Improved Approximation Algorithms for Packing and Scheduling Problems

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurswissenschaften
(Dr.-Ing.)

der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

MSc ETH Math. Stefan Erich Julius Kraft
Januar 2016

Stefan Erich Julius Kraft: *Improved Approximation Algorithms for Packing and Scheduling Problems*, © Januar 2016

Gutachter:
1. Gutachter: Prof. Dr. Klaus Jansen, Christian-Albrechts-Universität zu Kiel
2. Gutachter: Prof. Dr. Peter Widmayer, ETH Zürich

Datum der mündlichen Prüfung: 25. November 2015
Datum der Genehmigung des Drucks: 25. November 2015

Dedicated to my parents
and
the loving memory of Sandra (1920 – 2015) and Enrico (1925 – 2013).

Zusammenfassung

Diese Dissertation stellt Approximationsalgorithmen für geometrische Packungs- und Schedulingprobleme vor. Zuerst werden das Bin Packing-Problem und seine Verallgemeinerung, das Variable-sized Bin Packing-Problem, betrachtet. Bei Bin Packing muss eine Menge an Gegenständen (Items) in eine minimale Anzahl an Bins einer Größe gepackt werden, ohne die Kapazität (Größe) der einzelnen Bins zu überschreiten. Bei Variable-sized Bin Packing sind mehrere Bingrößen gegeben, wobei eine beliebige Anzahl an Bins jeder Größe gewählt werden darf. Ziel ist es, die Gegenstände in die Bins zu packen und gleichzeitig das Gesamtvolumen der benutzten Bins zu minimieren. Wir stellen für beide Probleme Algorithmen vor, die eine Lösung mit Wert höchstens $(1+\varepsilon)\mathrm{OPT}(I) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$ für jedes $\varepsilon > 0$ und jede Probleminstanz I finden. Dabei bezeichnet $\mathrm{OPT}(I)$ den optimalen Wert für die Instanz I.

Unsere Algorithmen müssen die unbeschränkte (unbounded) Variante des Knapsack-Problems (Rucksackproblems) und des Knapsack-Problems mit invers proportionalen Profiten (Knapsack Problem with Inversely Proportional Profits, KPIP) als Unterprobleme lösen. Beim Knapsack-Problem ist eine Knapsack-Größe zusammen mit einer Menge an Items gegeben, wobei jedes Item einen Profit und eine Größe hat. Ziel ist es, eine Menge an Items mit maximalem Profit zu wählen, die immer noch in den Knapsack passt. In der normalen 0-1 Variante des Knapsack-Problems kann jedes Item nur einmal gewählt werden. Bei der beschränkten Variante kann von jedem Item eine bestimmte Anzahl an Kopien genommen werden. Die unbeschränkte Variante erlaubt die unbeschränkte Anzahl Kopien jedes Items.
KPIP ist eine Verallgemeinerung des Knapsack-Problems, in der wir nicht nur eine, sondern mehrere Knapsack-Größen haben. Eine Knapsack-Größe muss zusammen mit einer entsprechenden Auswahl an Items gewählt werden, die in den Knapsack passt. Allerdings ist der Profit eines Items invers proportional zur Größe des Knapsacks, in den das Item gepackt wird. Das macht es schwierig, die richtige Knapsack-Größe zu wählen, die den Profit über alle Knapsack-Größen maximiert. Wie beim Knapsack-Problem gibt es bei KPIP ebenfalls die Varianten 0-1, beschränkt und unbeschränkt.
Wir stellen zuerst Algorithmen für alle drei Varianten von KPIP vor. Sie finden Lösungen mit einem Mindestprofit von $(1-\varepsilon)\mathrm{OPT}(I)$ für jedes $\varepsilon > 0$ und jede Probleminstanz I, zudem sind die Algorithmen schneller als der natürliche Ansatz, für jede Knapsack-Größe das entsprechende Knapsack-Problem einzeln zu lösen. Danach

stellen wir einen Algorithmus für die unbeschränkte Variante des Knapsack-Problems vor, ebenfalls mit einem Mindestprofit von $(1-\varepsilon)\mathrm{OPT}(I)$ für jedes $\varepsilon > 0$ und jede Instanz I. Er ist schneller und benötigt weniger Speicherplatz als zuvor bekannte Algorithmen. Schließlich kombinieren wir den Ansatz für KPIP und für das unbeschränkte Knapsack-Problem, um einen Algorithmus für die unbeschränkte Variante von KPIP zu finden, der wiederum eine kleinere Zeit- und Speicherkomplexität hat und dessen Lösung für Instanz I und $\varepsilon > 0$ einen Mindestprofit von $(1-\varepsilon)\mathrm{OPT}(I)$ besitzt. All diese Resultate verbessern die Laufzeit für den Bin Packing- und Variable-sized Bin Packing-Algorithmus.

Als Korollar wird die Laufzeit für einen Strip Packing-Algorithmus der Güte $(1+\varepsilon)\mathrm{OPT}(I) + \mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ verbessert. Bei Strip Packing ist das Ziel, eine Menge an Rechtecken in einen Streifen mit unbeschränkter Höhe zu packen, sodass sich die Rechtecke nicht überlappen und gleichzeitig die Höhe der Packung minimiert wird.

Schließlich wird das Scheduling-Problem auf unabhängigen Maschinen (Scheduling on Unrelated Machines) betrachtet, bei dem eine Menge an Maschinen und Jobs gegeben ist. Jeder Job hat auf einer Maschine eine Ausführungszeit, wobei diese auf jeder Maschine unterschiedlich sein kann. Ziel ist es, die Jobs auf die Maschinen so zu verteilen, dass die Laufzeit der am längsten laufenden Maschine minimiert wird. Wir betrachten den Spezialfall, in dem die Anzahl der Maschinentypen K konstant ist: Ein Job hat auf jeder Maschine desselben Typs die gleiche Ausführungszeit. Wir stellen für den Spezialfall einen Algorithmus vor, der eine Lösung mit Wert höchstens $(1+\varepsilon)\mathrm{OPT}(I)$ für jedes $\varepsilon > 0$ und jede Instanz I findet. Der Algorithmus ist schneller als das zuvor bekannte Verfahren für allgemeines (aber konstantes) K.

Abstract

This thesis presents approximation algorithms for geometric packing and scheduling problems. First, the Bin Packing Problem and its generalization, the Variable-sized Bin Packing Problem, are considered. In Bin Packing, a set of items has to be packed into a minimum number of bins of one size, without exceeding the size of any bin. In Variable-sized Bin Packing, several bin sizes are given, and an arbitrary number of bins of every size can be chosen. The objective is to pack the items into bins while minimizing the total volume of the bins used. We present algorithms for both problems that find a solution of value at most $(1+\varepsilon)\mathrm{OPT}(I) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$ for every $\varepsilon > 0$ and every problem instance I. In this thesis, $\mathrm{OPT}(I)$ denotes the optimum value for the instance I.

Our algorithms have to solve the unbounded variant of the Knapsack Problem and of the Knapsack Problem with Inversely Proportional Profits (KPIP) as subproblems. In the Knapsack Problem, a knapsack size is given together with a set of items, each with a profit and a size. The objective is to choose a subset of items with a maximum total profit that still fits into the knapsack. In the normal 0-1 variant of the Knapsack Problem, an item can be chosen only once. In the bounded variant, an individual bounded number of copies can be taken of every item. The unbounded variant allows for an infinite number of copies of every item.

KPIP is a generalization of the Knapsack Problem in which we have not only one, but several knapsack sizes. One knapsack size has to be chosen together with a corresponding subset of items that fits into the knapsack. However, the profit of an item is inversely proportional to the size of the knapsack into which it has been packed. This makes it non-trivial to choose the right knapsack size that maximizes the profit over all knapsack sizes. There are the 0-1, the bounded, and the unbounded variant of KPIP similar to the Knapsack Problem.

We first present algorithms for every of the three variants of KPIP. They find solutions of value at least $(1-\varepsilon)\mathrm{OPT}(I)$ for every $\varepsilon > 0$ and problem instance I, and they are moreover faster than the natural approach to separately solve for every knapsack size the corresponding Knapsack Problem. Second, we present an algorithm for the Unbounded Knapsack Problem with a solution of value at least $(1-\varepsilon)\mathrm{OPT}(I)$ for every $\varepsilon > 0$ and instance I. It is faster and needs less storage space than previously known algorithms. Finally, we combine the approaches of the KPIP and of the Unbounded

Knapsack algorithms to get an algorithm for the Unbounded KPIP that has again a better time and space complexity and whose solution to I has for $\varepsilon > 0$ a value of at least $(1 - \varepsilon)\mathrm{OPT}(I)$. All these results improve the running time for our Bin Packing and Variable-sized Bin Packing algorithms.

As a corollary, we also improve the running time for a Strip Packing algorithm of solution quality $(1 + \varepsilon)\mathrm{OPT}(I) + \mathcal{O}(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$. The goal of Strip Packing is to pack a set of rectangles into a strip of infinite height so that the rectangles do not overlap and the height of the packing is minimized.

Finally, Scheduling on Unrelated Machines is considered where we are given a set of machines and a set of jobs. Each job has a processing time on a machine, where the processing time of a job may be different on each machine. The goal is to distribute the jobs to the machines so that the total processing time of the longest-running machine is minimized. We consider the case with a constant number K of machine types: one job has the same processing time on every machine of the same type. We present an algorithm for this special case that finds a solution of value at most $(1 + \varepsilon)\mathrm{OPT}(I)$ for every $\varepsilon > 0$ and instance I. The algorithm has a better running time than the previously known algorithm for general (but constant) K.

Publications

The results in this thesis have appeared previously in the following publications:

[26] J. C. Gehrke, K. Jansen, S. E. J. Kraft, and J. Schikowski. *A PTAS for Scheduling Unrelated Machines of Few Different Types*. Tech. rep. 1506. Christian-Albrechts-Universität zu Kiel, 2015. ISSN: 2192-6247.

[27] J. C. Gehrke, K. Jansen, S. E. J. Kraft, and J. Schikowski. "A PTAS for Scheduling Unrelated Machines of Few Different Types". In: *Proceedings of 42nd International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2016*. Ed. by R. Freivalds, G. Engels, and B. Catania. Vol. 9587. LNCS: Advanced Research in Computing and Software Science. Springer, 2016. In press.

[42] K. Jansen and S. Kraft. "An Improved Approximation Scheme for Variable-Sized Bin Packing". In: *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science, MFCS 2012*. Ed. by B. Rovan, V. Sassone, and P. Widmayer. Vol. 7464. LNCS. Springer, 2012, pp. 529–541. DOI: 10.1007/978-3-642-32589-2_47.

[43] K. Jansen and S. Kraft. "An Improved Knapsack Solver for Column Generation". In: *Proceedings of the 8th International Computer Science Symposium in Russia, CSR 2013*. Ed. by A. Bulatov and A. Shur. Vol. 7913. LNCS. Springer, 2013, pp. 12–23. DOI: 10.1007/978-3-642-38536-0_2.

[44] K. Jansen and S. Kraft. "An Improved Approximation Scheme for Variable-Sized Bin Packing". In: *Theory of Computing Systems* (2015), pp. 1–61. DOI: 10.1007/s00224-015-9644-2. Pre-published.

[45] K. Jansen and S. E. J. Kraft. *A Faster FPTAS for the Unbounded Knapsack Problem*. 2014. arXiv: 1504.04650.

[46] K. Jansen and S. E. J. Kraft. "A Faster FPTAS for the Unbounded Knapsack Problem with Inversely Proportional Profits". 2015. Unpublished manuscript.

[47] K. Jansen and S. E. J. Kraft. "A Faster FPTAS for the Unbounded Knapsack Problem". In: *Proceedings of the 26th International Workshop on Combinatorial Algorithms, IWOCA 2015*. Ed. by Z. Lipták and B. Smyth. LNCS. Springer, 2015/2016. Forthcoming.

Acknowledgements

I would like to thank my supervisor Klaus Jansen for accepting me as a doctoral candidate, for his support and for his ideas.

I would also like to thank all members of our research group for the good cooperation, for interesting discussions, and the good time we had together: Ute Iaquinto, Maren Kaluza, Kim-Manuel Klein, Kati and Felix Land, Marten Maack, Parvaneh Karimi Massouleh, Marcin Pal, Lars Prädel, Malin Rau, Christina Robenek, and Ilka Schnoor as well as her husband Henning of the neighbouring research group.

I also thank all the students I supervised and taught during my PhD, for the discussions, the joy of teaching, and the things they have taught me. I would particularly thank my co-authors Jan Clemens Gehrke and Jakob Schikowski and my BSc student Dennis Papesch.

I want to thank the members of the research community I have met, for the insights into their work, and the overall good time we had together. I also thank the anonymous reviewers for the suggestions to and comments on the papers on which this dissertation is based, especially for the reference to the algorithm by Raravi and Nélis [74]. I am also grateful to the Deutsche Forschungsgemeinschaft (DFG) for the financial support of my thesis. My research was funded by the DFG project JA 612/14-1 "Entwicklung und Analyse von effizienten polynomiellen Approximationsschemata für Scheduling- und verwandte Optimierungsprobleme" and by its extension JA 612/14-2.

This list would not be complete without the friends I have made here in Kiel. I particularly thank all the current and former members of the KSG Kiel (including Martin Mayer and Leo Sunderdiek), the inhabitants of Haus Michael as well as the members of André's board game community. I also thank my former flatmate Ole Kayser and my former housemates Sahra Derichs and Johannes Effland for the company and the good time we had together. Last but not least, I particularly thank Bärbel and Reinhold Stauß for their friendship, care and support during my time in Kiel. My stay in Kiel would have been much more difficult without them, especially at the beginning of my doctorate.

Moreover, I thank all my friends in Zürich and in Lörrach.

