d+k)/\varepsilon^2$ and $\text{val}(Y) = \text{val}(R_T) \leq \varepsilon \cdot \text{val}(J)$. Note that all items in $J - X - Y$ are small because of the way R_T was constructed. Hence it follows that

$$\forall j \in [k], \forall i \in J_j - X - Y, s_j(i) \leq \varepsilon(M_j - s_j(X \cap J_j))$$

$$\forall i \in J - X - Y, \overrightarrow{w}(i) \leq \varepsilon \left(\overrightarrow{W} - \overrightarrow{w}(X)\right) \qquad \square$$

5.2.5 PTAS for Vector-Max-GAP

Let J^* be an optimal assignment for $(I, \overrightarrow{\text{val}}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$. Let $J_j^* \subseteq J^*$ be the items assigned to the j^{th} machine.

By Theorem 5.4, J^* can be partitioned into sets X^*, Y^* and Z^* such that $|X^*| \leq \frac{d+k}{\varepsilon^2}$ and $\text{val}(Y^*) \leq \varepsilon \cdot \text{val}(J^*)$. Let $\overrightarrow{W}^* = \overrightarrow{W} - \overrightarrow{w}(X^*)$ and $M_j^* = M_j - s_j(X^* \cap J_j^*)$. Then Z^* is ε-small for $(\overrightarrow{\text{val}}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}^*, \overrightarrow{W}^*)$.

For a set S, define $\Pi_k(S)$ as the set of k-partitions of S.

Algorithm 3 Vector-Max-GAP$(I, \overrightarrow{\text{val}}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$: PTAS for Vector-Max-GAP

```
1: $J_{\text{best}} = \{\}$.
2: for $X \subseteq I$ such that $|X| \le (d+k)/\varepsilon^2$ do
3:     for $(X_1, X_2, \ldots, X_k) \in \Pi_k(X)$ do
4:         $\overrightarrow{W}' = \overrightarrow{W} - \overrightarrow{w}(X)$
5:         $M'_j = M_j - s_j(X_j)$ for each $j \in [k]$.
6:         $\text{val}'_j(i) = \begin{cases} \text{val}_j(i) & \text{if } s_j(i) \le \varepsilon M'_j, \forall q \in [d], w_q(i) \le \varepsilon W'_q \\ 0 & \text{otherwise} \end{cases}$ for each $i \in I - X, j \in [k]$.
7:                                  $\triangleright$ $I - X$ is $\varepsilon$-small for $(\overrightarrow{\text{val}'}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}', \overrightarrow{W}')$
8:         $Z' = \text{assign-res-aug}_\varepsilon(I - X, \overrightarrow{\text{val}'}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}', \overrightarrow{W}')$.
9:         Trim $Z'$ to get $Z$ so that $Z$ is feasible for $(\overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}', \overrightarrow{W}')$.
10:        $J = X \cup Z$
11:        if $\text{val}(J) > \text{val}(J_{\text{best}})$ then
12:            $J_{\text{best}} = J$
13:        end if
14:    end for
15: end for
16: return $J_{\text{best}}$
```

Correctness: Since Z is feasible for $(\overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}', \overrightarrow{W}')$, $X \cup Z$ is feasible for $(\overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$.

Approximation guarantee:

For some iteration of Algorithm 3, $X = X^*$ and $X_j = X^* \cap J^*_j$ for all $j \in [k]$. When that happens, $\overrightarrow{W}' = \overrightarrow{W}^*$ and $\overrightarrow{M}' = \overrightarrow{M}^*$. Let val* be the maximum value ε-small assignment of items to the machines with capacities given by $\overrightarrow{M}'$ and over all weight constraints $\overrightarrow{W}'$. Therefore, $\text{val}^* \ge \text{val}(Z^*)$.

To try to find an optimal assignment of small items, we'll forbid non-small items to be assigned to a machine. We do this in line 6: if for item i, $s_j(i) > \varepsilon M'_j$ for any $j \in [k]$ or $w_q(i) > \varepsilon W'_q$ for any $q \in [d]$, set $\text{val}_j(i)$ to 0. Using our resource-augmented Vector-Max-GAP algorithm, we get $\text{val}(Z') \ge \text{val}^*$. By the property of trimming, $\text{val}(Z) \ge (1 - 2(d+1)\varepsilon)\text{val}(Z')$.

$$\text{val}(J_{\text{best}}) \ge \text{val}(X^*) + \text{val}(Z) \ge \text{val}(X^*) + (1 - 2(d+1)\varepsilon)\text{val}(Z^*) \ge (1 - 2(d+1)\varepsilon)(1 - \varepsilon)\text{val}(J^*)$$

This gives us a $(1-(2d+3)\varepsilon)$-approx solution. The running time can be easily seen to be polynomial as assign-res-aug runs in polynomial time and the number of iterations of the outer loop in Algorithm 3 is polynomial in n and for one iteration of the outer loop, the inner loop runs at most constant number of times.

5.3 Algorithm for (2, d) Knapsack Problem

In this section, we will obtain a $(2+\varepsilon)$-approximation algorithm for the (2, d) knapsack problem for both the cases of 'rotations allowed' as well as 'rotations not allowed'.

Let I be a set of n (2, d)-dimensional items. We are given a (2, d)-dimensional knapsack and we would like to pack a high profit subset of I in the knapsack. Let us denote this optimal profit by $\mathrm{OPT}_{\mathrm{GVKS}}(I)$. Let $\overrightarrow{w} = [w(1), \dots, w(n)]$, $\overrightarrow{h} = [h(1), \dots, h(n)]$, $\overrightarrow{p} = [p(1), \dots, p(n)]$. For an item $i \in I$, let $\overrightarrow{v}(i) = [v_1(i), \dots, v_d(i)]$ and let $\overrightarrow{v} = [\overrightarrow{v}(1), \dots, \overrightarrow{v}(n)]$.

In the whole section, a *container* is a rectangular region inside the knapsack. For our purposes, every container can be one of the four types: *large, wide, tall, area.* A large container can contain at most one item. An area container can only contain items that are ε-small for the container, i.e., an item can be packed into an area container of width w and height h only if the item has width at most εw and height at most εh. In a wide (resp. tall) container, items must be stacked up one on top of another (resp. arranged side by side). We also require that the containers do not overlap amongst themselves.

5.3.1 A Structural Result

Consider a set of items S that are packed in a knapsack. We now state a structural result, inspired by [GGH+17], where a subset of items $S' \subseteq S$ is packed into the knapsack. We may lose some profit but the packing has a nice structure which can be searched for, efficiently.

Theorem 5.5. *Let S denote a set of items that can be feasibly packed into a knapsack and let $0 < \varepsilon < 1$ be any small constant. Then there exists a subset $S' \subseteq S$ such that $p(S') \geq (1/2-\varepsilon) \cdot p(S)$. The items in S' are packed into the knapsack in containers. Further, the number of containers formed is $O_\varepsilon(1)$ and their widths and heights belong to a set whose cardinality is* $\mathrm{poly}(|S|)$ *and moreover, this set can be computed in time* $\mathrm{poly}(|S|)$.

Now, let us go back to our original problem instance I. Let I_{Opt} be the set of items packed into the knapsack in an optimal packing $\mathcal{P}$.

Let us apply Theorem 5.5 to the set of items I_{Opt} with $\varepsilon := \varepsilon_{\mathrm{struct}}$ ($\varepsilon_{\mathrm{struct}}$ will be defined later). Let I'_{Opt} be the resulting analog of S' in the theorem (there can be many candidates for

I'_{Opt} but let us pick one). Therefore,

$$p(I'_{\text{Opt}}) \geq \left(\frac{1}{2} - \varepsilon_{\text{struct}}\right) \cdot p(I_{\text{Opt}}) = \left(\frac{1}{2} - \varepsilon_{\text{struct}}\right) \cdot \text{OPT}_{\text{GVKS}}(I) \tag{5.1}$$

5.3.2 Proof of the Structural Result

In this subsection, we will prove Theorem 5.5.

The strategy to prove the theorem is to use the *corridor decomposition* scheme, introduced by [AW15]. First, we assume that we can remove $O_\varepsilon(1)$ number of items at the cost of zero profit loss from the originally packed items. Under this assumption, we show that we lose at most half profit by our restructuring. Finally, we show how to get rid of this assumption by using shifting argumentations.

Removing Medium items: Let $\varepsilon_{\text{small}}, \varepsilon_{\text{big}}$ be two fixed constants such that $\varepsilon_{\text{big}} > \varepsilon_{\text{small}}$. We partition the items in S based on the values of $\varepsilon_{\text{small}}$ and ε_{big} as follows:

- $S_S = \{i \in S : w(i) \leq \varepsilon_{\text{small}} \text{ and } h(i) \leq \varepsilon_{\text{small}}\}$
- $S_B = \{i \in S : w(i) > \varepsilon_{\text{big}} \text{ and } h(i) > \varepsilon_{\text{big}}\}$
- $S_W = \{i \in S : w(i) > \varepsilon_{\text{big}} \text{ and } h(i) \leq \varepsilon_{\text{small}}\}$
- $S_T = \{i \in S : w(i) \leq \varepsilon_{\text{small}} \text{ and } h(i) > \varepsilon_{\text{big}}\}$
- $S_{\text{med}} = \{i \in S : w(i) > \varepsilon_{\text{small}} \text{ and } w(i) \leq \varepsilon_{\text{big}} \quad \text{OR} \quad h(i) \leq \varepsilon_{\text{big}} \text{ and } h(i) > \varepsilon_{\text{small}}\}$

We call the items in S_S as *small* items as they are small in both the dimensions. Similarly, we call the items in $S_B, S_W, S_T, S_{\text{med}}$ as *big, wide, tall, medium* respectively.

By standard arguments, it is possible to choose the constants $\varepsilon_{\text{small}}$ and ε_{big} such that the total profit of all the medium items is at most $\varepsilon \cdot p(S)$. Hence, we can neglect the items in S_{med} from S while losing a very small profit (at most ε fraction). This is formalized in the following lemma.

Lemma 5.6. *Let $\varepsilon \in (0,1)$ and let $f : (0,1) \mapsto (0,1)$ be a strictly increasing function such that $f(x) > x$ for all $x \in (0,1)$ and let S be a given set of items. Let $f^{(k)}$ be the function f composed k times. For any given integer $k > 0$, define*

$$S_k = \left\{ i \in S \middle| w(i) \in \left(f^{(k)}(0), f^{(k+1)}(0)\right] \bigvee h(i) \in \left(f^{(k)}(0), f^{(k+1)}(0)\right] \right\}$$

Then for some $k' \in [2\lceil 1/\varepsilon \rceil]$,

$$p(S_{k'}) \leq \varepsilon p(S)$$

Proof. Consider any item i. There exists at most one k_w such that $w(i) \in \left(f^{(k_w)}(0), f^{(k_w+1)}(0)\right]$. Similarly there exists at most one k_h such that $h(i) \in \left(f^{(k_h)}(0), f^{(k_h+1)}(0)\right]$. So, an item $i \in S$ can belong to at most two sets in the family of sets $\{S_k\}_{k>0}$. Therefore, for some $k' \in [2\lceil 1/\varepsilon \rceil]$, $p(S_{k'}) \le \varepsilon p(S)$. □

Hence, we can choose $\varepsilon_{\text{small}}$ as $f^{(k')}(0)$ and ε_{big} as $f^{(k'+1)}(0)$ where k' is as guaranteed by the above lemma. This ensures that the profit of the medium items S_{med} is at most $\varepsilon p(S)$.

Remark 5.7. *Since $\varepsilon_{\text{big}} = f(\varepsilon_{\text{small}})$ and f can be any arbitrary increasing function, we can choose f such that the ratio $\varepsilon_{\text{big}}/\varepsilon_{\text{small}}$ is arbitrarily large. Hence, we can assume that $\varepsilon_{\text{big}}, \varepsilon_{\text{small}}$ are* independently chosen. *Also, since $\varepsilon_{\text{small}} = f(0)$, we can make $\varepsilon_{\text{small}}$ an arbitrarily small constant.*

The above remark will be extremely crucial for our structural result.