Finally, I am profoundly grateful to my family, for their unconditional support and love: my uncles and aunts (too numerous to mention them all), my late grandfather Enrico and his late wife Sandra, my cousins Alessandra, Alice, and Lilian, Hendrik

and Gabriel, my sister Claudia and my brothers Lucas and Adrian, and especially my parents Barbara and Hans. I do not know how to express my gratitude for your relentless and ongoing support. I love you.

Contents

1 Introduction

Optimization theory has a broad range of applications: from the cost-efficient cutting of timber into pieces of desired length over the creation of rosters for airline crews up to the correct and cost-efficient routing of goods or the distribution of data in cloud computing. From a theoretical point of view, an optimization problem $\Pi = (\mathcal{I}, F, w)$ consists of three elements as stated in [48]:

- a set $\mathcal{I}$ of problem instances of the optimization problem Π,
- a set of feasible solutions $F(I)$ to every problem instance $I \in \mathcal{I}$, and
- a value $w(S) \in \mathbb{R}$ for every solution $S \in F(I)$.

There are two types of optimization problems: for a given instance I, the goal is either to maximize $w(S)$ (which are *maximization problems*), or to minimize $w(S)$ (so-called *minimization problems*).

The study of optimization problems is also driven by the fact that many of them are NP-hard or NP-complete. Since the common assumption is P $\neq$ NP, optimal algorithms with an efficient running time seem highly unlikely. The running time is called efficient if it is polynomial in the input instance length $|I|$. The natural question is: if we cannot efficiently solve a problem to optimality, how good can we approximate the optimal value in polynomial time? Are there theoretical bounds, or is it possible to find a solution as close to the optimum as we wish to? Finally, can we improve upon the running time or approximation quality of known approximation algorithms?

1.1 Approximation Algorithms

We first introduce a formal definition of approximation. Let Π be an optimization problem. The optimal value for an instance $I \in \mathcal{I}$ of Π is denoted by $\mathrm{OPT}(I)$. Let A be an algorithm for the optimization problem Π. We denote the value of its solution to I by $A(I)$. The algorithm has an absolute approximation ratio $\rho \in \mathbb{R}_{>0}$ if we have

$$\sup_{I \in \mathcal{I}} \frac{A(I)}{\mathrm{OPT}(I)} \leq \rho \quad \text{in the case of minimization problems}$$

and

$$\inf_{I \in \mathcal{I}} \frac{A(I)}{\mathrm{OPT}(I)} \geq \rho \quad \text{in the case of maximization problems.}$$

An algorithm of absolute approximation ratio ρ is also called a ρ-approximation algorithm. Note that $1 \leq \sup_{I \in \mathcal{I}} \frac{A(I)}{\mathrm{OPT}(I)}$ if Π is a minimization problem and $\inf_{I \in \mathcal{I}} \frac{A(I)}{\mathrm{OPT}(I)} \leq 1$ if it is a maximization problem.

Especially interesting for NP-hard problems are *Polynomial Time Approximation Schemes* (PTAS). A PTAS is a family of approximation algorithms $(A_\varepsilon)_{\varepsilon>0}$ where A_ε has an absolute approximation ratio of $1+\varepsilon$ for minimization and $1-\varepsilon$ for maximization problems. The running time is in $|I|^{f(\frac{1}{\varepsilon})}$, i.e. polynomial for constant ε. The function f is however not bounded in general and may e.g. be double exponential in $\frac{1}{\varepsilon}$. *Efficient Polynomial Time Approximation Schemes* (EPTAS) are PTAS where the running time is in $f(\frac{1}{\varepsilon}) \cdot |I|^{\mathcal{O}(1)}$ such that the degree of the polynomial is independent of $\frac{1}{\varepsilon}$. Finally, *Fully Polynomial Time Approximation Schemes* (FPTAS) are polynomial in $|I|$ as well as $\frac{1}{\varepsilon}$. It should be noted that not all optimization problems allow for an FPTAS or even for a PTAS.

The asymptotic approximation ratio is often considered e.g. if a PTAS is not possible. An algorithm has an asymptotic approximation ratio ρ if the following holds:

$$\limsup_{k \to \infty} \sup_{I \in \mathcal{I} \,:\, \mathrm{OPT}(I)=k} \frac{A(I)}{\mathrm{OPT}(I)} \leq \rho \quad \text{for minimization problems}$$

and

$$\liminf_{k \to \infty} \inf_{I \in \mathcal{I} \,:\, \mathrm{OPT}(I)=k} \frac{A(I)}{\mathrm{OPT}(I)} \geq \rho \quad \text{for maximization problems.}$$

Roughly speaking, the asymptotic approximation ratio can be seen as the approximation ratio achieved for large problem instances. Similarly to above, there are *Asymptotic Polynomial Time Approximation Schemes* (APTAS) $(A_\varepsilon)_{\varepsilon>0}$ that have an asymptotic approximation ratio of $1+\varepsilon$ (for minimization problems) or $1-\varepsilon$ (for maximization problems). The running time is again in $|I|^{f(\frac{1}{\varepsilon})}$. Finally, *Asymptotic Fully Polynomial Time Approximation Schemes* (AFPTAS) are APTAS with a time complexity polynomial in $|I|$ and $\frac{1}{\varepsilon}$.

1.2 Basic Concepts of Optimization

We introduce some basic concepts that are commonly used in optimization theory.

1.2.1 Linear Programming

An extremely useful tool to model and solve optimization problems is linear programming. A linear program (LP) consists of an $(m \times n)$ matrix A with entries $a_{ij} \in \mathbb{Z}$ for

all $i \in \{1,\dots,m\}$ and $j \in \{1,\dots,n\}$. Moreover, we have vectors $c \in \mathbb{Z}^n$ and $b \in \mathbb{Z}^m$. Let a_i^{T} ($i \in \{1,\dots,m\}$) be the rows of A. Furthermore, let $N_1 \dot\cup N_2 = \{1,\dots,n\}$ be a partition of the columns of A and $M_1 \dot\cup M_2 = \{1,\dots,m\}$ a partition of the rows.

Linear programs have the following form:

A minimization LP		A maximization LP	
$\min c^{\mathrm{T}}x$		$\max c^{\mathrm{T}}x$	
$a_i^{\mathrm{T}}x = b_i$	$i \in M_1$	$a_i^{\mathrm{T}}x = b_i$	$i \in M_1$
$a_i^{\mathrm{T}}x \geq b_i$	$i \in M_2$	$a_i^{\mathrm{T}}x \leq b_i$	$i \in M_2$
$x_j \geq 0$	$j \in N_1$	$x_j \geq 0$	$j \in N_1$
$x_j \in \mathbb{R}$	$j \in N_2$	$x_j \in \mathbb{R}$	$j \in N_2$

This is the general form of a (minimization or maximization) LP. In the special case of $M_1 = N_2 = \emptyset$, an LP is in standard form, and in the case of $M_2 = N_2 = \emptyset$, it is in canonical form. These three forms are equivalent, as can be easily proved. Moreover, a maximization LP can be transformed into an equivalent minimization LP, and vice versa. Note that the standard form can also be written as

$$\min c^{\mathrm{T}}x,\ Ax \geq b,\ x \geq 0 \quad \text{and accordingly} \quad \max c^{\mathrm{T}}x,\ Ax \leq b,\ x \geq 0\ .$$

$Ax \geq b$ (or $Ax \leq b$) means that every entry of $Ax = ((Ax)_i)_{i\in\{1,\dots,m\}}$ satisfies $(Ax)_i \geq b_i$ (and accordingly $(Ax)_i \leq b_i$) for all i, similarly $x \geq 0$ means $x_j \geq 0$ for all j. The canonical form becomes

$$\min c^{\mathrm{T}}x,\ Ax = b,\ x \geq 0 \quad \text{and accordingly} \quad \max c^{\mathrm{T}}x,\ Ax = b,\ x \geq 0\ .$$

Note that the conditions $a_{ij}, b_i, c_j \in \mathbb{Z}$ are often relaxed to $a_{ij}, b_i, c_j \in \mathbb{R}$.

Consider a minimization LP in the canonical form. The common assumptions are $m \leq n$, that the matrix A is full rank, i.e. of rank m, and that the set of feasible solutions $F = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ is not empty. Moreover, it is assumed that the set of solution values $\{c^T x \mid x \in F\}$ is bounded from below. It is obvious that the LP has a finite optimum under these assumptions.

Let $B = \{A_{j_1}, \dots A_{j_m}\}$ be m linearly independent columns of A. We can view B as an $(m \times m)$ matrix $B = (A_{j_1}|\cdots|A_{j_m})$, which is a regular matrix. Define

$$\bar{x} = (\bar{x}_1, \dots, \bar{x}_m) := B^{-1}b$$

and the corresponding *basic solution*

$$x = (x_1, \dots, x_n) \quad \text{with } x_j := \begin{cases} \bar{x}_k & \text{if } A_j = A_{j_k} \in B \\ 0 & \text{otherwise} \end{cases}\ .$$

A basic solution has at most m entries $x_j > 0$. Moreover, a linear program has always an optimal basic solution if the assumptions above are true. These two properties are fundamental in linear programming.

For every LP, its dual LP can be defined. The original LP is called the primal LP. We state the general definition for minimization LPs:

The primal LP		The dual LP	
$\min c^{\mathrm{T}} x$		$\max \pi^{\mathrm{T}} b$	
$a_i^{\mathrm{T}} x = b_i$	$i \in M_1$	$\pi_i \in \mathbb{R}$	$i \in M_1$
$a_i^{\mathrm{T}} x \geq b_i$	$i \in M_2$	$\pi_i \geq 0$	$i \in M_2$
$x_j \geq 0$	$j \in N_1$	$\pi^{\mathrm{T}} A_j \leq c_j$	$j \in N_1$
$x_j \in \mathbb{R}$	$j \in N_2$	$\pi^{\mathrm{T}} A_j = c_j$	$j \in N_2$

If the primal has a finite optimum of value $z \in \mathbb{R}$, then the dual has also a finite optimum of the same value z. Note that the columns A_j of the primal correspond to the inequalities $\pi^T A_j \leq c_j$ or $\pi^T A_j = c_j$ of the dual.

Until now, all variables x_j have continuous values. If we restrict all variables to integer values such that $x_j \in \mathbb{N}$ or $x_j \in \mathbb{Z}$ holds for all $j \in \{1, \ldots, n\}$, we have an integer linear program (ILP). Solving ILPs is in general NP-hard.

There are several algorithms to solve linear programs, e.g. the famous simplex algorithm as well as the ellipsoid algorithm. ILPs can be solved by methods like branch-and-bound or cutting planes. As linear programming and integer linear programming are fundamental concepts in operations research and optimization theory, a vast amount of literature exists. We recommend the books by Papadimitriou and Steiglitz [71] as well as by Jansen and Margraf [48], which also prove the statements in this subsection. The information here have been taken from the second book.

1.2.2 Column Generation

You may have remarked that a linear program can have a large number of columns n compared to the m constraints $a_i^{\mathrm{T}} x \geq b_i$ or $a_i^{\mathrm{T}} x = b_i$. Optimization algorithms that solve LPs therefore often rely on *column generation* where columns are generated during the execution of the algorithm. The actual LP that has to be solved is called the Master Problem (MP) and the LP with the currently known columns (and the corresponding variables) the Restricted Master Problem (RMP). Based on the current solution of the RMP, a pricing (sub)problem is solved to decide whether all relevant columns of the MP are considered by the RMP. If not, the pricing subproblem returns a new column that is added to the RMP, and the procedure is repeated until all relevant columns have been found. One possibility for column generation is the consideration

of the dual of the LP and to (approximately or exactly) solve a separation problem as the pricing problem: it finds violated inequalities of the dual and therefore missing columns of the primal LP.

As column generation is also an important topic in the theory of optimization, there is a large amount of literature dedicated to it. We refer to [13, 66, 70] as an introduction.

1.2.3 Max-Min Resource Sharing

This thesis will solve linear programs by transforming them into *max-min resource sharing* problems.

Let $f_i : B \to \mathbb{R}_{\geq 0}$, $i \in \{1, \ldots, N\}$, be non-negative concave functions over a non-empty, convex and compact set $B \subset \mathbb{R}^L$. The goal is to solve

$$\max \lambda \quad \text{s.t.} \quad f(v) := \begin{pmatrix} f_1(v) \\ \vdots \\ f_N(v) \end{pmatrix} \geq \lambda \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \; v \in B \;, \tag{1.1}$$

i.e. to find the largest value λ^* and the corresponding vector $v^* \in B$ such that all functions $f_i(v)$ are together as large as possible.

Such problems can be approximately solved with an algorithm by Grigoriadis et al. [33]. It relies on a solver of a subproblem, the so-called *block problem* $\Lambda(p) := \max\{p^T f(v) \mid v \in B\}$. Here, $p \in \mathbb{R}^N_{\geq 0}$ is a vector in the standard simplex with $\sum_i p_i = 1$ and $p_i \geq 0$ for $i \in \{1, \ldots, N\}$. For further information on max-min resource sharing, we refer to Section A.1 and to [33, 41].

1.3 Problem Definitions

We present the optimization problems considered in this thesis. An informal introduction to the problem is followed by the formal definition.

1.3.1 Bin Packing and Variable-sized Bin Packing

The Bin Packing Problem (BP) is one of the classical NP-complete optimization problems. In it, we are given a set of items I and a bin size. The goal is to pack the items into as few bins as possible without exceeding the size of each bin. In the Variable-sized Bin Packing Problem (VSBPP or VBP), there are not only one, but several bin sizes C at our disposal such that we may take an arbitrary number of bins of every size. The goal is to minimize the total volume of the bins used in the packing. It is clear that Variable-sized Bin Packing is a generalization of the normal Bin Packing Problem. The

study of BP and VBP has been motivated from the beginning by the efficient use of raw materials:

> Very many materials used in industry and construction come in the form of whole units (sheets of glass, tin-plate, plywood, paper, roofing and sheet iron, logs, boards, beams, reinforcing rod, forms, etc.). In using them directly or for making semi-finished products, it is necessary to divide these units into parts of the required dimensions. In doing this, scrap is usually formed and the materials actually utilized constitute only a certain per cent of the whole quantity—the rest going into scrap. ... Therefore, the minimization of scrap appears to be a very important real problem, since it would permit reduction in the norms of expenditure of critical materials. [55]

Variants of BP and VBP where the input is given in a more compact form (see below) are in fact called the Cutting Stock Problem (CSP) and the Multiple-Length Cutting Stock Problem (MLCSP).

Formal Definition An instance (I, C) of the Variable-sized Bin Packing Problem (VBP) is a pair consisting of a list $I = \{a_1, \ldots, a_n\}$ of items and a list $C = \{c_1, \ldots, c_M\}$ of different bin sizes with $n, M \in \mathbb{N}$. Every item $a \in I$ has a size $s(a) \in (0, 1]$. The bin sizes $c_l \in C$ satisfy $c_l \in (0, 1]$, and there is one unit-sized bin $c_M = 1$. A set of items $S \subset I$ can be packed in a bin of size c_l, $l \in \{1, \ldots, M\}$, as long as the total volume of the items does not exceed the capacity (i.e. size) of a bin, i.e. $\sum_{a \in S} s(a) \leq c_l$.

The Variable-sized Bin Packing Problem. Pack the items I of an instance (I, C) into bins of size in C so that the total size of the bins used is minimized. The optimal value is denoted by $\mathrm{OPT}(I, C)$.

The Bin Packing Problem (BP) is a special case of the Variable-sized Bin Packing Problem with $C = \{1\}$. Thus, a BP instance may be abbreviated as $(I, C) = I$, and the optimal value as $\mathrm{OPT}(I, C) = \mathrm{OPT}(I)$.

Closely related to BP and VBP is the Multiple-Length Cutting Stock Problem (MLCSP), where the input is provided in a more compact form: it consists of a vector $(d, M, a, \vec{n}, c, \rho)$ of d item sizes $a = (a_1, \ldots, a_d)$, the vector $\vec{n} = (n_1, \ldots, n_d)$ where n_i is the number of items of size a_i, the M stock-lengths $c = (c_1, \ldots, c_M)$ and stock-length prices $\rho = (\rho_1, \ldots, \rho_M)$. It is asked to partition the items into sets so that every set fits into a stock and the total price of stocks used is minimized. (One stock length can of course be used several times.) In our case, the price of a stock would be equal to its length. Similar to BP, the normal Cutting Stock Problem (CSP) has only one stock length such that we have without loss of generality $c = (1)$ and $\rho = (1)$.

1.3.2 Strip Packing

In this problem, a set of rectangles has to be packed into a strip of infinite height such that the rectangles do not overlap. In the variant we consider, the rectangles must not be rotated. As a possible application, Prädel [73] suggests cutting out rectangles from sheets of raw material like wood. Rotations are not allowed e.g. because of the grain of the wood. Like Bin Packing, Strip Packing (SP) is NP-complete.

Formal Definition Let I be a set of n rectangles $I = \{a_1, \ldots, a_n\}$. Each rectangle has a width $w(a_j) \in (0, 1]$ and a height $h(a_j) \in (0, 1]$ for $j \in \{1, \ldots, n\}$. Pack the rectangles into a strip of unit width such that the total height of the packing is minimized. The rectangles must not overlap and must not be rotated.

1.3.3 Knapsack Problems

In the normal Knapsack Problem (KP), a set of items is given, each with a size and a profit, together with a knapsack size. The goal is to choose a subset of items such that they fit into the knapsack and at the same time maximize the total profit. In the 0-1 variant of KP (called 0-1 KP), an item may be taken only once. In the Bounded Knapsack Problem (BKP), an individual number of copies of every item is allowed, and the Unbounded Knapsack Problem (UKP) admits an unlimited number of item copies. A famous "application" is a burglar that breaks into a museum. Since his knapsack has only a limited capacity, his goal is to choose some of the exhibits such that the loot still fits into the knapsack and that has at the same time a value as large as possible.

The Knapsack Problem with Inversely Proportional Profits (KPIP) is a generalization of KP with several knapsack sizes of which only one can be used. It may therefore seem natural to choose the largest knapsack size. If an item is packed into a knapsack of size c_l, the profit of an item is however scaled by $\frac{1}{c_l}$. A knapsack of smaller size may therefore allow for a larger profit. The 0-1, bounded and unbounded variant of KPIP are defined similar to the normal Knapsack Problem. A motivation for KPIP was suggested by Felix Land: the scaling of the item profits takes into account that it is harder to fill a smaller knapsack as good as possible.

Formal Definition An instance I of the Knapsack Problem (KP) consists of a list of items $a_1, \ldots, a_n$, $n \in \mathbb{N}$. Every item has a profit $p(a_j) = p_j$ and a size $s(a_j) = s_j$. Moreover, a knapsack size c is given. In the literature, the profits p_j and sizes s_j as well as c are normally natural numbers such that $p_j, s_j, c \in \mathbb{N}$. For column generation, we

will study the Knapsack Problem with $p_j \in (0,1]$, $s_j \in (0,1]$, and $c = 1$. In both cases, the problem definition is the same:

The 0-1 Knapsack Problem. Choose a subset $V \subset \{a_1, \ldots, a_n\}$ such that the total profit of V is maximized and the total size of the items in V is at most c.

Mathematically, the problem can be defined by

$$\max \left\{ \sum_{j=1}^{n} p_j x_j \;\middle|\; \sum_{j=1}^{n} s_j x_j \leq c \;;\; x_j \in \{0,1\}\, \forall j \right\} \quad .$$

In the bounded variant (BKP), up to $d_j \in \mathbb{N}$ copies of each item a_j may be taken (i.e. $x_j \in \{0, \ldots, d_j\}$), and in the unbounded variant (UKP), an arbitrary number of copies of every item is allowed (i.e. $x_j \in \mathbb{N}$).

The 0-1 Knapsack Problem with Inversely Proportional Profits (0-1 KPIP) is a generalization of 0-1 KP where the items have sizes $s_j \in (0,1]$ and basic profits $p_j \in (0,1]$. Moreover, M knapsack sizes $0 < c_1 < \ldots < c_M = 1$ are given. If an item a_j is packed into the knapsack of size c_l for $l \in \{1, \ldots, M\}$, its profit counts as $\frac{p_j}{c_l}$.

The 0-1 Knapsack Problem with Inversely Proportional Profits. Find the knapsack size c_l and the corresponding item set V such that the total profit is maximized.

Mathematically, the problem is defined by

$$\max_{l \in \{1,\ldots,M\}} \max \left\{ \sum_{j=1}^{n} \frac{p_j}{c_l} x_j \;\middle|\; \sum_{j=1}^{n} s_j x_j \leq c_l \;;\; x_j \in \{0,1\}\, \forall j \right\} \quad .$$

The Bounded Knapsack Problem with Inversely Proportional Profits (BKPIP) and the Unbounded Knapsack Problem with Inversely Proportional Profits (UKPIP) are defined similar to BKP and UKP.

Note that the goal of the Knapsack Problem is to maximize the profit by choosing the right subset of items, whereas the goal of (Variable-sized) Bin Packing is to minimize the number of bins or the volume of the bins used when all items are packed.

1.3.4 Scheduling on Unrelated Machines of Few Different Types

Scheduling is a classical optimization problem. Jobs—e.g. computing tasks—have to be distributed to machines such that one objective is minimized, normally the maximum completion time of the jobs. One example is a cluster of processors that has to perform a large amount of computation tasks. In general, the machines may be heterogeneous: a processor may have been designed to perform a certain type of calculations very fast, but may not be suited for other ones. However, the number of different machine types may indeed be limited, as can be the case for e.g. a cluster of CPUs and GPUs.

Formal Definition An instance I consists of a set $\mathcal{J} = \mathcal{J}(I)$ of n jobs and a set $\mathcal{M} = \mathcal{M}(I)$ of m machines. Every job j has a processing time on machine i of $p_{ij} \geq 0$ for $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. A non-preemptive schedule is a distribution of the jobs to the machines such that every job is processed by exactly one machine. Formally, it is a mapping $\sigma : \mathcal{J} \to \mathcal{M}$ of each job j to a machine i. The objective is to find a schedule σ that minimizes the makespan $\max_{i \in \mathcal{M}} \sum_{j:\sigma(j)=i} p_{ij}$, i.e. the maximum completion time of all jobs. Thus, even the longest-running machine shall finish the processing as soon as possible. This classical problem is called Scheduling on Unrelated Machines and is denoted by $\mathrm{R}\,||\,C_{\max}$ in the 3-field notation [32].

As suggested above, we look at a variant where the machines are only of K different types, where K is seen as constant: for two different machines i and i' (with $i \neq i'$) of the same type, we have $p_{ij} = p_{i'j}$ for all jobs $j \in \{1, \ldots, n\}$. The machines of type k are denoted by $\mathcal{M}_k$ such that the sets $\mathcal{M}_1, \ldots, \mathcal{M}_K$ are a disjoint partition of $\mathcal{M}$. The number of machines of one type is $m_k := |\mathcal{M}_k|$ for $k \in \{1, \ldots, K\}$. Hence, $m_1 + \cdots + m_K = m$ holds. The problem is denoted by $(\mathrm{P}m_1, \ldots, \mathrm{P}m_K)\,||\,C_{\max}$ and called Scheduling on Unrelated Machines of Few Different Types .

$(\mathrm{P}m_1, \ldots, \mathrm{P}m_K)\,||\,C_{\max}$. Find a schedule $\sigma : \mathcal{J} \to \mathcal{M}$ that minimizes the makespan.

1.4 General Assumptions

We assume that basic arithmetic operations as well as the computation of the logarithm can be done in $\mathcal{O}(1)$. Moreover, we assume that all arithmetic operations can be done exactly, e.g. because the non-integral values are in $\mathbb{Q}$. It is clear that it may not be possible to express the logarithm of a number $a > 0$ exactly, but only approximately. However, we normally use the logarithm for mathematical expressions whose final values are rounded. We may e.g. only need $\lfloor \log_2 a \rfloor$ and not $\log_2 a$. Hence, we assume that we can determine the logarithm with a tolerance tight enough to obtain the same rounded value as if we were able to exactly compute the logarithm and exactly perform arithmetic operations with it.

1.5 Outline of the Thesis

The results of this thesis are presented in five chapters. Note that all results were obtained in collaboration with my supervisor Prof. Dr. Klaus Jansen.

Chapter 2 first presents a short introduction to the known results for the Bin Packing Problem (BP) and the Variable-sized Bin Packing Problem (VBP). It is fol-

lowed by the presentation of improved AFPTAS for both problems with $A_\varepsilon(I, C) \leq (1+\varepsilon)\mathrm{OPT}(I, C) + \mathcal{O}(\log^2 \frac{1}{\varepsilon})$. The results were first presented at the conference MFCS 2012 [42]. They have been accepted to be published in a special journal issue of the conference CSR 2013 [44].

Chapter 3 first states the known results for the Knapsack Problem (KP) and then presents FPTAS for all three variants of the Knapsack Problem with Inversely Proportional Profits (KPIP). They are faster than the natural approach to separately solve the Knapsack Problem for each knapsack size c_l. The Unbounded Knapsack Problem with Inversely Proportional Profits (UKPIP) is in fact the column generation subproblem for our algorithm of Chapter 2: hence, its running time is directly improved by the faster FPTAS for UKPIP. The Knapsack Problem with Inversely Proportional Profits was formally introduced at the conference CSR 2013 [43] where the algorithms were also presented for the first time. These results will be published together with the results in Chapter 2 in the special journal issue of the conference [44].

Chapter 4 presents a faster FPTAS for the Unbounded Knapsack Problem (UKP) that has also an improved space complexity. Note that the column generation subroutine of the AFPTAS for the Bin Packing Problem only has to solve UKP instances (and not UKPIP instances). The faster FPTAS for UKP therefore improves the time complexity of our algorithm for BP. The result was presented at IWOCA 2015 [47] and is also available on arXiv [45]. The chapter moreover shows at the end how the new algorithm for UKP decreases the running time of an AFPTAS for Strip Packing (SP).

Chapter 5 combines the results in Chapter 3 and 4. We get an FPTAS for the Unbounded Knapsack Problem with Inversely Proportional Profits that is faster and that has also a better space complexity. Thus, the running time of the AFPTAS for VBP is further improved. The paper on which this chapter is based has not been published [46].

Chapter 6 concludes the thesis with an improved PTAS for Scheduling on Unrelated Machines of Few Different Types ($(\mathrm{P}m_1, \ldots, \mathrm{P}m_K) \,||\, C_{\max}$). The result was obtained in collaboration with the students Jan Clemens Gehrke and Jakob Schikowski as well as my supervisor Klaus Jansen. It was accepted at SOFSEM 2016 [27] and is also available as a technical report [26].

2 Bin Packing and Variable-sized Bin Packing

2.1 Introduction

2.1.1 Known Results

As already mentioned, the Bin Packing Problem (BP) is a classic NP-complete problem [25]. The first to consider it in the form of the (Multiple-Length) Cutting Stock Problem (but naming it differently) were Kantorovich [55] and Eisemann [17]. Several approximation algorithms with a polynomial running time are known for Bin Packing (e.g. First-Fit (FF), Next-Fit (NF), Best-Fit, First-Fit Decreasing (FFD) or Next-Fit Decreasing (NFD)). As BP is a minimization problem, we always have $1 \leq \sup_I \frac{A(I)}{\mathrm{OPT}(I)}$ for the absolute approximation ratio of any algorithm A. However, no polynomial-time algorithm can achieve an absolute approximation ratio $\sup_I \frac{A(I)}{\mathrm{OPT}(I)} < \frac{3}{2}$ unless P = NP [25]. (In fact, First-Fit Decreasing attains this absolute ratio [82].)