5.3.2.1 Corridors

First, let us define what a subcorridor is. A subcorridor is just a rectangle in the 2D coordinate system with one side longer than ε_{big} and the other side having a length of at most ε_{big}. A subcorridor is called wide (resp. tall) if the longer side is parallel to the x-axis (resp. y-axis).

A corridor is just a subcorridor or a union of at most $1/\varepsilon$ subcorridors such that each wide (resp. tall) subcorridor overlaps with exactly two tall (resp. wide) subcorridors, except for at most two subcorridors which are called the *ends* of the corridors and can overlap with exactly one subcorridor. The overlap between any two subcorridors should be in such a way that one of their corners coincide. See Fig. 5.1 for an illustration.

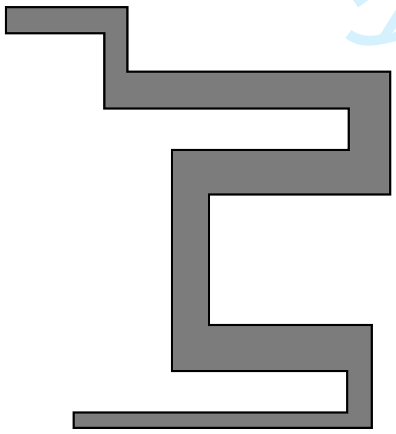


Figure 5.1: A corridor which is just a union of 9 subcorridors.

5.3.2.2 Corridor Decomposition

For now, let's consider a generic packing of a set of items S that can contain big, small, wide, tall items.

Since the number of big items packed can be at most a constant, and since we assumed that we can remove a constant number of items at no profit loss, let us remove all the big items from the packing. Let's also get rid of the small items for now (we will pack the small items later). Hence, we are left with wide and tall items packed in the knapsack. Let's name these items as *skewed* items and denote the set of these items by S_{skew}.

Now we will discuss the corridor packing lemma used to partition to the knapsack into corridors.

Lemma 5.8 (Corridor Packing lemma [AW15]). *Let $\varepsilon > 0$ be some accuracy parameter. Consider a set of items S such that each item has one of the sides at least δ. Then, there exist non-overlapping corridor regions in the knapsack such that we can partition S into sets S_{corr}, $S^{\text{nice}}_{\text{cross}}$, $S^{\text{bad}}_{\text{cross}}$, S_{free} such that*

- $\left|S^{\text{nice}}_{\text{cross}} \cup S_{\text{free}}\right| = O_{\varepsilon}(1)$
- $p(S^{\text{bad}}_{\text{cross}}) \leq O(1)\varepsilon \cdot p(S_{\text{skew}})$
- *Every item in S_{corr} is fully contained in one of the corridors. The number of corridors is $O_{\varepsilon,\delta}(1)$ and in each corridor, the number of subcorridors is at most $1/\varepsilon$.*
- *Each subcorridor has length at least δ and breadth less than δ (assuming length to denote the longest side and breadth to denote the smallest side).*

Remark 5.9. *In the above lemma, the set S can contain items which are not skewed as long as at least one of their sides is $\geq \delta$.*

The above remark will be useful in Section 5.3.2.5.

We apply the corridor decomposition lemma to items in S_{skew} with $\delta = \varepsilon_{\text{big}}$. We remove items in $S^{\text{nice}}_{\text{cross}} \cup S_{\text{free}}$ since they are at most a constant in number and items in $S^{\text{bad}}_{\text{cross}}$ since their total profit is very less. See Fig. 5.2 for an illustration of the lemma.

The last point of the lemma ensures that any item in S_{corr} is completely contained in exactly one subcorridor. Hence for every skewed item contained in a corridor, we can tell which subcorridor it *belongs* to. The last point also ensures that, there can not be a subcorridor which completely contains both wide and tall items. This fact allows us to label each subcorridor as

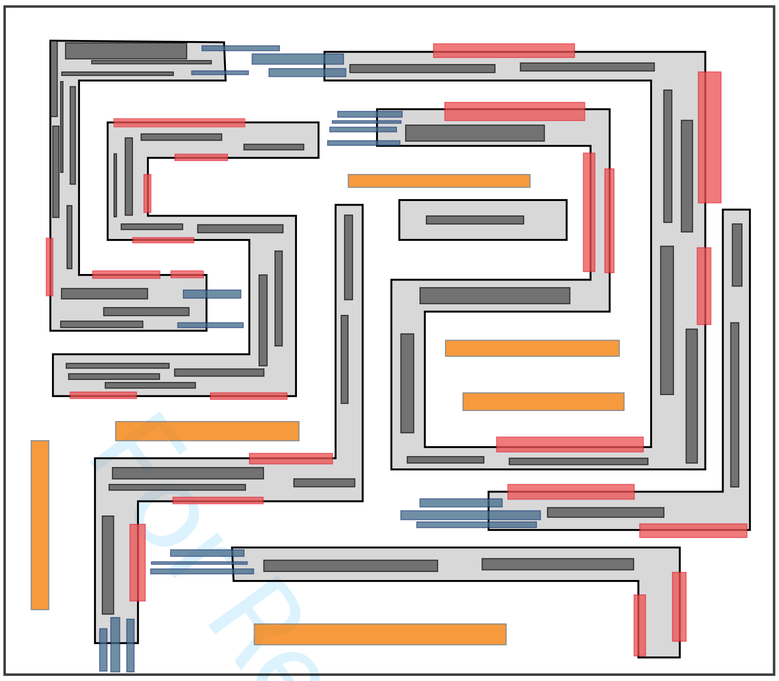


Figure 5.2: Skewed items in corridors. Dark items indicate the set S_{corr}. Red items indicate the set $S_{\text{cross}}^{\text{nice}}$. They are constant in number. Blue items indicate the set $S_{\text{cross}}^{\text{bad}}$. They carry very marginal profit. Orange items which indicate S_{free} are not contained in any corridor. They are constant in number.

wide or tall: If a subcorridor contains only wide (resp. tall) items, we say that it is a wide (resp. tall) subcorridor.

Removing either wide or tall items: Now, we will simplify the above structure of skewed items while losing at most half of the original profit.

Assume without loss of generality that the total profit of wide items is at least as much as the total profit of the tall items. Hence, if we get rid of the tall items, we lose at most half of the original profit. With this step, we can also remove all the tall subcorridors since they are all empty. We are left with wide items packed in wide subcorridors. Since the subcorridors are just rectangles and they no longer overlap, we just call these subcorridors as *boxes*.

Next, we will describe how to reorganize the items in these boxes into containers at a very marginal profit loss.

5.3.2.3 Reorganizing Boxes into Containers

Note that the boxes contain only wide items. And since the width of the wide items is lower bounded by ε_{big}, we can create an empty wide strip in each box at the loss of very small profit and at most a constant number of items. This is described in the following lemma.

Lemma 5.10. *Let B be a box of dimensions $a \times b$ such that each item contained in it has a width of at least μ, where μ is a constant. Let the total profit packed inside the item be P and let ε_{box} be small constant. Then, it is possible to pack all the items barring a set of items of profit at most $\varepsilon_{\text{box}} \cdot P$ and a set of $O_\mu(1)$ number of items into a box of dimensions $a \times (1 - \varepsilon_{\text{box}})b$.*

Proof. Assume the height of the box is along the y-axis and width of the box is along the x-axis and say, the bottom left corner of the box is situated at the origin. Draw lines at $y = y_i$ for $i \in \{1, \ldots, \lfloor 1/\varepsilon_{\text{box}} \rfloor\}$ such that $y_i = i \cdot \varepsilon_{\text{box}} b$. These lines will partition the box into $\lceil 1/\varepsilon_{\text{box}} \rceil$ regions. For all $i \in \{1, \ldots, \lceil 1/\varepsilon_{\text{box}} \rceil\}$, let s_i be the set of items completely contained in the i^{th} region. Then, there must exist some region j such that $p(s_j) \leq (1/\lceil 1/\varepsilon_{\text{box}} \rceil) \cdot P$. We will remove all the items in s_j. Also, the number of items partially overlapping with the region j can be at most $2/\mu$, which is $O_\mu(1)$. By removing these items, we create an empty strip of size $a \times \varepsilon_{\text{box}} b$ and hence the remaining items can be packed in a box of size $a \times (1 - \varepsilon_{\text{box}})b$. □

We apply the above lemma to each and every box with small enough ε_{box} and $\mu = \varepsilon_{\text{big}}$. Consider a box B and let S_{box} be the items packed inside B. Also, let $a \times b$ be the dimensions of the box. Let S'_{box} be the set of items in the knapsack after performing the steps in Lemma 5.10. S'_{box} can be packed within the box $a \times (1 - \varepsilon_{\text{box}})b$ and we are left with some empty space in the vertical direction. Hence, we can apply the resource augmentation lemma, which is taken from [GGH+17].

Lemma 5.11 (Resource Augmentation Lemma). *Let I be a set of items packed inside a box of dimensions $a \times b$ and let ε_{ra} be a given constant. Then there exists a container packing of a set $I' \subseteq I$ inside a box of size $a \times (1 + \varepsilon_{\text{ra}})b$ such that*

- $p(I') \geq (1 - O(1)\varepsilon_{\text{ra}}) \cdot p(I)$
- *The number of containers is $O_{\varepsilon_{\text{ra}}}(1)$ and the dimensions of the containers belong to a set of cardinality $O\left(|I|^{O_{\varepsilon_{\text{ra}}}(1)}\right)$.*
- *The total area of the containers is at most $a(I) + \varepsilon_{\text{ra}} \cdot ab$*

Applying the above lemma to the box B, we obtain a packing where S'_{box} is packed into box of size $a \times (1 - \varepsilon_{\text{box}})(1 + \varepsilon_{\text{ra}})b$. This process of obtaining boxes from containers is shown in Fig. 5.3.

We will choose $\varepsilon_{\text{box}}, \varepsilon_{\text{ra}}$ such that the product $(1 - \varepsilon_{\text{box}})(1 + \varepsilon_{\text{ra}}) < (1 - 2\varepsilon)$.

Lemma 5.12. *The total area of containers obtained is at most*

$$\min\{(1 - 2\varepsilon), a(S_{\text{corr}}) + \varepsilon_{\text{ra}}\}$$

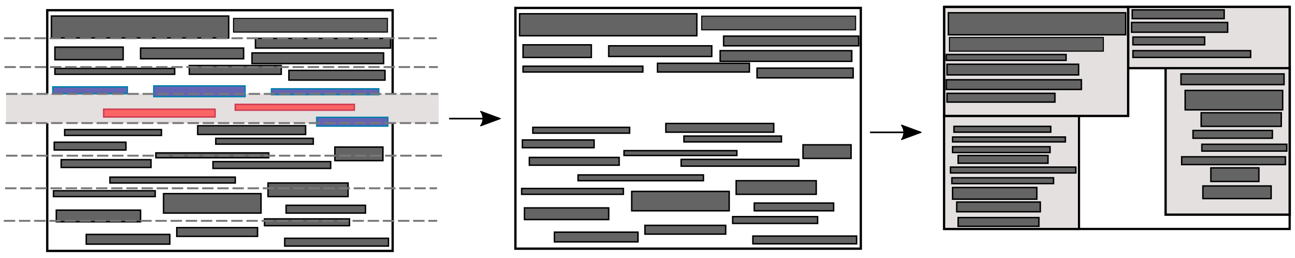


Figure 5.3: Obtaining containers from a box in two steps. First, we remove the strip (shaded in light grey) with least profit of items completely contained in it (shown in red). We also remove the items partially intersecting that strip (shown in blue. These will be constant in number). Then in the second step, we apply resource augmentation lemma to obtain constant number of containers.