Note that the bound $\frac{3}{2}$ for the absolute approximation ratio is due to the fact that a polynomial algorithm could otherwise distinguish between the optimum of 2 or 3 for BP instances and therefore solve the NP-complete Partition Problem in polynomial time [25]. Since only such small instances prevent an absolute ratio better than $\frac{3}{2}$, larger instances may allow for a better approximation ratio. It is therefore a good idea to consider the asymptotic approximation ratio, which can be seen as the approximation ratio for large instances (see Section 1.1). And in fact, every packing $FFD(I)$ found by FFD satisfies $FFD(I) \leq \frac{11}{9}\mathrm{OPT}(I) + \frac{6}{9}$ [15, 16] so that FFD has an asymptotic approximation ratio of $\frac{11}{9}$, which is obviously smaller than $\frac{3}{2}$.

In 1981, Fernandez de la Vega and Lueker [20] presented the first APTAS for BP: the running time is polynomial in the input size $|I| \geq n$, but exponential in $\frac{1}{\varepsilon}$. One year later, Karmarkar and Karp [56] found the first AFPTAS satisfying $A_\varepsilon(I) \leq (1+\varepsilon)\mathrm{OPT}(I) + \mathcal{O}(\frac{1}{\varepsilon^2})$. In 1991, Plotkin et al. [72] presented an improved algorithm satisfying $A_\varepsilon(I) \leq (1+\varepsilon)\mathrm{OPT}(I) + \mathcal{O}(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))$ and with a better running time in $\mathcal{O}(\frac{1}{\varepsilon^6}\log^6(\frac{1}{\varepsilon}) + \log(\frac{1}{\varepsilon})n)$. Shachnai and Yehezkely [78] reduced the running time further to $\mathcal{O}(\frac{1}{\varepsilon^4}\log^3(\frac{1}{\varepsilon}) \cdot \min\{\frac{1}{\varepsilon^2}, \frac{1}{\varepsilon^{0.62}}\log^{0.62}(\frac{1}{\varepsilon})N\} + \log(\frac{1}{\varepsilon})n)$, where N is the longest binary representation of any input element. For general BP instances, the algorithm therefore needs time in $\mathcal{O}(\frac{1}{\varepsilon^6}\log^3(\frac{1}{\varepsilon}) + \log(\frac{1}{\varepsilon})n)$. The algorithm is an appli-

cation of Shachnai's and Yehezkely's AFPTAS for Bin Packing with Size Preserving Fragmentation (BP-SPF).

Karmarkar and Karp [56] also presented a polynomial algorithm with the approximation guarantee $A(I) \leq \mathrm{OPT}(I) + \mathcal{O}(\log^2 \mathrm{OPT}(I))$, i.e. with an additive approximation factor. This result was only recently improved after 20 years: Rothvoß [75] first presented an algorithm with $A(I) \leq \mathrm{OPT}(I) + \mathcal{O}(\log(\mathrm{OPT}(I)) \log\log(\mathrm{OPT}(I)))$ and then Hoberg and Rothvoß [35] an algorithm with $A(I) \leq \mathrm{OPT}(I) + \mathcal{O}(\log \mathrm{OPT}(I))$. In both cases, the algorithms are randomized and have an expected polynomial running time in the input length. It should be noted that Cutting Stock with d different item sizes has an exact polynomial-time algorithm if d is considered to be constant [31].

The Variable-sized Bin Packing Problem (VBP) was studied by Friesen and Langston [22] who analysed three algorithms with the asymptotic approximation ratios 2, $\frac{3}{2}$ and $\frac{4}{3}$. Murgolo [69] presented an AFPTAS with $A_\varepsilon(I) \leq (1+\varepsilon)\mathrm{OPT}(I,C) + \mathcal{O}(\frac{1}{\varepsilon^4})$. This was improved to $(1+\varepsilon)\mathrm{OPT}(I,C) + \mathcal{O}(\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}))$ by Shachnai and Yehezkely [78], again by an application of their BP-SPF algorithm. The improved running time of the algorithm is $\mathcal{O}(\frac{1}{\varepsilon^8}\log^3(\frac{1}{\varepsilon}) \cdot \min\{\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}), \frac{1}{\varepsilon^{0.24}}N\} + (M+n)\log(\frac{1}{\varepsilon}))$, where N is (again) the longest binary representation of any input element. For general instances, the running time is therefore bounded by $\mathcal{O}(\frac{1}{\varepsilon^{10}}\log^4(\frac{1}{\varepsilon}) + \log(\frac{1}{\varepsilon})(M+n))$.

2.1.2 Our Result

This chapter presents an AFPTAS for BP and VBP with the smaller additive term $\mathcal{O}(\log^2(\frac{1}{\varepsilon}))$ and a further improved running time.

Theorem 2.1. *There is an AFPTAS $(A_\varepsilon)_{\varepsilon>0}$ for Variable-sized Bin Packing that finds for $\varepsilon \in (0, \frac{1}{2}]$ a packing of an instance (I,C) in $A_\varepsilon(I) \leq (1+\varepsilon)\mathrm{OPT}(I,C) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$ bins. Its running time is in*

$$\mathcal{O}\left(\frac{1}{\varepsilon^5}\log^5\frac{1}{\varepsilon} + M + \log\left(\frac{1}{\varepsilon}\right)n\right) \quad .$$

Theorem 2.2. *There is an AFPTAS $(A_\varepsilon)_{\varepsilon>0}$ for Bin Packing that finds for $\varepsilon \in (0, \frac{1}{2}]$ a packing of I in $A_\varepsilon(I) \leq (1+\varepsilon)\mathrm{OPT}(I) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$ bins. Its running time is in*

$$\mathcal{O}\left(\frac{1}{\varepsilon^5}\log^4\frac{1}{\varepsilon} + \log\left(\frac{1}{\varepsilon}\right)n\right) \quad .$$

For column generation, the VBP algorithm relies on FPTAS for the Unbounded Knapsack Problem with Inversely Proportional Profits (UKPIP) whereas the BP algorithm uses algorithms for the Unbounded Knapsack Problem (UKP). Hence, the theorems above state the running times of the AFPTAS for the case where the best FPTAS for UKP and UKPIP of this thesis are used.

Therefore, we present the actual time complexity of the AFPTAS if the running time of the algorithms for UKP and UKPIP is left open (with the exception of a lower bound). To do so, we introduce (in a slight abuse of notation) the functions $UKPIP$ and UKP.

Definition 2.3. *Take an instance of the Unbounded Knapsack Problem with Inversely Proportional Profits (UKPIP) with d_1 items and M_1 knapsacks, and where $c_{\min}$ is the smallest knapsack size. Then, $UKPIP(d_1, M_1, c_{\min}, \frac{\bar{\varepsilon}}{6})$ denotes the running time of an FPTAS for UKPIP that solves such an instance with the approximation ratio $(1 - \frac{\bar{\varepsilon}}{6}) \in \Theta(1 - \varepsilon)$. Similarly, let $UKP(d_1, \frac{\bar{\varepsilon}}{6})$ be the running time of an FPTAS for UKP with the same approximation ratio and d_1 items.*

Theorem 2.4. *Let (I, C) be a VBP instance with n items, M bin sizes and optimal value $\mathrm{OPT}(I, C)$. There is an AFPTAS $(A_\varepsilon)_{\varepsilon>0}$ such that A_ε finds for $\varepsilon \in (0, \frac{1}{2}]$ a solution of the instance with an objective value of $A_\varepsilon(I, C) \leq (1 + \varepsilon)\,\mathrm{OPT}(I, C) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$. The running time of A_ε is bounded by*

$$\mathcal{O}\left(UKPIP\left(d_1, M_1, c_{\min}, \frac{\bar{\varepsilon}}{6}\right) \cdot \frac{1}{\varepsilon^3} \log \frac{1}{\varepsilon} + M + \log\left(\frac{1}{\varepsilon}\right) n\right)$$

if we assume that $UKPIP(d_1, M_1, c_{\min}, \frac{\bar{\varepsilon}}{6}) \in \Omega(\frac{1}{\varepsilon^2})$ for $d_1 \in \mathcal{O}(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$, $M_1 \in \mathcal{O}(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ and $c_{\min} \geq \varepsilon$.

Theorem 2.5. *Let I be a BP instance with the optimal value $\mathrm{OPT}(I)$. There is an AFPTAS $(A_\varepsilon)_{\varepsilon>0}$ such that A_ε finds for $\varepsilon \in (0, \frac{1}{2}]$ a packing of I in $A_\varepsilon(I) \leq (1 + \varepsilon)\mathrm{OPT}(I) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$ bins. The running time is bounded by*

$$\mathcal{O}\left(UKP\left(d_1, \frac{\bar{\varepsilon}}{6}\right) \cdot \frac{1}{\varepsilon^3} \log \frac{1}{\varepsilon} + \log\left(\frac{1}{\varepsilon}\right) n\right) \quad .$$

if we assume that $UKP(d_1, \frac{\bar{\varepsilon}}{6}) \in \Omega(\frac{1}{\varepsilon^2})$ for $d_1 \in \mathcal{O}(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$.

2.2 Overview

First, Section 2.3 explains the integer linear program (ILP) for VBP, which is a well-known approach (see [17, 55, 69]). The basic idea is the following: let (I, C) be a VBP instance with d different item sizes $b_1 > \ldots > b_d$ and where every item has a size $s(a) \geq \delta$ for a constant $\delta > 0$. The corresponding ILP is

$$\min c^{\mathrm{T}} v \text{ with } Av \geq b,\ v \in \mathbb{Z}^q \text{ and } v \geq 0 \ . \tag{2.1}$$

The basis of the ILP are configurations that correspond to the columns of A: one configuration $K^{(l)}$ is a subset of items that fits into one bin c_l. AFPTAS usually consider

the relaxed version of (2.1) with $v \in \mathbb{R}^q, v \geq 0$. The relaxed optimal value is denoted by $\mathrm{LIN}(I, C)$. The main difficulty is to solve the relaxed ILP efficiently and to round the solution to an integer solution close to the optimum. Our algorithm also uses this principle.

Section 2.4 introduces a partial ordering on VBP instances, which is useful for the analysis of the algorithm. Moreover, the total size $\mathrm{Area}(I)$ of the items in instance I is formally defined.

Then, Section 2.5 presents our basic algorithm for rounding in the Subsections 2.5.1 and 2.5.2: it is explained for a VBP instance $(I^{(1)}, C^{(1)})$ with d_1 different item sizes and M_1 bin sizes. (The instance is denoted differently for convenience as will be seen in a moment.) First, it is shown how to solve the relaxed ILP for $(I^{(1)}, C^{(1)})$ approximately by applying to VBP the algorithm by Grigoriadis et al. [33], which yields the first solution $v^{(1)}$. Then, Shmonin's rounding technique [80] is adapted: the entries of $v^{(1)}$ are rounded down to the next integer, which already packs a subset $I^{(1)}_{\mathrm{int}} \subset I^{(1)}$. The remaining items $I^{(1)}_{\mathrm{res}} = I^{(1)} \setminus I^{(1)}_{\mathrm{int}}$ are split into J_1 and J'_1. The set J'_1 is packed using Next-Fit, while J_1 is rounded, which yields the new instance $(I^{(2)}, C^{(1)})$ that is processed in the same way. The procedure is then iterated such that we get instances $(I^{(k)}, C^{(1)})$ (of decreasing size and with d_k different item sizes) until all items of $(I^{(1)}, C^{(1)})$ have been packed. In contrast to Shmonin's proof, the relaxed ILPs (2.1) are only solved approximately and not optimally. The analysis of the basic algorithm in Subsection 2.5.3 therefore proves that the number of item sizes and the total size $\mathrm{Area}(I^{(k)})$ of the items halves in every iteration k, which makes it possible to bound the final objective value by $(1 + 4\varepsilon)\mathrm{OPT}(I^{(1)}, C^{(1)}) + \mathcal{O}(\log(\frac{1}{\delta}) \log(d_1))$.

When the basic algorithm solves the relaxed ILP, the needed columns of the matrix A are generated dynamically by solving instances of UKPIP. As mentioned in Section 1.2.2, it is necessary because there may be an exponential number of columns, i.e. $q \in 2^{\mathcal{O}(\frac{1}{\varepsilon} \log^2(\frac{1}{\varepsilon}))}$. Column generation for BP and VBP is for instance also used in [29, 30, 56, 69, 72, 78].

In general, the additive factor $\mathcal{O}(\log(\frac{1}{\delta}) \log(d_1))$ of our basic algorithm is not equal to $\mathcal{O}(\log^2(\frac{1}{\varepsilon}))$. Section 2.6 shows how to preprocess a general VBP instance (I, C) such that this is the case. First, we divide the items I into large and small items I_{large} and I_{small}, where the items in the first set have sizes $s(a) \geq \delta = \varepsilon^2$. A special subset $I_{\mathrm{huge}} \subset I_{\mathrm{large}}$ is determined that (roughly speaking) contains the largest items of I_{large}. The remaining items in $I_{\mathrm{large}} \setminus I_{\mathrm{huge}}$ are rounded up to get $I^{(1)}$. As the set I_{huge} contains the largest items of I, the item set $I^{(1)}$ still satisfies $\mathrm{OPT}(I^{(1)}, C) \leq \mathrm{OPT}(I, C)$. The rounding is done to have only $d_1 \in \mathcal{O}(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ different item sizes, which also decreases the overall running time. To further improve the running time, only a subset $C^{(1)} \subset C$ of bin sizes is used, which does not increase the approximation ratio

too much. The instance $(I^{(1)}, C^{(1)})$ is then packed with the basic algorithm. Now, $\mathcal{O}(\log(\frac{1}{\delta})\log(d_1)) = \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$ holds. The remaining items in I_{huge} and I_{small} are greedily packed, which (again) only slightly increases the approximation ratio.

The scheme of our algorithm is similar to the one of other AFPTAS for BP [20, 56, 72] and VBP [69, 78]. Our main contribution is the combination and extension of the algorithm by Grigoriadis et al. [33] and the theoretical result by Shmonin [80, Chapter 6] to solve the relaxed ILP and to round the fractional to an integer solution close to the optimum. Note that Shmonin's method is a modification of the Bin Packing algorithm by Karmarkar and Karp [56].

2.3 Integer Linear Programs and Linear Programs for VBP

In this subsection, the ILP for VBP is introduced. Let (I, C) be a VBP instance. Let d be the number of different item sizes, and let $b_1 > \ldots > b_d$ be the subsequence consisting of the different item sizes in $s_1 \geq \ldots \geq s_n$. Moreover, every item has a size of at least $s(a) \geq \delta$ for a constant $\delta > 0$. We introduce configurations: a configuration $K^{(l)}$ for the bin size c_l is a subset $J \subseteq I$ such that the items in J fit into a bin of size c_l, i.e. $\sum_{a \in J} s(a) \leq c_l$. Let $K_1^{(l)}, \ldots, K_{q(I,l)}^{(l)}$ be all configurations of a bin size c_l. The number $q(I, l)$ of the configurations may be exponential in the number of item sizes d.

A configuration $K_j^{(l)}$ can be described by a multiset

$$\left\{a(K_j^{(l)}, b_1) : b_1, \ldots, a(K_j^{(l)}, b_d) : b_d\right\}$$

where $a(K_j^{(l)}, b_i)$ denotes the number of items of size b_i in configuration $K_j^{(l)}$ (see Figure 2.1). Furthermore, let n_i be the total number of items of size b_i in I. (Obviously, $n_1 + \cdots + n_d = n$ holds.) It is possible to describe the VBP instance as an integer linear program (ILP) [17, 29, 30]:

$$\begin{aligned} &\min \sum_{l=1}^{M} \sum_{j=1}^{q(I,l)} c_l \cdot v_j^{(l)} \\ &\sum_{l=1}^{M} \sum_{j=1}^{q(I,l)} a(K_j^{(l)}, b_i) \cdot v_j^{(l)} = n_i \quad \text{for } i \in \{1, \ldots, d\} \\ &v_j^{(l)} \in \mathbb{N} \cup \{0\} \qquad \text{for } l \in \{1, \ldots, M\} \text{ and } j \in \{1, \ldots, q(I,l)\} \end{aligned} \tag{ILP-VBP}$$

The optimal value of this ILP is equal to $\text{OPT}(I, C)$ because $\sum_l \sum_j c_l v_j^{(l)}$ sums the volume up of the bins used. The constraints make sure that all n_i items of size d_i are packed: $a(K_j^{(l)}, b_i) \cdot v_j^{(l)}$ is the number of items of size b_i packed in the solution by

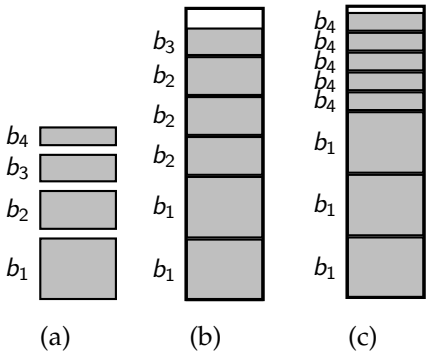


Figure 2.1: (a) shows the item sizes $b_1, \ldots, b_4$. Bin (b) is packed according to configuration $\{2 : b_1, 3 : b_2, 1 : b_3, 0 : b_4\}$ and bin (c) according to configuration $\{3 : b_1, 0 : b_2, 0 : b_3, 5 : b_4\}$.

configuration $K_j^{(l)}$. Thus, summing over all configurations and bin sizes yields the total number of packed items of size b_i. We now consider the LP relaxation with the optimum $\mathrm{LIN}(I, C)$ where the conditions $v_j^{(l)} \in \mathbb{N} \cup \{0\}$ are replaced by $v_j^{(l)} \geq 0$ and the conditions $\ldots = n_i$ by $\ldots \geq n_i$. In the ILP as well as the relaxed LP, a variable $v_j^{(l)}$ can be interpreted as a vertical slice of a bin, packed according to configuration $K_j^{(l)}$. The slice has the width $v_j^{(l)}$ (see Figure 2.2).

2.4 Useful Definitions

We introduce a partial order on VBP instances that will be useful later. It is a natural extension of the order by Fernandez de la Vega and Lueker [20].

Definition 2.6. *Let (J_1, C) and (J_2, C) be two VBP instances with the same set of bin sizes C. We write $(J_2, C) \leq (J_1, C)$ if there is an injective function $f : J_2 \to J_1$ such that $s(a) \leq s(f(a))$ holds for every $a \in J_2$. We write $J_2 \leq J_1$ for item sets if the same condition holds.*

The following lemmas will be used later.

Lemma 2.7. *Let $\mathrm{Area}(I) := \sum_{a \in I} s(a)$ be the total size of the items in I. Then we have $\mathrm{Area}(I) \leq \mathrm{LIN}(I, C) \leq \mathrm{OPT}(I, C)$.*

Proof. Let us consider the first inequality. All items are packed (fractionally) into bins. Since the volume of the items is at most the volume of the bins they are packed into, we have $\mathrm{Area}(I) \leq \mathrm{LIN}(I, C)$. The second inequality is obvious: integer solutions to ILPs are at the same time solutions to the relaxed LPs. □

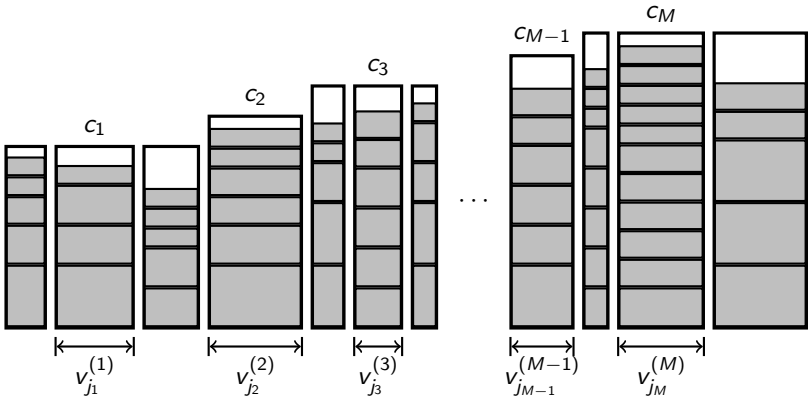


Figure 2.2: A fractional packing of a VBP instance. The variable $v_j^{(l)}$ can be interpreted as a slice of a bin c_l packed according to configuration $K_j^{(l)}$ with a width of $v_j^{(l)}$.

Lemma 2.8. *Let J_2 and J_1 be two item sets with $J_2 \leq J_1$, and let $(J_2, C) \leq (J_1, C)$ be the corresponding VBP instances that share the same set of bin sizes C. Then we have* $\text{Area}(J_2) \leq \text{Area}(J_1)$, $\text{LIN}(J_2, C) \leq \text{LIN}(J_1, C)$ *and* $\text{OPT}(J_2, C) \leq \text{OPT}(J_1, C)$.

Proof. $\text{Area}(J_2) \leq \text{Area}(J_1)$ is obvious. A (fractional or integral) packing of the items in J_1 can be transformed into a packing of the items in J_2 by replacing every item $b \in J_1$ by the corresponding item $a \in J_2$ with $f(a) = b$; we remove the items $b \in J_1$ for which there are not any $a \in J_2$ with $f(a) = b$. As some bins may now be empty, this packing of J_2 has at most the objective value of the packing for J_1. The optimal packing of J_2 may have an even smaller objective value, therefore $\text{LIN}(J_2, C) \leq \text{LIN}(J_1, C)$ or $\text{OPT}(J_2, C) \leq \text{OPT}(J_1, C)$ holds. Note that we have implicitly used the fact that (J_1, C) and (J_2, C) share the same set of bin sizes: if we had an instance $(J_1, \tilde{C})$ with $C \subsetneq \tilde{C}$, the constructed packing could use bin sizes not available for packing (J_2, C), and the packing could be infeasible. (This proof was adapted from [20].) □

2.5 The Basic Algorithm

In this subsection, we consider a VBP instance $(I^{(1)}, C^{(1)})$ with d_1 item sizes, M_1 bin sizes and (still) with $s(a) \geq \delta$ for all items $a \in I^{(1)}$. The basic scheme $(Alg_\varepsilon)_{\varepsilon>0}$ is presented in Algorithm 2.1: as mentioned above, it is an adaptation and a practical application of the methods presented by Karmarkar and Karp [56] and Shmonin [80, Chapter 6], combined with a method based on the max-min resource sharing algorithm by Grigoriadis et al. [33].

Algorithm 2.1: The basic VBP algorithm Alg_ε

Input: $\varepsilon > 0$, instance $(I^{(1)}, C^{(1)})$

1 Set $k := 1$;

2 **while** *true* **do**

2.1 Solve the relaxed linear program corresponding to $(I^{(k)}, C^{(1)})$ approximately with the accuracy $(1 + \varepsilon)$ (see Subsection 2.5.1) ;

2.2 Take the integral part $\lfloor v^{(k)} \rfloor = (\lfloor v_j^{(l,k)} \rfloor)$ of the approximate solution $v^{(k)} = (v_j^{(l,k)})$. Pack the items according to $\lfloor v^{(k)} \rfloor$ in the respective bins: these items are the instance $(I_{\text{int}}^{(k)}, C^{(1)})$. The remaining, non-packed items are the residual instance $(I_{\text{res}}^{(k)}, C^{(1)})$;

2.3 **if** $I_{\text{res}}^{(k)} = \emptyset$ **then**
    break;
  else
    Transform $(I_{\text{res}}^{(k)}, C^{(1)})$ into two instances $(J_k', C^{(1)})$ and $(J_k, C^{(1)})$, where the items in J_k have been rounded up (see Subsection 2.5.2);

2.4 Pack J_k' into unit-sized bins with Next-Fit; open a new bin if necessary;

2.5 **if** $J_k = \emptyset$ **then**
    break;
  else
    Set $(I^{(k+1)}, C^{(1)}) := (J_k, C^{(1)})$;
    $k := k + 1$;

3 Replace the rounded-up sizes of the items by their original sizes in $I^{(1)}$;

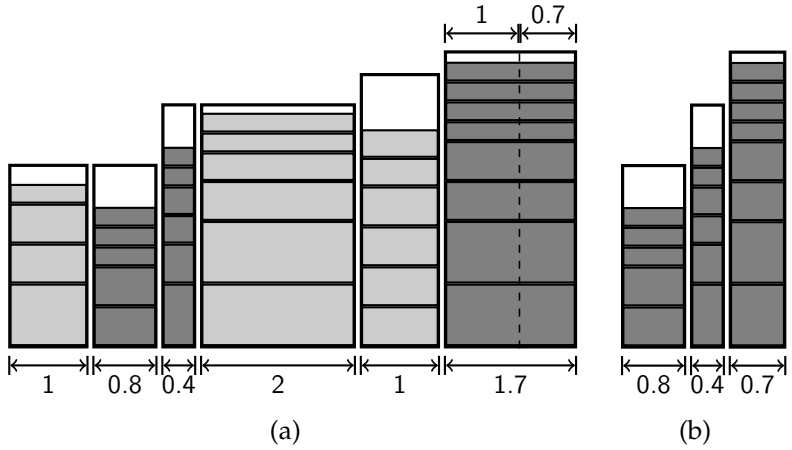


Figure 2.3: (a) shows the packing v of the VBP instance (I, C). The values of the $v_j^{(l)}$ are written under the bins. Bins corresponding to integral variables $v_j^{(l)}$ have light-gray items, fractional bins have dark-grey items. The fractional part $v - \lfloor v \rfloor$ corresponds to a packing, which is shown in (b). These fractionally packed items are the item set I_{res}.

Remark 2.9. The algorithm Alg_ε finishes if either all items in an instance $I^{(k)}$ are packed by $\lfloor v^{(k)} \rfloor$, i.e. $I_{\text{int}}^{(k)} = I^{(k)}$, or all items that have not been packed by $\lfloor v^{(k)} \rfloor$ are contained in J_k' and packed by Next-Fit. If neither of these conditions is met, the remaining unpacked items J_k become the new instance $(I^{(k+1)}, C^{(1)})$, which is processed again in the same way. Since items are packed integrally either by $\lfloor v_j^{(l,k)} \rfloor$ or by Next-Fit, we obtain an integral and feasible packing. (Figure 2.3 illustrates how $I_{\text{res}}^{(k)}$ is obtained from $I^{(k)}$.) Note that $I_{\text{res}}^{(k)}$ always contains only complete (and not fractional) items although $v - \lfloor v^{(k)} \rfloor$ packs them fractionally.

2.5.1 Solving the LPs Approximately

In Step 2.1, the algorithm has to approximately solve the relaxation of the integer programs (ILP-VBP) for the instances $(I^{(k)}, C^{(1)})$, where we have d_k different item sizes and M_1 bin sizes. There is a well-known method for LPs of packing problems (see e.g. [14, 41] for Strip Packing) based on the max-min resource sharing algorithm by Grigoriadis et al. [33]. The method can be adapted to VBP. We introduce it for general VBP instances (I, C).