Proof. The second upper bound directly follows from the last point of Lemma 5.11: Area of the containers in a box B of dimensions $a \times b$ is at most $a(S_{\text{box}}) + \varepsilon_{\text{ra}} \cdot ab$. Summing over all boxes we get the bound $a(S_{\text{corr}}) + \varepsilon_{\text{ra}}$.

For the first upper bound, observe that every box of dimensions $a \times b$ is shrunk into a box of dimensions $a \times (1 - 2\varepsilon)b$. Since all the original boxes were packable into the knapsack, the total area of the boxes (and hence the containers) after shrinking is at most $(1 - 2\varepsilon)$ times the area of the knapsack itself, which is 1. □

5.3.2.4 Packing Small Items

In this subsection, we show how to pack small items that have been removed from the packing temporarily. Let the set of small items be denoted by S_{small}. Let $\varepsilon_{\text{grid}} = \varepsilon_{\text{small}}/\varepsilon$. We will make use of the crucial fact that the number of containers created does not depend on $\varepsilon_{\text{small}}$ and $\varepsilon_{\text{grid}}$ does not depend on ε_{ra}.

Let us define a uniform grid $\mathcal{G}$ in the knapsack where grid lines are equally spaced at a distance of $\varepsilon_{\text{grid}}$. It is easy to see that the number of grid cells formed is at most $\lceil 1/\varepsilon_{\text{grid}} \rceil^2$ which is at most a constant. We mark a grid cell as *free* if it has no overlapping with any container and *non-free* otherwise. We delete all the non-free cells and the free cells will serve as the area containers that we use to pack the small items.

Lemma 5.13. *For some choice of $\varepsilon_{\text{small}}$ and ε_{ra}, the total area of non-free cells is at most*

$$\min\left\{(1 - \varepsilon), a(S_{\text{corr}}) + 3\varepsilon^2\right\}$$

Proof. Let A be the total area of containers and k be the number of containers. The total area of cells completely covered by containers is at most A. The partially overlapped cells are the cells that intersect the edges of the containers. Since, the number of containers is k and each container has 4 edges and each edge can overlap with at most $\lceil 1/\varepsilon_{\text{grid}} \rceil$ number of cells, the area of completely and partially overlapped cells is at most

$$A + 4k \cdot \lceil 1/\varepsilon_{\text{grid}} \rceil \cdot \varepsilon_{\text{grid}}^2$$

As we noted before, $\varepsilon_{\text{grid}}$ depends on $\varepsilon_{\text{small}}$ but does not depend on ε_{ra}. On the other hand, k, the number of containers depends on ε_{ra} but does not depend on $\varepsilon_{\text{small}}$. Hence, for some carefully chosen $\varepsilon_{\text{grid}}$ and ε_{ra}, we can ensure that the above quantity is at most $A + 2\varepsilon^2$.

By Lemma 5.12, the value of A is bounded by

$$\min \{(1 - 2\varepsilon), a(S_{\text{corr}}) + \varepsilon_{\text{ra}}\} .$$

Hence, the total area of deleted cells is at most

$$\min \left\{(1 - 2\varepsilon + 2\varepsilon^2), a(S_{\text{corr}}) + \varepsilon_{\text{ra}} + 2\varepsilon^2\right\}$$

For $\varepsilon < 1/2$ and by choosing $\varepsilon_{\text{ra}} < \varepsilon^2$, we get the desired result. □

We now show that there is a significant area of free cells left to pack the small items profitably.

Lemma 5.14. *The area of free cells is at least* $(1 - 3\varepsilon) \cdot a(S_{\text{small}})$

Proof. Since S_{small} and S_{corr} were packable in the knapsack, $a(S_{\text{small}} \cup S_{\text{corr}}) \leq 1$.

Now if $a(S_{\text{small}}) \geq \varepsilon$, then the area of free cells is at least $1 - a(S_{\text{corr}}) - 3\varepsilon^2 \geq a(S_{\text{small}}) - 3\varepsilon^2 \geq (1 - 3\varepsilon) \cdot a(S_{\text{small}})$

On the other hand, if $a(S_{\text{small}}) < \varepsilon$, then the area of free cells is at least ε which in turn is at least $a(S_{\text{small}})$. □

Each free cell has dimensions $\varepsilon_{\text{grid}} \times \varepsilon_{\text{grid}}$ and a small item has each side at most $\varepsilon_{\text{small}}$. Hence, all small items are ε-small for the created free cells and these can be used to pack the small items. Using NFDH, we can pack a very high profit in the area containers as described in the following lemma.

Lemma 5.15. *Let I be a set of items and let there be k identical area containers such that each item $i \in I$ is ε-small for every container and the whole set I can be packed in these containers. Then there exists an algorithm which packs a subset $I' \subseteq I$ in these area containers such that $p(I') \geq (1 - 2\varepsilon) \cdot p(I)$.*

Proof. Without loss of generality, assume that each container has dimensions 1×1. So, we can assume that every item $i \in I$ has a width of at most ε and a height of at most ε. Order the items in I in non-increasing ratio of profit/area (we call this profit density) breaking ties arbitrarily and also order the containers arbitrarily.

Start packing items into the containers using NFDH. If we are able to pack all the items in I, we are done. Otherwise, consider an arbitrary container C. Let I_C be the items we packed in C and let i_C be the item we could not pack in C. Then by Lemma 5.1, $a(I_C \cup \{i_C\}) > (1-\varepsilon)^2 \cdot a(C) = (1-\varepsilon)^2$. But since $a(i_C) \leq \varepsilon^2$, $a(I_C) > 1-2\varepsilon$. Hence, we have packed at least $(1-2\varepsilon)$ fraction of the total area of the containers with the densest items and thus the claim follows. □

To this end, we have packed at least $(1/2 - O(1)\varepsilon)$ fraction of the total profit in the original packing assuming that we can leave out a constant number of items at no cost.

5.3.2.5 Shifting Argumentation

We assumed that we can drop $O_\varepsilon(1)$ number of items from S at no profit loss. But this may not be true because they can carry significant amount of profit in the original packing. The left out constant number of items are precisely the big items and items in $S_{\text{cross}}^{\text{nice}} \cup S_{\text{free}}$ and the items partially overlapping with the removed strip in Lemma 5.10. Also, recall that we neglected small items from the packing and added them right at the end. Here, we will not neglect small items all together. Rather, we will neglect only a tiny volume of them as we go. We will add back this tiny volume at the end.

The main idea is to fix some items in the knapsack and then carry out our algorithm discussed in the previous sections with some modifications. Again, we may leave out some items with high profit. We fix these items too and repeat this process. We can argue that at some point, the left out items have a very less profit and hence this process will end.

Let us define the set $K(0)$ as the set of items that were removed in the above process. If the total profit of $K(0)$ is at most $\varepsilon \cdot p(S)$, then we are done. If this is not the case, then we use shifting argumentation.

Assume that we completed the t^{th} round for some $t > 0$ and say $p(K(r)) > \varepsilon \cdot p(S)$ for all $0 \leq r \leq t$. Let $\mathcal{K}(t) = \bigcup_{r=0}^{t} K(t)$. We will argue that for every $r \leq t$, $|K(r)|$ is at most a constant. Then $|\mathcal{K}(t)|$ is also at most a constant if $t < \lceil 1/\varepsilon \rceil$ (in fact, we will argue that t will not go beyond $\lfloor 1/\varepsilon \rfloor$).

Let us define a non-uniform grid $\mathcal{G}(t)$ induced by the set $\mathcal{K}(t)$ as follows: The x and y coordinates of the grid are given by all the corners of the items in $\mathcal{K}(t)$. Note that the number of horizontal (resp. vertical) grid lines is bounded by $2 \cdot |\mathcal{K}(t)|$. This grid partitions the knapsack into a set of cells $\mathcal{C}(t)$. Since $|\mathcal{K}(t)|$ is at most a constant, the number of grid cells created is also at most a constant. Let the collection of these grid cells be denoted by $\mathcal{C}(t)$.

Let us denote an arbitrary grid cell by C and the items in S which intersect C and which are not in $\mathcal{K}(t)$ by $S(C)$. For an item i in $S(C)$, let $h(i \cap C)$ and $w(i \cap C)$ denote the width and height of the overlap of an item i with C. We categorize the items in $S(C)$ relative to C as follows.

- $S_{\text{small}}(C) = \{i \in S(C) : h(i \cap C) \leq \varepsilon_{\text{small}} h(C) \text{ and } w(i \cap C) \leq \varepsilon_{\text{small}} w(C)\}$
- $S_{\text{big}}(C) = \{i \in S(C) : h(i \cap C) > \varepsilon_{\text{big}} h(C) \text{ and } w(i \cap C) > \varepsilon_{\text{big}} w(C)\}$
- $S_{\text{wide}}(C) = \{i \in S(C) : h(i \cap C) \leq \varepsilon_{\text{small}} h(C) \text{ and } w(i \cap C) > \varepsilon_{\text{big}} w(C)\}$
- $S_{\text{tall}}(C) = \{i \in S(C) : h(i \cap C) > \varepsilon_{\text{big}} h(C) \text{ and } w(i \cap C) \leq \varepsilon_{\text{small}} w(C)\}$

We call an item i *small* if it is not in $S_{\text{big}}(C) \cup S_{\text{wide}}(C) \cup S_{\text{tall}}(C)$ for some cell C. We call an item i *big* if it is in $S_{\text{big}}(C)$ for some cell C. We call an item i *wide* (resp. *tall*) if it is in $S_{\text{wide}}(C)$ (resp. $S_{\text{tall}}(C)$) for some cell C.

We call an item i *medium* if there is a cell C such that $h(i \cap C) \in (\varepsilon_{\text{small}} h(C), \varepsilon_{\text{big}} h(C)]$ or $w(i \cap C) \in (\varepsilon_{\text{small}} w(C), \varepsilon_{\text{big}} w(C)]$.

It is easy to observe that an item must belong to exactly one of small, big, wide, tall, medium categories. Note that the width and height of a small item are at most $\varepsilon_{\text{small}}$. We call an item i skewed if it is either wide or tall.

Also, observe that an item i is medium if and only if this condition is satisfied for at least one of the four possible cells that contain the four corners of i. This observation gives us the following lemma:

Lemma 5.16. *Let $\varepsilon > 0$ be a constant and $f : (0, 1) \mapsto (0, 1)$ be a positive increasing function such that $f(x) > x$ for all $x \in (0, 1)$. There exist values $\varepsilon_{\text{small}}, \varepsilon_{\text{big}}$ satisfying $\varepsilon \geq \varepsilon_{\text{big}} \geq f(\varepsilon_{\text{small}}) \geq \Omega_\varepsilon(1)$ and $\varepsilon_{\text{small}} \in \Omega_\varepsilon(1)$ such that the total profit of medium items is at most $\varepsilon \cdot p(S)$.*

Remark 5.17. *The values of $\varepsilon_{\text{small}}, \varepsilon_{\text{big}}$ may depend on the current round t we are in.*

Remark 5.18. *Similar to Remark 5.7, we can choose $\varepsilon_{\text{small}}, \varepsilon_{\text{big}}$ independently, and we can make $\varepsilon_{\text{small}}$ arbitrarily small.*

We add all the big items to $K(t + 1)$. We can do this because the big items are at most constant in number: Consider any cell C. The number of big items associated with it is at most a constant and the number of cells themselves is at most a constant.

We create a corridor decomposition in the knapsack w.r.t. the skewed items as follows: First we transform this non-uniform grid into a uniform grid by moving the grid lines and simultaneously stretching or compressing the items. Let's make a few useful observations about this transformation of non-uniform grid to uniform grid.