Let r be a guessed value for $\mathrm{LIN}(I,C)$. By introducing an additional variable λ and dividing every constraint $\sum_l \sum_j a(K_j^{(l)}, b_i)\ v_j^{(l)} \geq n_i$ by n_i, we can rewrite the relaxed (ILP-VBP) as

$$\begin{aligned} &\max \lambda \\ &\sum_{l=1}^{M} \sum_{j=1}^{q(I,l)} \frac{a(K_j^{(l)}, b_i)}{n_i} v_j^{(l)} \geq \lambda \quad \text{for } i \in \{1,\ldots,d\} \\ v = \left(v_j^{(l)}\right) \in B := &\left\{ v \;\middle|\; \sum_{l=1}^{M} c_l \sum_{j=1}^{q(I,l)} v_j^{(l)} = r;\ v_j^{(l)} \geq 0 \right\} \ . \end{aligned} \tag{2.2}$$

Remark 2.10. $r \geq \mathrm{LIN}(I,C)$ holds for (2.2) if and only if $\lambda \geq 1$. Moreover, the conditions $\sum_l \sum_j a(K_j^{(l)}, b_i)\ v_j^{(l)} \geq n_i$ hold if and only if $\lambda \geq 1$.

Let $\tilde{A}$ be the matrix

$$\tilde{A} := \left(\frac{a(K_j^{(l)}, b_i)}{n_i} \right)_{i,g(l,j)} \quad \text{for} \quad \begin{array}{l} i \in \{1,\ldots,d\}\,, \\ l \in \{1,\ldots,M\}\,, j \in \{1,\ldots,q(I,l)\} \end{array} \ .$$

Here, $g(l,j)$ is a suitable enumeration of the variables $v_j^{(l)}$, which correspond to the columns of $\tilde{A}$. We can now write Problem (2.2) as

$$\max \lambda \quad \text{s. t.} \quad \tilde{A}v = f(v) = \begin{pmatrix} f_1(v) \\ \vdots \\ f_d(v) \end{pmatrix} \geq \lambda \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \quad v \in B \ , \tag{2.3}$$

where $(1,\ldots,1)^{\mathrm{T}}$ is the d-dimensional 1-vector and $f_i(v)$ is the i.th row of the vector $\tilde{A}v$. The Inequality (2.3) therefore means that every entry of the vector $\tilde{A}v \in \mathbb{R}^d$ is larger than or equal to λ.

The Problem (2.2) is obviously a max-min resource sharing problem (see Subsection 1.2.3). Let λ_0 be the optimal value of (2.2). The algorithm by Grigoriadis et al. [33, 41] (see also Section A.1) finds for a given $\bar{\varepsilon} \in \Theta(\varepsilon)$ a solution $\tilde{\tilde{v}} \in B$ of Problem (2.2) with

$$\sum_{l=1}^{M} \sum_{j=1}^{q(I,l)} \frac{a(K_j^{(l)}, b_i)}{n_i} \tilde{\tilde{v}}_j^{(l)} \geq (1-\bar{\varepsilon})\,\lambda_0 \quad \text{for } i \in \{1,\ldots,d\} \ .$$

Using the notation of (2.3), this is

$$\tilde{A}\tilde{\tilde{v}} \geq (1-\bar{\varepsilon})\lambda_0 (1,\ldots,1)^{\mathrm{T}} \ . \tag{2.4}$$

(We will see below how to scale $\tilde{\tilde{v}}$ to get a solution $\tilde{v}$ with $\tilde{A}\tilde{v} \geq \lambda_0(1,\ldots,1)^{\mathrm{T}}$.) Let $\Gamma = \{p \in \mathbb{R}^M_{\geq 0} | \sum_i p_i = 1\}$ be the M-dimensional standard simplex. The block problem is

$$\Lambda(p) = \max_{v \in B} p^T f(v) = \max_{v \in B} p^T \tilde{A} v \text{ for } p \in \Gamma \ . \tag{2.5}$$

As stated in Subsection 1.2.3, the max-min resource sharing algorithm has to approximately solve in each of its iterations the block problem for a given $p \in \Gamma$. It does so by calling a blocksolver $ABS(p, \frac{\bar{\varepsilon}}{6})$ that finds an approximate solution $v \in B$ to $\Lambda(p)$ with

$$p^{\mathrm{T}} f(v) \geq \left(1 - \frac{\bar{\varepsilon}}{6}\right) \Lambda(p) \ .$$

For (2.2), the Block Problem (2.5) corresponds to

$$\begin{aligned} \max_{v \in B} p^T \tilde{A} v &= \max_{v \in B} \sum_{i=1}^{d} p_i \sum_{l=1}^{M} \sum_{j=1}^{q(I,l)} \frac{a(K_j^{(l)}, b_i)}{n_i} v_j^{(l)} \\ &= \max_{v \in B} \sum_{l=1}^{M} \sum_{j=1}^{q(I,l)} c_l v_j^{(l)} \sum_{i=1}^{d} \frac{p_i}{n_i c_l} a(K_j^{(l)}, b_i) \\ &\quad \text{with } \sum_{l=1}^{M} \sum_{j=1}^{q(I,l)} c_l v_j^{(l)} = r \ . \end{aligned} \tag{2.6}$$

This problem is a linear program, where an optimum basic solution v has only one entry $v_j^{(l)} > 0$ (see Subsection 1.2.1). In fact, it is sufficient to find the index pair l_1 and j_1 such that the sum $\sum_{i=1}^{d} \frac{p_i}{n_i c_{l_1}} a(K_{j_1}^{(l_1)}, b_i)$ is maximal and to set $v_{j_1}^{(l_1)} = \frac{r}{c_{l_1}}$ and the other $v_j^{(l)} = 0$. Choosing the right bin size c_l and the configuration $K_j^{(l)}$ therefore corresponds to solving the problem

$$\max_{l \in \{1,\ldots,M\}} \max \left\{ \sum_{i=1}^{d} \frac{p_i}{n_i c_l} z_i \,\middle|\, \sum_{i=1}^{d} b_i z_i \leq c_l, z_i \in \mathbb{N} \cup \{0\} \right\} \ . \tag{2.7}$$

This is indeed an instance of the Unbounded Knapsack Problem with Inversely Proportional Profits (UKPIP) (see Subsection 1.3.3) for which an FPTAS is presented in the Chapters 3 and 5. We get the following result:

Remark 2.11. During the execution of the max-min resource sharing solver, the blocksolver directly generates the columns needed for an approximate solution to the max-min resource sharing problem (2.2). It does so by finding in each iteration an $(1 - \frac{\bar{\varepsilon}}{6})$ approximate solution to the UKPIP instance (2.7).

As mentioned above, we have to "guess" the right value $r = \mathrm{LIN}(I, C)$. It is even sufficient that $r = r_0 \leq \mathrm{LIN}(I, C)$ as long as the max-min resource sharing algorithm

finds a solution $\tilde{\tilde{v}} = (\tilde{\tilde{v}}_j^{(l)})$ with $\tilde{A}\tilde{\tilde{v}} \geq (1-\bar{\varepsilon})(1,\ldots,1)^{\mathrm{T}}$. Then, the covering constraints $\sum_l \sum_j a(K_j^{(l)}, b_i) v_j^{(l)} \geq n_i$ can be satisfied by scaling: set $\tilde{v} := (1+4\bar{\varepsilon})\tilde{\tilde{v}}$ to get

$$\tilde{A}\tilde{v} \geq (1+4\bar{\varepsilon})(1-\bar{\varepsilon})(1,\ldots,1)^{\mathrm{T}} \geq (1,\ldots,1)^{\mathrm{T}} \tag{2.8}$$

and

$$\sum_{l=1}^{M} \sum_{j=1}^{q(I,l)} c_l \tilde{v}_j^{(l)} = (1+4\bar{\varepsilon})\, r_0 \leq (1+\varepsilon)\, \mathrm{LIN}(I,C) \tag{2.9}$$

for $\varepsilon \leq \frac{1}{2}$ and $\bar{\varepsilon} = \frac{\varepsilon}{4}$. Therefore, $\tilde{v} = (\tilde{v}_j^{(l)})$ is indeed a $(1+\varepsilon)$ approximate solution to the LP relaxation of (ILP-VBP).

Such an r_0 can be found by exploiting an interesting property of the max-min resource sharing algorithm. It was already stated in an unpublished paper by Aizatulin, Diedrich and Jansen [1] for the max-min resource sharing formulation of the Strip Packing Problem, which is closely related to the Bin Packing Problem (see also [14, p. 112]).

Lemma 2.12. *Let $r = 1$, and let v^1 be the corresponding solution to the max-min resource sharing problem* (2.2) *found by the algorithm of Grigoriadis et al. [33]. Let v^0 be the solution obtained with $r = r_p \in \mathbb{R}_{>0}$. Then $v^0 = r_p \cdot v^1$.*

Proof. The proof can be found in Appendix A.2. □

Thus, the solutions $\tilde{\tilde{v}}_1$ for $r = 1$ and $\tilde{\tilde{v}}_{r_p}$ for $r = r_p := \mathrm{LIN}(I,C)$ satisfy

$$\begin{aligned} r_p \cdot (f_1(\tilde{\tilde{v}}_1),\ldots,f_d(\tilde{\tilde{v}}_1))^{\mathrm{T}} = r_p \cdot \tilde{A}\tilde{\tilde{v}}_1 = \tilde{A}\tilde{\tilde{v}}_{r_p} &\geq (1-\bar{\varepsilon})\, \lambda_{r_p}\, (1,\ldots,1)^{\mathrm{T}} \\ &\geq (1-\bar{\varepsilon})\, (1,\ldots,1)^{\mathrm{T}} \quad . \end{aligned} \tag{2.10}$$

Here, λ_{r_p} denotes the optimal value of Problem (2.2) for $r = r_p$; note that we have $\lambda_{r_p} \geq 1$ according to Remark 2.10. We deduce that $\frac{1-\bar{\varepsilon}}{\min\{f_1(\tilde{\tilde{v}}_1),\ldots,f_d(\tilde{\tilde{v}}_1)\}} \leq r_p = \mathrm{LIN}(I,C)$ so that it is sufficient to set $r_0 := \frac{1-\bar{\varepsilon}}{\min\{f_1(\tilde{\tilde{v}}_1),\ldots,f_d(\tilde{\tilde{v}}_1)\}} \leq \mathrm{LIN}(I,C)$. A short calculation together with the Inequalities (2.8) and (2.9) yields that

$$\tilde{v} := (1+4\bar{\varepsilon})\tilde{\tilde{v}} := (1+4\bar{\varepsilon}) r_0 \cdot \tilde{\tilde{v}}_1$$

is a solution to (2.2). Note that $\tilde{A}\tilde{\tilde{v}} \geq (1-\varepsilon)\,(1,\ldots,1)^T$ holds by the definition of r_0.

We get the main result of this subsection.

Theorem 2.13. *Let $\varepsilon > 0$, $\bar{\varepsilon} := \frac{\varepsilon}{4}$, and let $(I^{(k)}, C^{(1)})$ be a VBP instance with M_1 bin sizes, d_k item sizes and smallest bin size $c_{\min}$. Moreover, let $UKPIP(d_k, M_1, c_{\min}, \frac{\bar{\varepsilon}}{6})$ be the running time of an algorithm for UKPIP with the approximation ratio $(1-\frac{\bar{\varepsilon}}{6})$ that*

solves an instance with d_k item sizes, M_1 knapsack sizes and smallest knapsack size $c_{\min}$. There is an algorithm that finds a solution $v = (v_j^{(l)})$ to the relaxed (ILP-VBP) *satisfying $\sum_l c_l \sum_j v_j^{(l)} \leq (1+\varepsilon)\mathrm{LIN}(I^{(k)}, C^{(1)})$ and with only $d_k + 1$ variables $v_j^{(l)} > 0$. The running time is in*

$$\mathcal{O}\Bigg(\max\left\{ UKPIP\left(d_k, M_1, c_{\min}, \frac{\bar{\varepsilon}}{6}\right), \mathcal{O}\left(d_k \log\log\left(d_k \frac{1}{\varepsilon}\right)\right)\right\} d_k \left(\log(d_k) + \frac{1}{\varepsilon^2}\right) + d_k^{2.5356}\left(\log d_k + \frac{1}{\varepsilon^2}\right)\Bigg) \ .$$

Proof. The correctness of the algorithm has already been presented in this subsection. Explanations for the other properties of the solution and the analysis of the running time can be found in Section A.2. □

2.5.2 Transforming $I_{\mathrm{res}}^{(k)}$

The transformation of $I_{\mathrm{res}}^{(k)}$ into J_k and J_k' in Step 2.3 of Alg_ε is the geometric rounding technique by Shmonin [80, Section 6.4]. It is the core element of his proof $\mathrm{OPT}(I) \leq \mathrm{LIN}(I) + \mathcal{O}(\log^2(d))$ for CSP. The rounding is a slight modification of the geometric grouping by Karmarkar and Karp [56].

Roughly speaking, J_k contains rounded-up items so that it has less item sizes than $I_{\mathrm{res}}^{(k)}$, and J_k' contains the largest and smallest items of $I_{\mathrm{res}}^{(k)}$ so that $J_k \leq I_{\mathrm{res}}^{(k)}$ holds even with the rounded-up items in J_k.