Observation 5.19. *Each cell of the thus obtained uniform grid has each of its side lengths at least*

$$\frac{1}{2\,|\mathcal{K}(t)|+1}$$

Observation 5.20. *Since each item is simultaneously stretched or compressed along with the grid lines, the fraction of its width (resp. height) lying inside a cell doesn't change with the transformation. This, in particular, implies that every wide (resp. tall) item has width (resp. height) at least*

$$\varepsilon'_{\text{big}} := \frac{\varepsilon_{\text{big}}}{2\,|\mathcal{K}(t)|+1}$$

Since we have ensured that each skewed item has one of its lengths at least $\varepsilon'_{\text{big}}$, we apply Lemma 5.8 on the set of skewed items with $\delta := \varepsilon'_{\text{big}}$ and create a set of $O_{\varepsilon,\varepsilon'_{\text{big}}}(1)$ corridors in the knapsack.

Remark 5.21. *We do the corridor decomposition only with respect to the skewed items, i.e., as if only skewed items are present in the knapsack. This can lead to some corridors intersecting some small items or some items from $\mathcal{K}(t)$.*

Let $S_{\text{corr}}, S^{\text{nice}}_{\text{cross}}, S^{\text{bad}}_{\text{cross}}, S_{\text{free}}$ be the partition of the skewed items obtained as defined in Lemma 5.8. The set S_{corr} is the set of items that are packed in the corridors completely. We add the set $S^{\text{nice}}_{\text{cross}} \cup S_{\text{free}}$ to $K(t+1)$ since they are constant in number. As the set $S^{\text{bad}}_{\text{cross}}$ has a very small profit, we discard them. We also remove all the skewed items in tall subcorridors assuming, without loss of generality, that their profit is at most that of the skewed items in wide subcorridors.

Now, we have the items in $\mathcal{K}(t)$ fixed inside the knapsack and the wide items in boxes (which are just wide subcorridors obtained after deleting the tall items and hence deleting the tall subcorridors). But, there can be small items intersecting the edges of a box. We initialize an empty set TempSmall and move all these small items to it temporarily. The set TempSmall will be packed at the end.

Remark 5.22. *The volume of all these small items temporarily moved to* TempSmall *is at most $\varepsilon_{\text{small}}$ multiplied by 'four times the number of boxes'. The number of boxes can be a large constant, but it only depends on $\varepsilon, \varepsilon'_{\text{big}}$.*

We would like to split the boxes into containers as in Section 5.3.2.3 but there is an issue: There can be items in $\mathcal{K}(t)$ which overlap with the boxes. But these are at most a constant in number and hence we can resolve this issue in the following way.

Consider an item i in $\mathcal{K}(t)$ partially overlapping with a box. Without loss of generality, assume that it intersects the upper edge. We extend each of its horizontal edge that is inside the box in both the directions till the ends of the box. This extended edge and i divide the items in the box into at most five categories: The wide items intersecting the extended edge, the small items intersecting the extended edge, the items to the left of i, the items to the right of i, the items below i. We note that the items in the first category are at most a constant in number and hence add them to $K(t+1)$. The items in the second category are added to TEMPSMALL. Thus i splits the box into at most three smaller boxes. We repeat this process for all the items in $\mathcal{K}(t)$ overlapping with the box. We obtain smaller boxes but with the required property that there are no overlapping items in $\mathcal{K}(t)$ with the boxes. This is depicted in Figure 5.4.

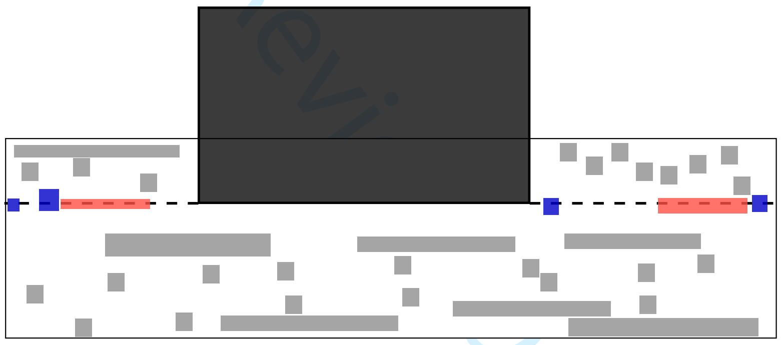


Figure 5.4: Division of box into smaller boxes: The dark item is the item in $\mathcal{K}(t)$ that overlaps with the box. The dashed lines are the extended edges and the red items are those that will be included in $K(t+1)$. The blue items are the small items that intersect the extended edges. They will be added to TEMPSMALL. This leads to three smaller boxes.

Remark 5.23. *For a box, the volume of small items added to* TEMPSMALL *in the above step is* $\varepsilon_{\text{small}}$ *multiplied by 'the total length of the extended edges'. But the total length of the extended edges is at most* $|\mathcal{K}(t)|$.

For each box, we perform the above process of getting rid of overlaps with items in $\mathcal{K}(t)$.

Remark 5.24. *The total number of boxes finally obtained only depends on* $|\mathcal{K}(t)|, \varepsilon, \varepsilon'_{\text{big}}$.

We apply Lemma 5.10 to these smaller boxes with a small change. As usual, we remove a least profitable strip, but while doing this, we add all the wide items partially overlapping with the removed strip to $K(t+1)$. We also add all the small items partially overlapping with the removed strip to TEMPSMALL.

Remark 5.25. *The volume of small items added to* TempSmall *in the above step is at most* $\varepsilon_{\text{small}}$ *multiplied by 'twice the number of boxes'.*

Then, we apply Lemma 5.11 to these new boxes to split them into containers as in Section 5.3.2.3. Recall that a box of dimensions $a \times b$ is first converted to a box of dimensions $a \times (1-\varepsilon_{\text{box}})b$ by removing a strip. Then it is converted to a box of dimensions $a \times (1-\varepsilon_{\text{box}})(1+\varepsilon_{\text{ra}})b$ due to resource augmentation. We chose $\varepsilon_{\text{box}}, \varepsilon_{\text{ra}}$ such that

$$(1-\varepsilon_{\text{box}})(1+\varepsilon_{\text{ra}}) \leq (1-2\varepsilon)$$

At this point, the $(t+1)^{\text{th}}$ round ends. The items added to $K(t+1)$ are the big items whose number only depends on ε_{big}, some wide items whose number only depends on $\varepsilon_{\text{big}}, |\mathcal{K}(t)|$. Hence we can inductively argue that the number of items in $\mathcal{K}(t)$ only depends on ε_{big} and not $\varepsilon_{\text{small}}$.

Now, we look at the set $K(t+1)$. If $p(K(t+1)) \leq \varepsilon \cdot p(S)$, we end; otherwise we continue to round $t+2$. We can argue that the number of rounds are at most $1/\varepsilon$: $K(r)$ and $K(r+1)$ are disjoint for all $r \geq 0$. Hence, for some $r < \lceil 1/\varepsilon \rceil$, we can guarantee that $p(K(r))$ is at most $\varepsilon \cdot p(S)$.

Therefore, after the r^{th} round, we end the process. We are left with the task of adding back the items in TempSmall.

Packing the Small Items. Note that the items in S_{small} − TempSmall are already packed into area containers obtained from the boxes. To pack TempSmall, we use the empty space that was created when we transformed a box into multiple containers. Recall that a box of dimensions $a \times b$ was transformed into a box of dimensions $a \times (1-2\varepsilon)b$ leaving an empty space of $2\varepsilon ab$. We are going to use this empty space to pack TempSmall. Recall from our classification of items that for a cell C,

$$S_{\text{small}}(C) = \{i \in S(C) : h(i \cap C) \leq \varepsilon_{\text{small}} h(C) \text{ and } w(i \cap C) \leq \varepsilon_{\text{small}} w(C)\}$$

and an item is called small if it not big or wide or tall with respect to any cell.

Observation 5.26. *A small item can intersect at most four cells.*

Proof. Note that an edge of a small item can intersect at most two cells. If an edge intersects three cells, then it means that at least one of these cells is completely cut by the edge. But this would imply that the item is wide or tall or big with respect to that cell, which is a contradiction. Since each edge can intersect at most two cells, the item can intersect at most four cells. □

We assign each item in $i \in$ TEMPSMALL to the cell C with the highest area it intersects. This cell must have the largest width as well as the largest height among all the four cells that the item may intersect. This can be seen to be true in Fig. 5.5. Hence it must be the case that

$$w(i) \leq 2\varepsilon_{\text{small}} w(C) \text{ and } h(i) \leq 2\varepsilon_{\text{small}} h(C)$$

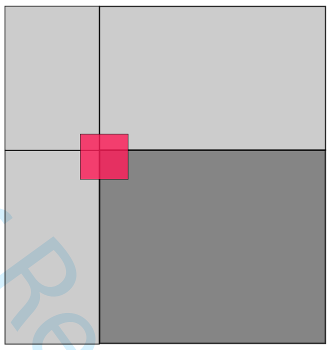


Figure 5.5: The item (red) will be assigned to the cell that is shaded darker.

In this manner, we assign each small item in TEMPSMALL to exactly one of the cells. Let TEMPSMALL(C) denote the set of small items from TEMPSMALL assigned to C. To pack TEMPSMALL(C) in C, we would like to use the empty space created in C when transforming boxes into multiple containers.

Lemma 5.27. *The free space in a cell C is at least $\varepsilon a(C)$.*

Proof. Recall from Observation 5.19, the side length of each grid cell is at least $1/(2\,|\mathcal{K}(t)|+1)$. We applied corridor decomposition with δ as $\varepsilon_{\text{big}}/(2\,|\mathcal{K}(t)|+1)$. Hence, the height of each subcorridor, and hence each box, must be at most $\varepsilon_{\text{big}}/(2\,|\mathcal{K}(t)|+1)$. This implies that the height of a box is at most ε_{big} fraction of the height of a cell.

Let boxes(C) denote the set of boxes that partially or fully intersect C. As shown in Fig. 5.6, some of these boxes contribute to empty space in C, while some don't. To be precise, the boxes that intersect the top edge and the bottom edge of the cell may not contribute to free space. Let's call these boxes as *bad* boxes and the remaining boxes in boxes(C) as *good boxes*. The total area of intersection of bad boxes with the cell C is only at most $2\varepsilon_{\text{big}} a(C)$ since each has height is at most $\varepsilon_{\text{big}} h(C)$. Hence, there is at least $(1 - 2\varepsilon_{\text{big}})a(C)$ area in C that is intersected by good boxes or which is just empty space. Each good box provides an empty area of at least

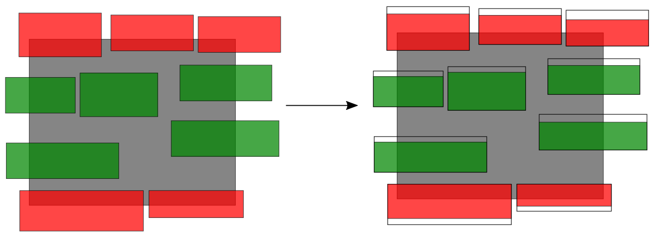


Figure 5.6: Green boxes contribute to a free area of at least 2ε fraction of their intersection with the cell. Red boxes on the other hand may not contribute any empty space.

2ε fraction of its intersection with the cell. Overall, we obtain an empty space of at least

$$2\varepsilon(1 - 2\varepsilon_{\text{big}})a(C) \geq \varepsilon a(C)$$

area, if we ensure that $\varepsilon_{\text{big}} < 1/4$. □

Now, let us look at the total area of TEMPSMALL(C). Remarks 5.22, 5.23 and 5.25 imply that this area is at most $\varepsilon_{\text{small}} N a(C)$ where N is a function of $\varepsilon, \varepsilon_{\text{big}}, \varepsilon'_{\text{big}}, |\mathcal{K}(t)|$. But $\varepsilon'_{\text{big}}$ in turn is a function of $\varepsilon_{\text{big}}, |\mathcal{K}(t)|$, and $|\mathcal{K}(t)|$ is again a function of ε_{big}. Hence, N is just a constant that depends on ε_{big}. Thus, by choosing a small enough $\varepsilon_{\text{small}}$, we can ensure that the total area of the items in TEMPSMALL(C) is at most $\varepsilon^3 a(C)$. Thus, we ensure that the total area to be packed is at most ε^2 fraction of the free space. So, just like in Section 5.3.2.4, we can pack a profit of at least $(1 - \varepsilon)p(\text{TEMPSMALL}(C))$ in the cell C into area containers.