More formally, let $h_k := |I_{\mathrm{res}}^{(k)}|$. As will be seen in the proof of Theorem 2.21, the items are either already sorted or can be sorted in $\mathcal{O}(d_k \log(d_k) + \frac{d_k}{\delta})$ so that we have

$$I_{\mathrm{res}}^{(k)} = \left\{\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_{h_k}\right\} \text{ with } s(\tilde{a}_1) = \tilde{s}_1 \geq s(\tilde{a}_2) = \tilde{s}_2 \geq \ldots \geq s(\tilde{a}_{h_k}) = \tilde{s}_{h_k} \ .$$

We split this sequence into different groups: Let $G_1^{(k)}$ be the first l_1 items such that $\sum_{i=1}^{l_1-1} \tilde{s}_i \leq 2$, but $\sum_{i=1}^{l_1} \tilde{s}_i > 2$. Next, we collect l_2 items in group $G_2^{(k)}$ such that $\sum_{i=l_1+1}^{l_1+l_2-1} \tilde{s}_i \leq 2$, but $\sum_{i=l_1+1}^{l_1+l_2} \tilde{s}_i > 2$. We proceed until having grouped all items, i.e.

$$I_{\mathrm{res}}^{(k)} = \bigcup_{l=1}^{K_k} G_l^{(k)}$$

(the last group $G_{K_k}^{(k)}$ may have a total size smaller than 2, i.e. $\sum_{\tilde{a} \in G_{K_k}^{(k)}} s(\tilde{a}) \leq 2$).

Now, let $k_l^{(k)} := |G_l^{(k)}|$ for $l \in \{1, \ldots, K_k\}$. Since we added the items $\tilde{a}_i$ in non-increasing order to the sets $G_l^{(k)}$, we have $k_1^{(k)} \leq k_2^{(k)} \leq \ldots \leq k_{K_k-1}^{(k)}$. (The last group $G_{K_k}^{(k)}$ may contain less items than $G_{K_k-1}^{(k)}$, i.e. $k_{K_k}^{(k)} \leq k_{K_k-1}^{(k)}$ may hold.) Moreover, $k_l^{(k)} \leq \frac{2}{\delta}$

holds because $\tilde{s}_i \geq \delta$, and we have $K_k \leq \lfloor \frac{\text{Area}(I_{\text{res}}^{(k)})}{2} \rfloor + 1$ because $\text{Area}(G_l^{(k)}) > 2$ for $l \in \{1, \ldots, K_k - 1\}$.

For $l \in \{2, \ldots, K_k - 1\}$, let $G_l'^{(k)}$ contain the largest $k_{l-1}^{(k)}$ items of $G_l^{(k)}$. Let $H_l^{(k)}$ be the set of items obtained by rounding up the sizes of all items in $G_l'^{(k)}$ to the largest item size in $G_l'^{(k)}$. We have $G_l'^{(k)} \leq H_l^{(k)} \leq G_{l-1}^{(k)}$ for $l \in \{2, \ldots, K_k - 1\}$. We define

$$J_k := \bigcup_{l=2}^{K_k-1} H_l^{(k)} \quad \text{and} \quad J_k' := G_1^{(k)} \cup G_{K_k}^{(k)} \cup \bigcup_{l=2}^{K_k-1} \left(G_l^{(k)} \setminus G_l'^{(k)} \right). \tag{2.11}$$

Lemma 2.14. *Let d_k be the number of different item sizes that instance $I^{(k)}$ has in the execution of $Alg_\varepsilon(I^{(1)}, C^{(1)})$, and let k_0 be the number of times the main loop 2 of Alg_ε is executed. The following properties hold:*

- $(J_k, C^{(1)}) \leq (I_{\text{res}}^{(k)}, C^{(1)}) \leq (J_k \cup J_k', C^{(1)})$ *for* $k \in \{1, \ldots, k_0\}$.
- *For* $k \in \{1, \ldots, k_0 - 1\}$, *the set* $I^{(k+1)} = J_k$ *has* $d_{k+1} \leq K_k - 1 \leq \lfloor \frac{\text{Area}(I_{\text{res}}^{(k)})}{2} \rfloor - 1$ *different item sizes.*
- *The latest moment Alg_ε terminates is when* $\text{Area}(I_{\text{res}}^{(k)}) \leq 4$.

Proof. It is not difficult to see that $(J_k, C^{(1)}) \leq (I_{\text{res}}^{(k)}, C^{(1)}) \leq (J_k \cup J_k', C^{(1)})$ holds. Moreover, every $H_l^{(k)}$ for $l \in \{2, \ldots, K_k - 1\}$ contains only items of one (rounded-up) size, and $K_k \leq \lfloor \frac{\text{Area}(I_{\text{res}}^{(k)})}{2} \rfloor + 1$ holds. As $J_k = \bigcup_{l=2}^{K_k-1} H_l^{(k)}$, the number of different item sizes d_{k+1} follows.

Note that a sufficient condition for the termination of Alg_ε is $\text{Area}(I_{\text{res}}^{(k)}) \leq 4$: by definition of J_k', all items in $I_{\text{res}}^{(k)}$ will then be contained in J_k' so that $J_k' = I_{\text{res}}^{(k)}$ and $J_k = \emptyset$ hold. This is exactly a stopping criterion for Alg_ε as explained in Remark 2.9. □

2.5.3 Analysis and Running Time of the Basic Algorithm

In this section, we prove that $Alg_\varepsilon(I^{(1)}, C^{(1)})$ finds a packing for a given instance $(I^{(1)}, C^{(1)})$ that is not much larger than the optimum $\text{OPT}(I^{(1)}, C^{(1)})$.

Theorem 2.15. *Let $(I^{(1)}, C^{(1)})$ be a VBP instance with d_1 different item sizes and $s(a) \geq \delta > 0$ for $a \in I^{(1)}$. Algorithm $Alg_\varepsilon(I^{(1)}, C^{(1)})$ finds a packing $v = (v_j^{(l)})$ such that*

$$\sum_{l=1}^{M} c_l \sum_j v_j^{(l)} \leq (1 + 4\varepsilon)\,\text{OPT}(I^{(1)}, C^{(1)}) + \mathcal{O}\left(\log\left(\frac{1}{\delta}\right) \log(d_1) \right) .$$

For the proof, we need some lemmas.

Lemma 2.16. *Let d_k be the number of different item sizes in $I^{(k)}$ and k_0 the number of iterations of the main loop 2 of Alg_ε. Then we have*

1. $\text{Area}(I_{\text{res}}^{(k)}) \leq d_k + 1$ *for* $k \in \{1, \ldots, k_0\}$,
2. $\text{Area}(I_{\text{res}}^{(k)}) \leq d_k + 1 \leq \frac{1}{2}\text{Area}(I_{\text{res}}^{(k-1)}) \leq \frac{d_{k-1}}{2} + \frac{1}{2}$ *for* $k \in \{2, \ldots, k_0\}$,
3. $d_k \leq \frac{d_1}{2^{k-1}}$ *for* $k \in \{1, \ldots, k_0\}$, *and*
4. $\text{Area}(I_{\text{res}}^{(k)}) \leq \frac{\text{Area}(I_{\text{res}}^{(1)})}{2^{k-1}}$ *for* $k \in \{1, \ldots, k_0\}$ *and* $\text{Area}(I^{(k)}) \leq \frac{\text{Area}(I^{(1)})}{2^{k-2}}$ *for* $k \in \{2, \ldots, k_0\}$.

Proof. Let $v^{(k)} = (v_j^{(l,k)})$ be the approximate solution to the relaxed ILP (ILP-VBP) corresponding to $I^{(k)}$. As stated in Theorem 2.13, there are at most $d_k + 1$ variables $v_j^{(l,k)} > 0$. Thus, there are at most $d_k + 1$ variables $v_j^{(l,k)} - \lfloor v_j^{(l,k)} \rfloor > 0$, which form a fractional packing of $I_{\text{res}}^{(k)}$ and correspond to at most $d_k + 1$ bins because we have $v_j^{(l,k)} - \lfloor v_j^{(l,k)} \rfloor < 1$. Since every bin is at most of size 1, we get the first inequality. The second inequality follows from the first one and Lemma 2.14. The two remaining inequalities are proved by induction and the fact that $I^{(k)} = J_{k-1} \leq I_{\text{res}}^{(k-1)} \leq I^{(k-1)}$ (see Lemma 2.14). □

Lemma 2.17. *As above, let k_0 be the number of iterations of the main loop 2 in Alg_ε. Then $k_0 \leq \lfloor \log_2(d_1 + 1) \rfloor$ holds.*

Proof. Lemma 2.14 already states that the algorithm terminates when $\text{Area}(I_{\text{res}}^{(k)}) \leq 4$. A direct corollary of Lemma 2.16 (obtained by combining the first and last inequality) is the inequality $\text{Area}(I_{\text{res}}^{(k)}) \leq \frac{1}{2^{k-1}}(d_1 + 1)$ for $k \in \{1, \ldots, k_0\}$. If we want $\text{Area}(I_{\text{res}}^{(k)}) \leq 4$, it is therefore sufficient that $\frac{1}{2^{k-1}}(d_1 + 1) \leq 4$ holds, i.e. $d_1 + 1 \leq 2^{k+1}$ and therefore $\log_2(d_1 + 1) \leq k + 1$. The smallest $k \in \mathbb{N}$ satisfying this inequality is $\lfloor \log_2(d_1 + 1) \rfloor$, therefore $k_0 \leq \lfloor \log_2(d_1 + 1) \rfloor$ holds. □

Lemma 2.18. *Let the set J'_k be defined as in Subsection 2.5.2. Next-Fit packs J'_k in at most $\mathcal{O}(\log(\frac{1}{\delta}))$ unit-sized bins.*

Proof. By the grouping of $I_{\text{res}}^{(k)}$ into $G_1^{(k)}, \ldots, G_{K_k}^{(k)}$ as presented in Subsection 2.5.2, we know that $\text{Area}(G_1^{(k)} \cup G_{K_k}^{(k)}) \leq 6$. Let $i \in \{2, \ldots, K_k - 1\}$. The first $k_i^{(k)} - 1$ items of $G_i^{(k)}$ have a total volume of at most 2 by the definition of $G_i^{(k)}$, and the last item may be even smaller than the other items. The average item size in $G_i^{(k)}$ is therefore at most

$\frac{2}{k_i^{(k)}-1}$. Since $\Delta G_i^{(k)} := G_i^{(k)} \setminus G_i'^{(k)}$ contains the smallest $k_i^{(k)} - k_{i-1}^{(k)}$ items of $G_i^{(k)}$, we have $\text{Area}(\Delta G_i^{(k)}) \leq 2\frac{k_i^{(k)}-k_{i-1}^{(k)}}{k_i^{(k)}-1}$. We conclude that

$$\begin{aligned}
\text{Area}(J_k') &\leq 6 + \sum_{i=2}^{K_k-1} \text{Area}(\Delta G_i^{(k)}) \leq 6 + 2\sum_{i=2}^{K_k-1} \frac{k_i^{(k)} - k_{i-1}^{(k)}}{k_i^{(k)} - 1} \\
&\leq 6 + 2\sum_{i=2}^{K_k-1} \sum_{j=k_{i-1}^{(k)}}^{k_i^{(k)}-1} \frac{1}{j} = 6 + \sum_{j=k_1^{(k)}}^{k_{K_k-1}^{(k)}-1} \frac{1}{j} \leq 6 + 2\ln(k_{K_k-1}^{(k)}) \\
&\leq 6 + 2\ln\left(\frac{2}{\delta}\right)
\end{aligned}$$

where we have used that $k_l^{(k)} \leq \frac{2}{\delta}$ holds for $l \in \{1, \ldots, K_k - 1\}$ because we have $s(a) \geq \delta$. Therefore, we need at most $2\text{Area}(J_k') + 1 \leq 13 + 4\ln(\frac{2}{\delta}) = \mathcal{O}(\log(\frac{1}{\delta}))$ bins to pack J_k' with Next-Fit.

(This proof is a slight modification of the proof of Theorem 6.7 in [80], which is derived from the geometric grouping presented in [56]. The only difference is the fact that we have $s(a) \geq \delta$ in contrast to $s(a) \geq \frac{1}{d_k}$.) □

Lemma 2.19. *The following inequalities hold:*

- $\text{LIN}(I^{(k)}, C^{(1)}) - \text{LIN}(I_{\text{res}}^{(k)}, C^{(1)}) \leq \text{LIN}(I_{\text{int}}^{(k)}, C^{(1)})$,
- $\text{LIN}(I^{(k+1)}, C^{(1)}) \leq \text{LIN}(I_{\text{res}}^{(k)}, C^{(1)})$, *and*
- $\text{LIN}(I^{(k)}, C^{(1)}) - \text{LIN}(I_{\text{res}}^{(k)}, C^{(1)}) \leq \text{LIN}(I^{(k)}, C^{(1)}) - \text{LIN}(I^{(k+1)}, C^{(1)})$.

Proof. An optimal (fractional or integral) packing of $(I_{\text{int}}^{(k)}, C^{(1)})$ together with an optimal packing of $(I_{\text{res}}^{(k)}, C^{(1)})$ is a feasible packing of $(I^{(k)}, C^{(1)})$, therefore we have $\text{LIN}(I^{(k)}, C^{(1)}) \leq \text{LIN}(I_{\text{int}}^{(k)}, C^{(1)}) + \text{LIN}(I_{\text{res}}^{(k)}, C^{(1)})$, which proves the first inequality. Since $(I^{(k+1)}, C^{(1)}) = (J_k, C^{(1)}) \leq (I_{\text{res}}^{(k)}, C^{(1)})$ holds according to Lemma 2.14, the second inequality follows from Lemma 2.8. The last inequality follows directly from the second one. □

We are now able to prove Theorem 2.15, i.e. the bound on the solution quality of Alg_ε.

Proof of Theorem 2.15. Let $\text{Bins}(L)$ denote the volume of the bins into which the algorithm has packed the instance $(L, C^{(1)})$. Obviously, the solution found by Alg_ε has the total objective value $\sum_{k=1}^{k_0}(\text{Bins}(I_{\text{int}}^{(k)}) + \text{Bins}(J_k'))$ because items are either packed by an integral part $I_{\text{int}}^{(k)}$ or by J_k' (see Remark 2.9).