In this manner, considering each cell at a time, we pack an almost optimal subset of TEMPSMALL into area containers in the knapsack.

In [GGH+17], it is also shown how to modify the formed containers so that their widths and heights come from a set of cardinality poly$(|S|)$. This proves Theorem 5.5.

5.3.3 Container Packing Problem and Vector-Max-GAP

From now on, our main goal would be to derive an algorithm to construct the nice structure as in Theorem 5.5. We first formally define the problem of packing items into containers. We then define an extension of the Generalized Assignment Problem, which we call the Vector-Max-GAP problem, for which we obtain a PTAS which in turn can be used to solve the container packing problem.

Let I be a set of items and let $\overrightarrow{w}, \overrightarrow{h}, \overrightarrow{p}, \overrightarrow{v}$ denote the associated widths, heights, profits and weights respectively.

Let C be a given set of containers such that the number of containers is constant. Out of C, let C_A, C_H, C_V, C_L denote area, wide, tall and large containers respectively.

In the *Container Packing Problem*, we would like to pack a subset of I into these containers such that

- A large container can either be empty or can contain exactly one item.
- In a wide (resp. tall) container, all the items must be stacked up one on top of another (resp. arranged side by side).
- The total area of items packed in an area container must not exceed $(1-\varepsilon')^2$ times the area of the container itself (assume that ε' is a constant given as a part of the input).
- The total weight of items packed in all the containers in any dimension $j \in \{1, \ldots, d\}$ should not exceed one.

We denote an instance of the container packing problem by the tuple $(I, \overrightarrow{w}, \overrightarrow{h}, \overrightarrow{p}, \overrightarrow{v}, C, \varepsilon')$.

Now, let us define the Vector-Max-GAP problem. Let I be a set of n items and let us say, we have k machines such that the j^{th} machine has a capacity M_j. An item i has a size of $s_j(i)$, value of $\text{val}_j(i)$ in the j^{th} machine and weight $w_q(i)$ in the q^{th} dimension. The objective is to assign a maximum value subset of items $I' \subseteq I$, each item to a machine, such that the total size of items in a machine does not exceed the capacity of that machine. We also require that the total weight of I' does not exceed W_q (a non-negative real) in any dimension $q \in [d]$.

Let $\overrightarrow{M} = [M_1, M_2, \ldots, M_k]$, $\overrightarrow{W} = [W_1, W_2, \ldots, W_d]$, $\overrightarrow{w}(i) = [w_1(i), w_2(i), \ldots, w_d(i)]$, $\overrightarrow{s}(i) = [s_1(i), s_2(i), \ldots, s_k(i)]$, $\overrightarrow{\text{val}}(i) = [\text{val}_1(i), \text{val}_2(i), \ldots, \text{val}_k(i)]$.

Also, let $\overrightarrow{s} = [\overrightarrow{s}(1), \overrightarrow{s}(2), \ldots, \overrightarrow{s}(n)]$, $\overrightarrow{w} = [\overrightarrow{w}(1), \overrightarrow{w}(2), \ldots, \overrightarrow{w}(n)]$, $\overrightarrow{\text{val}} = [\overrightarrow{\text{val}}(1), \overrightarrow{\text{val}}(2), \ldots, \overrightarrow{\text{val}}(n)]$. We denote an instance of Vector-Max-GAP by $(I, \overrightarrow{\text{val}}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$.

5.3.4 Reduction of Container Packing Problem to Vector-Max-GAP

Let $\mathcal{C} = (I, \overrightarrow{w}, \overrightarrow{h}, \overrightarrow{p}, \overrightarrow{v}, C, \varepsilon')$ denote an instance of the container packing problem. In this subsection, we show how to reduce $\mathcal{C}$ to an instance $\mathcal{G} = (I, \overrightarrow{\text{val}}, \overrightarrow{s}, \overrightarrow{w}, \overrightarrow{M}, \overrightarrow{W})$ of the Vector-Max-GAP problem.

We retain I from $\mathcal{C}$. Since we have unit vector constraints over all the containers combined in the container packing problem, we initialize $\overrightarrow{W}$ to be the vector of all ones. We choose the number of machines in $\mathcal{G}$ to be same as the number of containers in $\mathcal{C}$. Let $|C| = k$ and $|I| = n$.

The vectors $\overrightarrow{\text{val}}, \overrightarrow{s}, \overrightarrow{M}$ are defined as follows in case of rotations being forbidden:

$$s_j(i) = \begin{cases} 1 & \text{if } C_j \text{ is a large container and } i \text{ can fit in inside } C_j \\ \infty & \text{if } i \text{ can not fit in inside } C_j \\ h(i) & \text{if } C_j \text{ is a wide container and } i \text{ fits in inside } C_j \\ w(i) & \text{if } C_j \text{ is a tall container and } i \text{ fits in inside } C_j \\ w(i)h(i) & \text{if } C_j \text{ is an area container and } i \text{ is } \varepsilon'\text{-small for } C_j \\ \infty & \text{if } C_j \text{ is an area container but } i \text{ is not } \varepsilon'\text{-small for } C_j \end{cases}$$

$$M_j = \begin{cases} 1 & \text{if } C_j \text{ is a large container} \\ h(C_j) & \text{if } C_j \text{ is a wide container} \\ w(C_j) & \text{if } C_j \text{ is a tall container} \\ (1-\varepsilon')^2 h(C_j)w(C_j) & \text{if } C_j \text{ is an area container} \end{cases}$$

$$\text{val}_j(i) = p(i)$$
$$w_q(i) = v_q(i)$$

In the above definitions of $s_j(i), M_j, \text{val}_j(i)$ and $w_q(i)$, i varies from 1 to n, j varies from 1 to k and q varies from 1 to d.

If rotations are allowed, the reduction is exactly the same except for the values of $s_j(i)$: If C_j is a tall (resp. wide) container,

$$s_j(i) = \begin{cases} \infty, \text{if } i \text{ can fit neither with rotation nor without rotation} \\ w(i)(resp.\, h(i)), \text{if } i \text{ fits without rotation but not with rotation} \\ h(i)(resp.\, w(i)), \text{if } i \text{ fits with rotation but not without rotation} \\ \min\{w(i),\ h(i)\}, \text{if } i \text{ fits both with and without rotation} \end{cases}$$

If C_j is a large container, we set $s_j(i) = \infty$ if i does not fit in C_j with or without rotations. Otherwise we set $s_j(i)$ to 1. In case of C_j being an area container, $s_j(i)$ is same as the case without rotations.

Let I' denote a subset of I packed in a feasible solution to $\mathcal{C}$. Then I' can also be feasibly assigned to the machines of our Vector-Max-GAP problem: Just assign all the items in a container C_j to the j^{th} machine.

- If C_j is a large container, then the only item packed into it has size 1 in machine M_j and capacity of M_j is also 1 and hence assigning this item to the j^{th} machine is feasible.

- If C_j is a wide (resp. tall) container, the items packed in C_j are wide and stacked up (resp. tall and arranged side by side). Hence their total height (resp. width), which is the total size of items assigned to the j^{th} machine, does not exceed the total height (resp. width) of the container, which is the capacity of the j^{th} machine.

- If C_j is an area container, the total area of items packed in B_j does not exceed $(1-\varepsilon')^2 \cdot a(C_j)$ which yields that the total size of items assigned to the j^{th} machine does not exceed the capacity of the j^{th} machine.

The total weight of all the items assigned to machines does not exceed $\overrightarrow{W}$ (which is equal to the all ones vector) as we did not change $\overrightarrow{v}$ while reducing $\mathcal{C}$ to $\mathcal{G}$. This proves that the optimal value obtainable for $\mathcal{G}$ is at least as much as that of the container packing problem $\mathcal{C}$.

On the other hand, consider any feasible assignment of a subset of items $J \subseteq I$ to the machines in our instance $\mathcal{G}$. Let J_j be the subset of items assigned to the j^{th} machine. We can pack J_j into container C_j in the following way: Since the assignment is feasible, the size of all items in J_j does not exceed the capacity of M_j. If C_j is wide (resp. tall), $\sum_{i \in J_j} h(i) \le h(C_j)$ (resp. $\sum_{i \in J_j} w(i) \le w(C_j)$). Hence, all the items in J_j can be stacked up (resp. arranged side by side) inside the container C_j. If C_j is an area container, then J_j consists of only small items which are ε'-small for C_j and $a(J_j) \le (1-\varepsilon')^2 \cdot a(C_j)$. Hence, by Lemma 5.1, we can pack the set J_j into C_j using NFDH. If an item is assigned to a large container, then it occupies the whole capacity (since item size and machine capacity are both equal to 1) and hence, only a single item can be packed in a large container.

The above arguments prove that the container packing problem is a special instance of the Vector-Max-GAP problem. In Section 5.2, we presented a PTAS for the Vector-Max-GAP problem and hence we also have a PTAS for the container packing problem.

Theorem 5.28. *There exists a PTAS for the container packing problem.*

5.3.5 Algorithm

Our main goal is to search for the container packing structure in Theorem 5.5.

For this we need to guess the set of containers used to pack the items in I'_{Opt} of (5.1). As mentioned in Section 5.3.2, the number of containers used is at most a constant (let this number be denoted by c) and they have to be picked from a set whose cardinality is in poly$(|I|)$ (let this cardinality be denoted by $q(|I|)$). Therefore, the number of guesses required is $\binom{q(|I|)+c}{c}$ which is in turn in poly$(|I|)$.

Once we have the containers, we need to guess which of them are large containers, wide containers, tall containers and area containers. The number of guesses required is at most $\binom{c+4}{c}$ which is a constant.

Then we use the PTAS of the container packing problem with approximation factor $(1 - \varepsilon_{\text{cont}})$, and since the optimal profit of the container packing problem is at least $(1/2 - \varepsilon_{\text{struct}}) \cdot \text{OPT}_{\text{GVKS}}(I)$, by choosing $\varepsilon_{\text{cont}} := \varepsilon$ and $\varepsilon_{\text{struct}} := \varepsilon/2$, we get the desired approximation factor of $(1/2 - \varepsilon)$ for the $(2, d)$ knapsack problem.

Chapter 6

The d-D Hypercube Knapsack Problem

In the d-D Hypercube Knapsack problem, we are given a set of hypercubes, each carrying a profit. The objective is to find a maximum profitable packing into a knapsack which is just a unit hypercube.

We will discuss near-optimal algorithms for two special cases of the d-D Hypercube Knapsack problem in this chapter. More specifically, we devise PTASes for the cardinality case as well as the bounded-density case.

Using more sophisticated techniques, we can design a PTAS for the general d-D Hypercube Knapsack problem where each item can have an arbitrary profit. We will not discuss this general result in this thesis. Please see [JKLS22] for this result.

Let I denote the set of input items. Let ε be the accuracy parameter. For an item (which is a d-D hypercube) i, let $p(i)$ denote its profit and let $\mathrm{vol}(i)$ denote its d-dimensional volume. We can extend this notation of p, vol to a set of items S: $p(S)$ denotes the total profit of all the items in S and $\mathrm{vol}(S)$ denotes the total volume of all the items in S.

6.1 Prior Work

As discussed in Section 1.3.4, for a fixed d, the d-D Hypercube Knapsack is not a generalization of the classical knapsack problem. The (strong) NP-Hardness of this problem for $d = 2$ has been proved by [LTW+90] in 1990. Only recently, in 2015, [LCC15] proved that this problem is strongly NP-Hard for $d \geq 3$. Hence, we can't expect the problem to have an FPTAS for $d \geq 2$ unless P = NP. These hardness results extend to the special cases that we will consider (cardinality case, and bounded-density case).

The first non-trivial approximation algorithm for the problem was given by [Har06], who showed a $\left(\frac{2^d+1}{2^d} + \varepsilon\right)$ approximation algorithm for any $d \geq 2$. Notice how the approximation ratio inches closer to 1 as the dimension increases. Hence, it is natural to expect that there likely exists a PTAS for this problem for any $d \geq 2$. Jansen and Solis-Oba [JS12] showed a PTAS for the special case when $d = 2$ (in this case, the problem is also called the *Square Knapsack* problem). Then, only recently in 2017, [HW17] gave an EPTAS for the square knapsack problem, and this is essentially the best possible algorithm as there can't exist an FPTAS unless $\mathsf{P} = \mathsf{NP}$. For $d \geq 3$ though, there hasn't been any improvement since [Har06]'s result until recently, when [JKLS22] devised a PTAS.

6.2 PTAS for the Bounded-density Case

We consider the *bounded-density* case, where each item i in the input satisfies the condition that

$$\frac{p(i)}{\mathrm{vol}(i)} \in [1, r]$$

where r is an arbitrary but fixed constant.

For a given set of items S, and a positive real v, let $\mathrm{KS}(v, S)$ denote an instance of the classical knapsack problem where each item in $i \in S$ is replaced by a one-dimensional item with size $\mathrm{vol}(i)$ and profit $p(i)$ and the capacity of the knapsack is given by v. Further let $\mathrm{OptKS}(v, S)$ denote the optimum value of the instance $\mathrm{KS}(v, S)$.

Observation 6.1. *Let S be a given input set of items and B be a knapsack with volume v. Then $\mathrm{OptKS}(v, S) \geq \mathrm{Opt}(S)$ (here $\mathrm{Opt}(S)$ denotes the optimal profit obtainable by packing a subset of S into B).*

Proof. This is because the solution to a geometric knapsack instance must satisfy the condition that the total volume of the items packed is at most v. □

Observation 6.2. *For any set of items $S, \mathrm{vol}(S) \leq p(S) \leq r\,\mathrm{vol}(S)$*

We also state the following lemma proved in [BCKS06] which will be used later. For the sake of completeness, we give a proof sketch here.

Lemma 6.3. *Consider a packing of m items into a d-D bin. The remaining unfilled region can be divided into at most $(2m)^d$ hypercuboidal regions.*

Proof. We can assume, without loss of generality, that the corner of one of the items is placed exactly at one of the bin corners. Let's iterate over each dimension $k \in [d]$. Each item has two faces perpendicular to the k^{th} axis. We extend these two faces till they meet the ends of the bin. Thus, we extend at most $2m$ faces for each k. Overall, we finally obtain a grid containing at most $(2m)^d$ cells where the already present items are completely contained in some of these cells. □

6.2.1 NFDH for Small Items

Recall the higher dimensional version of NFDH discussed in Section 2.4.2. Also, recall Lemma 2.6 which can be used to pack small items compactly. We restate the lemma here.

Lemma 6.4. *Let S be a set of d-D hypercubes each with side length at most δ. Consider a d-D hypercuboidal region $\mathcal{R}$ with side lengths $r_1, r_2, \ldots, r_d$ (with each r_i at most 1). Suppose we try to pack S into $\mathcal{R}$ using NFDH. Then we either pack all the items, or the wasted space in $\mathcal{R}$ is at most $\delta(r_1 + r_2 + \cdots + r_d)$.*

We can extend the above lemma to pack small items in multiple hypercuboidal regions.

Lemma 6.5. *Let $\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_\ell$ be a set of hypercuboidal regions and let $v_1, v_2, \cdots, v_\ell$ denote their respective volumes. Assume that all these hypercuboidal regions have all of their dimensions at most 1. Let S be a set of cubes whose lengths are at most δ and consider packing all these items into these hypercuboidal regions. If*

$$\mathrm{vol}(S) \leq \sum_{i=1}^{\ell} v_i - \ell\delta d$$

then using NFDH, we can pack the entire set S into these hypercuboidal regions.

Proof. First try to pack S using NFDH into $\mathcal{R}_1$. If some items are still left, pack them in $\mathcal{R}_2$ and so on. We will prove that all the items will be packed in this manner. For the sake of contradiction, assume that some items are unpacked. By Lemma 6.4, at least $v_i - \delta d$ volume must be packed into every region $\mathcal{R}_i$. Therefore, the volume of all the packed items is at least $\sum_{i=1}^{\ell} v_i - \ell\delta d$ which proves that there can not be any unpacked items. □

6.2.2 Algorithm

We will now proceed to devise a PTAS for the bounded-density case. Let Opt denote an optimal packing. For every item $i \in I$, let s_i denote the length of i. For now assume a small constant $\varepsilon_1 < \varepsilon$: it will be defined during the analysis part. For $k \in \{1, 2, \cdots, \lceil 1/\varepsilon_1 \rceil\}$ define the set

$$M_k = \left[\varepsilon_1^{(2d^2)^k}, \varepsilon_1^{(2d^2)^{k-1}}\right)$$

and the set $N_k = \{i \in I | s_i \in M_k\}$. Clearly, for some $k' \in [\lceil 1/\varepsilon_1 \rceil]$,

$$p(N_{k'} \cap \text{Opt}) \leq \varepsilon_1 \cdot \text{Opt}(I)$$

Define the quantities

$$\varepsilon_{\text{small}} = \varepsilon_1^{(2d^2)^{k'}} \text{ and } \varepsilon_{\text{big}} = \varepsilon_1^{(2d^2)^{k'-1}}$$

Remark 6.6. $\varepsilon_{\text{big}} \leq \varepsilon_1$ *and* $\varepsilon_{\text{small}} = (\varepsilon_{\text{big}})^{2d^2}$.

Define the set $J := I \setminus N_{k'}$ and partition the set J into two classes *small* and *large*: $S := \{j \in J | s_j < \varepsilon_{\text{small}}\}$ and $L := \{j \in J | s_j \geq \varepsilon_{\text{big}}\}$. Although we do not know apriori the set $N_{k'}$, we can "guess" this set since there are only constant number of possibilities for k'. From the above discussion, it is easy to see that $\text{Opt}(J) \geq (1 - \varepsilon_1)\text{Opt}(I)$. Hence, from now on, we concentrate on constructing an almost optimal packing w.r.t. J.

Note that the number of items from L that can fit in the knapsack $\mathcal{B}$ is bounded by $\lfloor 1/\varepsilon_{\text{big}}^d \rfloor$. Let L_{Opt} denote items from L that are packed in $\mathcal{B}$ in the optimal packing and let $m := |L_{\text{Opt}}|$. Again, we do not know apriori the value of m or the set L_{Opt}, but since m is bounded by $\lfloor 1/\varepsilon_{\text{big}}^d \rfloor$ which is a constant, we can "guess" the set L_{Opt} through brute force in polynomial time. We can also find a packing of L_{Opt} into $\mathcal{B}$ in constant time since the number of items we need to pack is no more than a constant.

Now, we have a packing with the large items (these large items are exactly those that are in the optimal packing). Without loss of generality, we can assume that there are gaps left in the bin (if not, we already found the optimal packing). Our next goal is to pack the small items efficiently into these gaps. We divide these gaps into at most $(2m)^d$ hypercuboidal regions using Lemma 6.3. Let us weaken this bound and say that there are at most

$$\lambda := \left(2 \lfloor 1/\varepsilon_{\text{big}}^d \rfloor\right)^d \leq \frac{2^d}{\varepsilon_{\text{big}}^{d^2}} \tag{6.1}$$

empty hypercuboidal regions.

Let V be the total volume of these gaps. First, using an FPTAS for classical knapsack (e.g., [Law77]), select $S' \subseteq S$ such that

$$\text{vol}(S') \leq V - \lambda\varepsilon_{\text{small}}d \text{ and } p(S') \geq (1 - \mu)\text{OptKS}(V - \lambda\varepsilon_{\text{small}}d, S)$$

where μ is a very small constant that will be defined later. Lemma 6.5 states that, using NFDH, we can pack all the items in S' in these gaps.

The packing is complete now and the remaining work is to fix the value of μ, ε_1 and argue that this packing is an almost optimal packing. To summarize, the algorithm is as follows :

1. First, remove a subset of items $N_{k'}$ from I which carries a very small profit compared to $\mathrm{Opt}(I)$. Let us call this new set of items J.

2. Divide J into two sets: S, consisting of items whose side length is $< \varepsilon_{\text{small}}$ and L, consisting of items whose side length is $\geq \varepsilon_{\text{big}}$.

3. First guess the large items that are packed in Opt and pack them into the bin.

4. Gaps are created in the bin. Let V denote the total volume of these gaps. Divide these gaps into at most λ hypercuboidal regions.

5. Select a subset $S' \subseteq S$ such that $p(S') \geq (1-\mu)\mathrm{OptKS}(V - \lambda\varepsilon_{\text{small}}d, S)$ and $\mathrm{vol}(S') \leq V - \lambda\varepsilon_{\text{small}}d$.

6. Pack S' into these hypercuboidal regions using NFDH.

Remark 6.7. *In step 5, we can safely assume that* $\mathrm{vol}(S') \geq V - \lambda\varepsilon_{\text{small}}d - \varepsilon_{\text{small}}^d$*, otherwise we can add small items to* S' *until this condition is met without violating the condition* $\mathrm{vol}(S') \leq V - \lambda\varepsilon_{\text{small}}d$*. If there are not enough small items to add to* S'*, it would mean that we already have the exact optimal solution.*

6.2.3 Analysis

Let $L_{\text{Opt}}, S_{\text{Opt}}$ respectively denote the set of large and small items that are packed in Opt. Let $L_{\text{Alg}}, S_{\text{Alg}}$ respectively denote the large and small items that are packed by our algorithm. By the construction of our algorithm, $L_{\text{Alg}} = L_{\text{Opt}}$. Let $\mathrm{Alg}(J) = p(L_{\text{Alg}}) + p(S_{\text{Alg}})$ denote the profit packed by our algorithm. Also, note that $\mathrm{Opt}(J) = p(L_{\text{Opt}}) + p(S_{\text{Opt}})$.

First we observe that

$$p(S_{\text{Opt}}) \leq \mathrm{OptKS}(V, S) \leq \mathrm{OptKS}(V - \lambda\varepsilon_{\text{small}}d, S) + (\lambda\varepsilon_{\text{small}}d + \varepsilon_{\text{small}}^d)r$$

The first inequality follows from Observation 6.1 and the second inequality is because, starting from an optimal solution to $\mathrm{KS}(V, S)$ we can remove a few items of volume at most $\lambda\varepsilon_{\text{small}}d + \varepsilon_{\text{small}}^d$ to get a feasible solution to $\mathrm{KS}(V - \lambda\varepsilon_{\text{small}}d, S)$. These removed items will have profit at most $(\lambda\varepsilon_{\text{small}}d + \varepsilon_{\text{small}}^d)r$.

$$\begin{aligned}
\mathrm{Alg}(J) &= p(L_{\mathrm{Alg}}) + p(S_{\mathrm{Alg}}) \\
&\geq p(L_{\mathrm{Alg}}) + (1-\mu)\mathrm{OptKS}(V - \lambda\varepsilon_{\mathrm{small}}d, S) \\
&\geq p(L_{\mathrm{Alg}}) + (1-\mu)\Big(p(S_{\mathrm{Opt}}) - (\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d)r\Big)
\end{aligned}$$

Now, to show that $\mathrm{Alg}(J)$ is very close to $\mathrm{Opt}(J)$,

$$\begin{aligned}
\frac{\mathrm{Alg}(J)}{\mathrm{Opt}(J)} &\geq \frac{p(L_{\mathrm{Alg}}) + (1-\mu)\Big(p(S_{\mathrm{Opt}}) - (\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d)r\Big)}{\mathrm{Opt}(J)} \\
&= \frac{p(L_{\mathrm{Opt}}) + p(S_{\mathrm{Opt}}) - \mu\cdot p(S_{\mathrm{Opt}}) - (1-\mu)(\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d)r}{\mathrm{Opt}(J)} \\
&= 1 - \frac{\mu\cdot p(S_{\mathrm{Opt}}) + (1-\mu)(\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d)r}{\mathrm{Opt}(J)} \\
&\geq 1 - \mu - \frac{(1-\mu)(\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d)r}{\mathrm{Opt}(J)} \\
&\geq 1 - \mu - \frac{(1-\mu)(\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d)r}{1 - (\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d)} \\
&\quad \text{(since } \mathrm{vol}(J) > 1 - (\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d) \text{ (Remark 6.7) and } p(J) \geq \mathrm{vol}(J)) \\
&\geq (1-\mu)\Big(1 - (1+r)(\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d)\Big)
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathrm{Alg}(J) &\geq (1-\mu)\Big(1 - (1+r)(\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d)\Big)\mathrm{Opt}(J) \\
&\geq (1-\varepsilon_1)(1-\mu)\Big(1 - (1+r)(\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d)\Big)\mathrm{Opt}(I)
\end{aligned}$$

If we can make sure that each of $\varepsilon_1, \mu, (1+r)(\lambda\varepsilon_{\mathrm{small}}d + \varepsilon_{\mathrm{small}}^d)$ is less than $\varepsilon/3$ then we will get the desired result. First let's choose

$$\varepsilon_1 = \frac{\varepsilon}{6(1+r)} \text{ and } \mu = \frac{\varepsilon}{3}$$

From Eq. (6.1) and from Remark 6.6,

$$\lambda\varepsilon_{\text{small}} d \leq \frac{2^d}{\varepsilon_{\text{big}}^{d^2}} (\varepsilon_{\text{big}})^{2d^2} d = 2^d \varepsilon_{\text{big}}^{d^2} d \leq 2^d \varepsilon_1^{d^2} d \leq \varepsilon_1 = \frac{\varepsilon}{6(1+r)}$$

Similarly, we can prove that $\varepsilon_{\text{small}}^d \leq \frac{\varepsilon}{6(1+r)}$. Overall, we obtain that

$$(1+r)(\lambda\varepsilon_{\text{small}} d + \varepsilon_{\text{small}}^d) \leq \frac{\varepsilon}{3}$$

This completes the PTAS for the bounded-density case.

6.3 PTAS for the Cardinality Case

In this section, we will present a PTAS for the d-D Hypercube Knapsack problem in the special case when each item has unit profit.

We will again use NFDH to pack small items. Hence, we will restate the lemma.

Lemma 6.8. *Let S be a set of d-D hypercubes each with side length at most δ. Consider a d-D hypercuboidal region $\mathcal{R}$ with side lengths $r_1, r_2, \dots, r_d$ (with each r_i at most 1). Suppose we try to pack S into $\mathcal{R}$ using NFDH. Then we either pack all the items, or the wasted space in $\mathcal{R}$ is at most $\delta(r_1 + r_2 + \cdots + r_d)$.*

Let I denote the set of input items and let $n := |I|$. Define $\varepsilon_1 := \varepsilon/(3d)$. Consider any optimal packing Opt. Note that since we are in the cardinality case, for any set of items S, $p(S) = |S|$. If $|\text{Opt}| \leq (1/\varepsilon_1)^{d+1}$, then we can guess $|\text{Opt}|$ in constant time. Once we know $|\text{Opt}|$, in time

$$\binom{n}{\lceil (1/\varepsilon_1)^{d+1} \rceil} = O_\varepsilon\left(n^{(1/\varepsilon_1)^{d+2}}\right)$$

we can guess the items in Opt. Since $|\text{Opt}|$ is bounded by a constant, we can find an exact packing of Opt in polynomial time.

So, from now on, assume that

$$|\text{Opt}| > \left(\frac{1}{\varepsilon_1}\right)^{d+1}$$

If an item has side length at least ε_1, we call it a *large* item; otherwise it is *small*.

Let L_{Opt} denote the set of large items in Opt and let S_{Opt} denote the set of small items in

Opt. Since a large item has volume at least ε_1^d, we have that

$$p(L_{\text{Opt}}) \le \frac{1}{\varepsilon_1^d} \le \varepsilon_1 p(\text{Opt})$$

This in turn implies that

$$p(S_{\text{Opt}}) = p(\text{Opt}) - p(L_{\text{Opt}}) \ge (1-\varepsilon_1)p(\text{Opt}) \tag{6.2}$$

Order the small items in non-increasing order of profit density (or non-decreasing order of side lengths). Select the largest prefix S' such that

$$\text{vol}(S') \le 1 - \varepsilon_1 d$$

By Lemma 6.8, using NFDH, we can pack the entire set S' in the knapsack.

If $S' = S$, then from Eq. (6.2), we directly we have that the profit packed is at least

$$p(S) \ge (1-\varepsilon_1)p(\text{Opt}) \ge (1-\varepsilon)p(\text{Opt})$$

On the other hand, if $S' \neq S$, since any small item has volume at most ε_1^d, we have that

$$\text{vol}(S') \ge (1 - \varepsilon_1 d - \varepsilon_1^d)$$

by the maximality of S'. Moreover, since S' contains the items with the highest profit density whose volume equals $\text{vol}(S')$, we have that

$$\begin{aligned}
p(S') &\ge (1-\varepsilon_1 d - \varepsilon_1^d)\frac{p(S_{\text{Opt}})}{\text{vol}(S_{\text{Opt}})} \\
&\ge (1-\varepsilon_1 d - \varepsilon_1^d)(1-\varepsilon_1)p(\text{Opt}) \\
&\ge \left(1 - \frac{\varepsilon}{3} - \varepsilon_1^d\right)\left(1-\frac{\varepsilon}{3}\right)p(\text{Opt}) && (\text{since } \varepsilon_1 = \varepsilon/(3d)) \\
&\ge \left(1 - \frac{2\varepsilon}{3}\right)\left(1-\frac{\varepsilon}{3}\right)p(\text{Opt}) \\
&\ge (1-\varepsilon)p(\text{Opt})
\end{aligned}$$

This completes the PTAS for the cardinality case.

Chapter 7

Conclusion

In this thesis, we studied several variants of the classical bin packing and knapsack problems.

First we studied online bin packing under two stochastic models: *(i)* the i.i.d. model, *(ii)* the random-order model. For the first setting, we devised a meta-algorithm which takes any offline algorithm $\mathcal{A}_\alpha$ with an AAR of α, and produces an online algorithm with an ECR of $(\alpha + \varepsilon)$. This shows that online bin packing under the i.i.d. model and offline bin packing are almost equivalent. Using any AFPTAS as $\mathcal{A}_\alpha$ results in an online algorithm with an ECR of $(1 + \varepsilon)$ for any constant $\varepsilon > 0$.

Then, we analyzed the well-known Best-Fit algorithm under the random-order model. First, we proved that the ECR of Best-Fit is equal to one if all the item sizes are greater than $1/3$. Then, we improved the analysis of the Best-Fit from 1.5 to ≈ 1.4941, for a very hard special case where the item sizes lie in the range $(1/4, 1/2]$.

Then we moved to the knapsack regime. For the 3-D Knapsack problem, if rotations are allowed, we designed a $(31/7 + \varepsilon)$ approximation algorithm, thus beating the current best ratio of $(5 + \varepsilon)$. If rotations are not allowed, we gave a simpler $(7 + \varepsilon)$-approximate algorithm which matches the current best approximation ratio. For the problem of maximizing the volume packed, when rotations are allowed, we devised a $(3 + \varepsilon)$-approximate algorithm. We gave improved approximation algorithms for the cardinality case as well.

Then, we studied the rectangle knapsack problem with vector constraints, also denoted as $(2, d)$ Knapsack. For this problem, we devised a $(2 + \varepsilon)$-approximate algorithm using the corridor decomposition technique.

Finally, we devised PTASes for the d-D Hypercube Knapsack problem in the special cases of cardinality and bounded-density.

7.1 Open Problems

Classical Bin Packing. Despite being very well-studied over the past four decades, there exist several interesting open questions on bin packing. [HR17b] gave an algorithm which packs in $\text{Opt} + O(\log \text{Opt})$ number of bins, but an interesting open question is whether it is possible to pack in $\text{Opt} + O(1)$ number of bins. The famous textbook by Shmoys and Williamson [WS11] mentions this problem as one of the most important open problems in the field of approximation algorithms.

Online Bin Packing. The current best lower bound on the competitive ratio (CR) of any online algorithm is ≈ 1.54278 by [BBD+19]. On the other hand, the current best algorithm by [BBD+18] has an approximation ratio of ≤ 1.57829. Closing the gap between these upper and lower bounds is an interesting open question.

Online Bin Packing under the i.i.d. Model. In Chapter 3, we gave an algorithm with an ECR of $(1 + \varepsilon)$. Whether an algorithm with ECR exactly equal to one is possible or not is an interesting open question.

Online Bin Packing under the Random-order Model. Fischer in his thesis [Car19] gave an exponential time, randomized algorithm which achieves a random-order ratio of $(1 + \varepsilon)$. It is interesting to see if we can reduce the runtime to polynomial time, while maintaining near-optimality. Another important problem is to deduce the exact ECR of Best-Fit in this model, which is conjectured to be ≈ 1.15 at present. The current best lower bound and upper bound are 1.1 and 1.5, respectively. Hence, a good direction would be to tighten this gap.

Geometric Bin Packing. Another important and well-studied variant of bin packing is the d-D geometric bin packing where items are d-D hypercuboids. For $d \geq 3$, the current best approximation algorithm, given by Caprara [Cap08], has an AAR of 1.69^{d-1}. For the three-dimensional case, this ratio turns out to be ≈ 2.86. Since three-dimensional packing is very relevant in practice, it would be an interesting task to improve this ratio. For general d, it would be interesting to see if an approximation factor of $O(\text{poly}(d))$ is possible or not.

Rectangle Knapsack. For the rectangle knapsack problem, where each item is just a rectangle, existence of a PTAS is conjectured. In fact, a recent QPTAS for this problem by [AW15] strengthens the belief. However, the current best approximation ratio is 1.89 [GGH+17], which is far from a PTAS.

3-D Knapsack. In Chapter 4, we improved the current best approximation ratio from $(5 + \varepsilon)$ to $(31/7 + \varepsilon)$ when rotations are allowed, and gave a simpler $(7 + \varepsilon)$-approximate algorithm when rotations are not allowed, matching the current best factor. Both algorithms are constructive in nature. However, most state-of-the-art results on packing are non-constructive, which first modify an optimal packing into a polynomial-time searchable structure. Hence, one promising approach to improve our ratios would be to derive good structural results about an optimal packing. The special case of bounded profit density has links with 3-D geometric bin packing: We can show that any algorithm for the case of bounded profit density with a AR better than 2.86 will lead to an algorithm for 3-D geometric bin packing with an AAR better than 2.86, thus beating Caprara's current best factor.

Geometric Knapsack. For the d-D geometric knapsack problem, [Sha21] gave a $(3^d + \varepsilon)$-approximate algorithm. Improving this ratio would be an interesting task. The special case of bounded-density is also important since any approximation ratio better than 1.69^{d-1} would yield a better approximation ratio for the d-D geometric bin packing problem discussed above.

Bibliography

[AKL21a] Susanne Albers, Arindam Khan, and Leon Ladewig. Best fit bin packing with random order revisited. *Algorithmica*, 83(9):2833–2858, 2021. 4, 11, 25, 28, 51

[AKL21b] Susanne Albers, Arindam Khan, and Leon Ladewig. Improved online algorithms for knapsack and GAP in the random order model. *Algorithmica*, 83(6):1750–1785, 2021. 25

[AW15] Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the two-dimensional geometric knapsack problem. In *SODA*, pages 1491–1505, 2015. 89, 91, 116

[BBD$^+$18] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new and improved algorithm for online bin packing. In *ESA*, volume 112, pages 5:1–5:14, 2018. 24, 116

[BBD$^+$19] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new lower bound for classic online bin packing. In *WAOA*, volume 11926, pages 18–28. Springer, 2019. 24, 116

[BBG10] János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. In *WAOA*, 2010. 3

[BC81] 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. 28

[BCJ$^+$09] Nikhil Bansal, Alberto Caprara, Klaus Jansen, Lars Prädel, and Maxim Sviridenko. A structural lemma in 2-dimensional packing, and its implications on approximability. In *ISAAC*, pages 77–86, 2009. 22

[BCKS06] 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. 22, 108

[BEK16] Nikhil Bansal, Marek Eliás, and Arindam Khan. Improved approximation for vector bin packing. In *SODA*, pages 1561–1579, 2016. 28

[BGK00] Jozsef Bekesi, Gabor Galambos, and Hans Kellerer. A 5/4 linear time bin packing algorithm. *Journal of Computer and System Sciences*, 60(1):145–160, 2000. 23

[BGK11] Anand Bhalgat, Ashish Goel, and Sanjeev Khanna. *Improved Approximation Results for Stochastic Knapsack Problems*, pages 1647–1665. 2011. 4

[BJL+84] Jon Louis Bentley, David S Johnson, Frank Thomson Leighton, Catherine C McGeoch, and Lyle A McGeoch. Some unexpected expected behavior results for bin packing. In *STOC*, pages 279–288, 1984. 24

[Cap08] Alberto Caprara. Packing d-dimensional bins in d stages. *Mathematics of Operations Research*, 33:203–215, 2008. 116

[Car19] Carsten Oliver Fischer. *New Results on the Probabilistic Analysis of Online Bin Packing and its Variants*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, December 2019. 25, 27, 50, 116

[CGJT80] Edward G. Coffman, Jr, Michael R. Garey, David S. Johnson, and Robert E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9:808–826, 1980. 21

[CJJLS93] Edward G Coffman Jr, David S Johnson, George S Lueker, and Peter W Shor. Probabilistic analysis of packing and related partitioning problems. *Statistical Science*, 8(1):40–47, 1993. 51

[CJJSW97] Edward G Coffman Jr, David S Johnson, Peter W Shor, and Richard R Weber. Bin packing with discrete item sizes, part ii: Tight bounds on first fit. *Random Structures & Algorithms*, 10(1-2):69–101, 1997. 24

[CJK+06] Janos Csirik, David S Johnson, Claire Kenyon, James B Orlin, Peter W Shor, and Richard R Weber. On the sum-of-squares algorithm for bin packing. *Journal of the ACM (JACM)*, 53(1):1–65, 2006. 24, 25

[CJSHY80] Edward G Coffman Jr, Kimming So, Micha Hofri, and AC Yao. A stochastic model of bin-packing. *Information and Control*, 44(2):105–115, 1980. 24

[CS88] K. L. Clarkson and P. W. Shor. Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, SCG '88, page 12–17. Association for Computing Machinery, 1988. 4

[DEH+17] Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. Stochastic k-server: How should uber work? In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017. 4

[DGV08] Brian C Dean, Michel X Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Mathematics of Operations Research*, 33(4):945–964, 2008. 25

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. 1

[DH09] Nikhil R. Devanur and Thomas P. Hayes. The adwords problem: Online keyword matching with budgeted bidders under random permutations. In *Proceedings of the 10th ACM Conference on Electronic Commerce*, EC '09, page 71–78, New York, NY, USA, 2009. Association for Computing Machinery. 4

[DHJ+07] Florian Diedrich, Rolf Harren, Klaus Jansen, Ralf Thöle, and Henning Thomas. Approximation algorithms for 3d orthogonal knapsack. In Jin-Yi Cai, S. Barry Cooper, and Hong Zhu, editors, *Theory and Applications of Models of Computation*, pages 34–45, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. iii, 11, 60, 62, 68, 71, 77

[dlVL81] W Fernandez de la Vega and George S Lueker. Bin packing can be solved within 1+epsilon in linear time. *Combinatorica*, 1(4):349–355, 1981. 11, 24, 26, 28

[EPR11] Friedrich Eisenbrand, Dömötör Pálvölgyi, and Thomas Rothvoß. Bin packing via discrepancy of permutations. In *SODA*, pages 476–481, 2011. 24

[FC84] A. M. Frieze and M. R. B. Clarke. Approximation algorithms for the m-dimensional $0-1$ knapsack problem: worst-case and probabilistic analyses. *EJOR*, 15:100–109, 1984. 81

[Fer89] Thomas S Ferguson. Who solved the secretary problem? *Statistical science*, 4(3):282–289, 1989. 25

[FMMM09] Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and Shan Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *FOCS*, pages 117–126, 2009. 25

[FR16] Carsten Fischer and Heiko Röglin. Probabilistic analysis of the dual next-fit algorithm for bin covering. In *LATIN*, pages 469–482, 2016. 25

[FR18] Carsten Fischer and Heiko Röglin. Probabilistic analysis of online (class-constrained) bin packing and bin covering. In *LATIN*, volume 10807, pages 461–474. Springer, 2018. 25

[GG63] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem-part ii. *Operations Research*, 11(6):863–888, 1963. 1

[GGH$^+$17] Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via l-packings. In *FOCS*, pages 260–271, 2017. 80, 88, 93, 102, 116

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the theory of NP-completeness*. Freeman NY, 1979. 2

[GKL22] Anupam Gupta, Gregory Kehne, and Roie Levin. Random order online set cover is as easy as offline. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1253–1264, 2022. 4

[GKNS21] Anupam Gupta, Amit Kumar, Viswanath Nagarajan, and Xiangkun Shen. Stochastic load balancing on unrelated machines. *Mathematics of Operations Research*, 46(1):115–133, 2021. 25

[GKR12] Anupam Gupta, Ravishankar Krishnaswamy, and R Ravi. Online and stochastic survivable network design. *SIAM Journal on Computing*, 41(6):1649–1672, 2012. 25

[GS20] Anupam Gupta and Sahil Singla. Random-order models. In *Beyond the Worst-Case Analysis of Algorithms*, pages 234–258. Cambridge University Press, 2020. 25

[GS21] Anupam Gupta and Sahil Singla. *Random-Order Models*, page 234–258. Cambridge University Press, 2021. 4

[Har06] Rolf Harren. Approximating the orthogonal knapsack problem for hypercubes. In *ICALP*, pages 238–249, 2006. 107, 108

[HJPvS14] Rolf Harren, Klaus Jansen, Lars Prädel, and Rob van Stee. A $(5/3 + \varepsilon)$-approximation for strip packing. *Comput. Geom.*, 47(2):248–267, 2014. 61

[HR17a] Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *SODA*, pages 2616–2625, 2017. 11, 24, 28

[HR17b] Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *SODA*, pages 2616–2625, 2017. 116

[HW17] Sandy Heydrich and Andreas Wiese. Faster approximation schemes for the two-dimensional knapsack problem. In *SODA*, pages 79–98, 2017. 108

[IK75] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, October 1975. 80

[JCRZ08] Edward G Coffman Jr, János Csirik, Lajos Rónyai, and Ambrus Zsbán. Random-order bin packing. *Discret. Appl. Math.*, 156(14):2810–2816, 2008. 25

[JDU$^+$74] David S Johnson, Alan Demers, Jeffrey D Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on computing*, 3(4):299–325, 1974. 23, 55

[JG85] David S Johnson and Michael R Garey. A 71/60 theorem for bin packing. *J. Complex.*, 1(1):65–106, 1985. 23, 28

[JKLS22] Klaus Jansen, Arindam Khan, Marvin Lira, and K. V. N. Sreenivas. A PTAS for packing hypercubes into a knapsack. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229, pages 78:1–78:20, 2022. 107, 108

[Joh73] David S Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973. 28

[Joh74] David S Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8(3):272–314, 1974. 25

[JS12] Klaus Jansen and Roberto Solis-Oba. Packing squares with profits. *SIAM J. Discrete Math.*, 26(1):263–279, 2012. 108

[JZ04] Klaus Jansen and Guochuan Zhang. On rectangle packing: maximizing benefits. In *SODA*, pages 204–213, 2004. 61

[Kar72] R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. 1

[Ken96] Claire Kenyon. Best-fit bin-packing with random order. In *SODA*, pages 359–364, 1996. 25

[KK82] Narendra Karmarkar and Richard M Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *FOCS*, pages 312–320, 1982. 24

[KTRV14] Thomas Kesselheim, Andreas Tönnis, Klaus Radke, and Berthold Vöcking. Primal beats dual on online packing lps in the random-order model. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 303–312, New York, NY, USA, 2014. Association for Computing Machinery. 4

[Law77] Eugene L. Lawler. Fast approximation algorithms for knapsack problems. In *FOCS*, pages 206–213, 1977. 80, 110

[LCC15] Yiping Lu, Danny Z Chen, and Jianzhong Cha. Packing cubes into a cube in $(d > 3)$-dimensions. In *COCOON*, pages 264–276. Springer, 2015. 13, 107

[LL85a] Chan C Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32(3):562–572, 1985. 6

[LL85b] Chan C Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32(3):562–572, July 1985. 24, 28

[LL87] Chan C Lee and Der-Tsai Lee. Robust on-line bin packing algorithms. *Technical Report, Northwestern University*, 1987. 24

[LS89] Tom Leighton and Peter Shor. Tight bounds for minimax grid matching with applications to the average case analysis of algorithms. *Combinatorica*, 9(2):161–187, 1989. 24

[LTW+90] Joseph Y. T. Leung, Tommy W. Tam, C. S. Wong, Gilbert H. Young, and Francis Y. L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990. 13, 107

[MLG04] Omid Madani, Daniel J. Lizotte, and Russell Greiner. The budgeted multi-armed bandit problem. In John Shawe-Taylor and Yoram Singer, editors, *Learning Theory*, pages 643–645, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. 4

[Mur88] Frank D Murgolo. Anomalous behavior in bin packing algorithms. *Discret. Appl. Math.*, 21(3):229–243, 1988. 25, 47

[MY11] Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *STOC*, pages 597–606, 2011. 25

[NNN12] Alantha Newman, Ofer Neiman, and Aleksandar Nikolov. Beck's three permutations conjecture: A counterexample and some consequences. In *FOCS*, pages 253–262, 2012. 24

[Ram89] Prakash V Ramanan. Average-case analysis of the smart next fit algorithm. *Inf. Process. Lett.*, 31(5):221–225, 1989. 25

[Rhe94] W. T. Rhee. Inequalities for bin packing-iii. *Optimization*, 29(4):381–385, 1994. 32

[RT88] Wansoo T Rhee and Michel Talagrand. Exact bounds for the stochastic upward matching problem. *Transactions of the American Mathematical Society*, 307(1):109–125, 1988. 26

[RT93a] WanSoo T. Rhee and Michel Talagrand. On-line bin packing of items of random sizes, ii. *SIAM Journal on Computing*, 22(6):1251–1256, 1993. 4

[RT93b] Wansoo T Rhee and Michel Talagrand. On-line bin packing of items of random sizes, ii. *SIAM Journal on Computing*, 22(6):1251–1256, 1993. 24, 26, 27, 32

[RT93c] Wansoo T Rhee and Michel Talagrand. On line bin packing with items of random size. *Mathematics of Operations Research*, 18(2):438–445, 1993. 30, 32, 39

[Sha21] Eklavya Sharma. Harmonic algorithms for packing d-dimensional cuboids into bins. In *41st IARCS Annual Conference on Foundations of Software Technology* *and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213, pages 32:1–32:22, 2021. 117

[Sho86] Peter W Shor. The average-case analysis of some on-line algorithms for bin packing. *Combinatorica*, 6(2):179–200, 1986. 24, 39, 51

[Ski99] Steven S. Skiena. Who is interested in algorithms and why? lessons from the stony brook algorithms repository. *SIGACT News*, 30(3):65–74, sep 1999. 1

[Spe94] Joel Spencer. *Ten lectures on the probabilistic method*. SIAM, 1994. 24

[Ste97] A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997. 61

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011. 116