First, we derive a bound on $\text{Bins}(I_{\text{int}}^{(k)})$. For an instance $(I^{(k)}, C^{(1)})$ with the approximate packing $v^{(k)} = (v_j^{(l,k)})$ found by Alg_ε in Step 2.1, we have the following inequality:

$$\begin{aligned}\sum_{l=1}^{M} c_l \sum_{j=1}^{q(I^{(k)},l)} v_j^{(l,k)} &= \sum_{l=1}^{M} c_l \sum_{j=1}^{q(I^{(k)},l)} \left\lfloor v_j^{(l,k)} \right\rfloor + \sum_{l=1}^{M} c_l \sum_{j=1}^{q(I^{(k)},l)} \left(v_j^{(l,k)} - \left\lfloor v_j^{(l,k)} \right\rfloor \right) \\ &\leq (1+\varepsilon)\,\text{LIN}(I^{(k)}, C^{(1)}) \ .\end{aligned} \tag{2.12}$$

Since $(v_j^{(l,k)} - \lfloor v_j^{(l,k)} \rfloor)$ is a fractional packing of $(I_{\text{res}}^{(k)}, C^{(1)})$ with $\text{LIN}(I_{\text{res}}^{(k)}, C^{(1)}) \leq \sum_l c_l \sum_j (v_j^{(l,k)} - \lfloor v_j^{(l,k)} \rfloor)$, we get

$$\begin{aligned}\text{Bins}\left(I_{\text{int}}^{(k)}\right) &= \sum_{l=1}^{M} c_l \sum_{j=1}^{q(I^{(k)},l)} \left\lfloor v_j^{(l,k)} \right\rfloor \\ &\overset{(2.12)}{\leq} (1+\varepsilon)\,\text{LIN}(I^{(k)}, C^{(1)}) - \sum_{l=1}^{M} c_l \sum_{j=1}^{q(I^{(k)},l)} \left(v_j^{(l,k)} - \left\lfloor v_j^{(l,k)} \right\rfloor \right) \\ &\leq (1+\varepsilon)\,\text{LIN}(I^{(k)}, C^{(1)}) - \text{LIN}(I_{\text{res}}^{(k)}, C^{(1)}) \\ &\overset{\text{Lem. 2.19}}{\leq} (1+\varepsilon)\,\text{LIN}(I^{(k)}, C^{(1)}) - \text{LIN}(I^{(k+1)}, C^{(1)}) \ .\end{aligned} \tag{2.13}$$

We deduce with $\text{LIN}(I_{\text{res}}^{(k_0)}, C^{(1)}) \geq 0$ and $(I^{(3)}, C^{(1)}) \leq (I^{(2)}, C^{(1)}) \leq (I^{(1)}, C^{(1)})$ that

$$\begin{aligned}\sum_{k=1}^{k_0} \text{Bins}\left(I_{\text{int}}^{(k)}\right) &\leq \sum_{k=1}^{k_0-1} \left[(1+\varepsilon)\,\text{LIN}(I^{(k)}, C^{(1)}) - \text{LIN}(I^{(k+1)}, C^{(1)}) \right] \\ &\quad + (1+\varepsilon)\,\text{LIN}(I^{(k_0)}, C^{(1)}) - \text{LIN}(I_{\text{res}}^{(k_0)}, C^{(1)}) \\ &= \text{LIN}(I^{(1)}, C^{(1)}) + \varepsilon \sum_{k=1}^{k_0} \text{LIN}(I^{(k)}, C^{(1)}) - \text{LIN}(I_{\text{res}}^{(k_0)}, C^{(1)}) \\ &\leq \text{LIN}(I^{(1)}, C^{(1)}) + 3\,\varepsilon\,\text{LIN}(I^{(1)}, C^{(1)}) + \varepsilon \sum_{k=4}^{k_0} \text{LIN}(I^{(k)}, C^{(1)}) \ .\end{aligned} \tag{2.14}$$

$\text{LIN}(I^{(k)}, C^{(1)}) \leq (2\text{Area}(I^{(k)}) + 1)$ holds because First-Fit could always provide a packing for $I^{(k)}$ of this value. Moreover, we have $2\text{Area}(I^{(k)}) \leq \frac{2}{2^{k-2}}\text{Area}(I^{(1)}) = \frac{1}{2^{k-3}}\text{Area}(I^{(1)})$ according to Lemma 2.16. Hence, the Sum (2.14) can be bounded as follows:

$$\begin{aligned}\sum_{k=1}^{k_0} \text{Bins}\left(I_{\text{int}}^{(k)}\right) &\leq \text{LIN}(I^{(1)}, C^{(1)}) + 3\,\varepsilon\,\text{LIN}(I^{(1)}, C^{(1)}) + \varepsilon \sum_{k=4}^{k_0} \left[\frac{1}{2^{k-3}}\text{Area}(I^{(1)}) + 1 \right] \\ &\leq \text{LIN}(I^{(1)}, C^{(1)}) + 3\,\varepsilon\,\text{LIN}(I^{(1)}, C^{(1)}) + \varepsilon\,\text{LIN}(I^{(1)}, C^{(1)}) + \varepsilon\,(k_0 - 3) \\ &\leq (1+4\varepsilon)\,\text{LIN}(I^{(1)}, C^{(1)}) + \varepsilon \log_2(d_1 + 1) \ .\end{aligned} \tag{2.15}$$

For the last two inequalities, we have used $\text{Area}(I^{(1)}) \leq \text{LIN}(I^{(1)}, C^{(1)})$ and $k_0 \leq \log_2(d_1 + 1)$ (see Lemma 2.17).

Finally, $\sum_{k=1}^{k_0} \text{Bins}(J'_k) \in \mathcal{O}(\log(\frac{1}{\delta}) \log(d_1))$ holds because we have $\text{Bins}(J'_k) \in \mathcal{O}(\log(\frac{1}{\delta}))$ (see Lemma 2.18) and $k_0 \leq \log_2(d_1 + 1)$. By combining this with the Bound (2.15) and $\text{LIN}(I^{(1)}, C^{(1)}) \leq \text{OPT}(I^{(1)}, C^{(1)})$, we obtain the desired result. □

Remark 2.20. The relaxed ILPs of the instances $(I^{(k)}, C^{(1)})$ are only approximately solved in contrast to Shmonin's proof [80] where optimal solutions are used. This results in the additional additive terms $\varepsilon \text{LIN}(I^{(k)}, C^{(1)})$ and $\varepsilon \sum_k \text{LIN}(I^{(k)}, C^{(1)})$ in (2.13) and (2.14). Thus, it is necessary to prove that the sum is bounded by $\sum_k \frac{1}{2^{k-3}} \text{Area}(I^{(k)})$, which yields the final estimate (2.15).

After the proof of the approximation ratio of the algorithm, we only need to bound the running time.

Theorem 2.21. *Let $\varepsilon > 0$, and let $(I^{(1)}, C^{(1)})$ be a VBP instance with d_1 item sizes, M_1 bin sizes and $s(a) \geq \delta$ for $a \in I^{(1)}$. Then Alg_ε has a running time in*

$$\begin{aligned} \mathcal{O}\Bigg(\max\left\{ UKPIP\left(d_1, M_1, c_{\min}, \frac{\bar{\varepsilon}}{6}\right), \mathcal{O}\left(d_1 \log\log\left(d_1 \frac{1}{\varepsilon}\right)\right)\right\} \\ \cdot d_1 \left(\log(d_1) + \frac{1}{\varepsilon^2}\right) + d_1^{2.5356}\left(\log(d_1) + \frac{1}{\varepsilon^2}\right) + \frac{d_1}{\delta} + n\Bigg) \ . \end{aligned} \tag{2.16}$$

Proof. We show how to bound the running time over all iterations of the main loop 2. In every loop iteration, the relaxed ILP is solved in Step 2.1. The complexity of the solver in one iteration is given in Theorem 2.13 (see also Section A.2 in the Appendix). We assume that the running time $UKPIP(d, M, c_{\min}, \frac{\bar{\varepsilon}}{6})$ of the knapsack solver is monotonic increasing in d. We know that $d_k \leq \frac{d_1}{2^{k-1}}$ as seen in Lemma 2.16. Thus, the running time of Step 2.1 in the k.th iteration is at most half as long as the running time in the previous iteration because every summand of the running time either contains d_k or is dominated by other summands. Hence, the overall sum and therefore the overall running time of Step 2.1 is still bounded by the term stated in Theorem 2.13.

The next Step 2.2 consists of packing the items according to $(\lfloor v_j^{(l,k)} \rfloor)$. There are only $d_k + 1$ variables $v_j^{(l,k)} > 0$ (see Theorem 2.13) so that $d_k + 1$ times a configuration vector $K_j^{(l,k)}$ with d_k entries has to be read and the items packed accordingly. The running time for reading the configuration vectors is bounded by $\sum_{k=1}^{k_0} \mathcal{O}(d_k^2) = \mathcal{O}(d_1^2)$ over all iterations of Step 2. Since there are at most n items that can be packed over all iterations, we get a total running time in $\mathcal{O}(d_1^2 + n)$.

Constructing J_k and J'_k in Step 2.3 can be performed in the following way: we only have to sort the d_k different item sizes of $I_{\text{res}}^{(k)}$ and then group the items in $I_{\text{res}}^{(k)}$ according to the item sizes (i.e. we use BucketSort). As $\text{Area}(I_{\text{res}}^{(k)}) \leq d_k + 1$ (see Lemma 2.16) and $s(a) \geq \delta$, the set $I_{\text{res}}^{(k)}$ has $\mathcal{O}(\frac{d_k}{\delta})$ items so that sorting $I_{\text{res}}^{(k)}$ and partitioning it into J_k and J'_k can be performed in $\mathcal{O}(d_k \log(d_k) + \frac{d_k}{\delta})$. (By using the right data structure, we can even assume that sorting is only necessary for $k = 1$ because the items stay in the right order afterwards.) Since Next-Fit has a linear running time, packing the items in J'_k can be bounded by $\mathcal{O}(n)$ over all iterations. The running time of Steps 2.3 and 2.4 is therefore bounded by $\mathcal{O}(d_1 \log(d_1) + \frac{d_1}{\delta} + n)$.

Replacing the rounded-up sizes by their original ones in Step 3 can be performed in time $\mathcal{O}(n)$. By omitting dominated summands, we get the general running time of Alg_ε. □

2.6 The General Algorithm

For a general VBP instance (I, C), we may not have $\mathcal{O}(\log(\frac{1}{\delta})\log(d_1)) = \mathcal{O}(\log^2 \frac{1}{\varepsilon})$ for the additive term, e.g. if the smallest item size $s(a)$ does not satisfy $s(a) \geq \varepsilon$ and if we have too many different item sizes d_1. Therefore, we preprocess the items and the bin sizes of a general instance (I, C) to apply Alg_ε.

2.6.1 Rounding the Items

We first show how to bound the number of different item sizes by $\mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$.

Let $\delta := \varepsilon^2$ and

$$I = I_{\text{large}} \cup I_{\text{small}} := \{a \in I \mid s(a) > \delta\} \cup \{a \in I \mid s(a) \leq \delta\} \ .$$

We now use the geometric grouping introduced by Karmarkar and Karp [56] with the scaling factor $k = \lceil \frac{\varepsilon \text{Area}(I)}{\log_2(\frac{1}{\delta})+1} \rceil$ to group and round I_{large} (see also [72, p. 298]).

First, the items in I_{large} are grouped into intervals

$$I_r := \left\{ a \mid s(a) \in (2^{-(r+1)}, 2^{-r}] \right\} \ \text{ for } \ r \in \left\{ 0, 1, \ldots, \left\lfloor \log_2\left(\frac{1}{\delta}\right) \right\rfloor \right\} \ .$$

Each set I_r is then further grouped into sets $I_{r,1}, I_{r,2}, \ldots, I_{r,q(r)}$ such that the first group $I_{r,1}$ contains the first $k \cdot 2^r$ largest items, $I_{r,2}$ the next $k \cdot 2^r$ largest items until all items have been grouped. Note that $I_{r,q(r)}$ may contain less than $k \cdot 2^r$ items. Now, we set $I_{\text{huge}} := \bigcup_r I_{r,1}$. On the other hand, let $j \in \{2, \ldots, q(r)\}$ and round every item in $I_{r,j}$ up to the largest item in the group. Call the resulting group $I'_{r,j}$. We then set $I^{(1)} := \bigcup_r \bigcup_{j=2}^{q(r)} I'_{r,j}$.

As shown in [56, Proof of Lemma 5], the items in I_{huge} can be packed in at most $k \cdot \lceil \log \frac{1}{\delta} \rceil \leq \varepsilon \operatorname{Area}(I) + \mathcal{O}(\log(\frac{1}{\delta}))$ unit-sized bins with Next-Fit if the items of I_{huge} in $(2^{-(r+1)}, 2^{-r}]$ for $r = 0$ are packed first, then the items for $r = 1, 2, \ldots, \lfloor \log_2(\frac{1}{\delta}) \rfloor$ (this can be seen as a less strict variant of Next-Fit Decreasing).

Since I_{huge} contains the largest items of every I_r, we still have $I^{(1)} \leq I_{\text{large}}$. Due to the rounding and the choice of k, the item set $I^{(1)}$ has only

$$d_1 \leq \frac{2}{k}\operatorname{Area}(I) + \left\lceil \log_2 \frac{1}{\delta} \right\rceil \in \mathcal{O}\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right) \tag{2.17}$$

different item sizes [56, Lemma 5]. This will help us to bound the running time needed to solve the LP relaxation of (ILP-VBP).

2.6.2 Reducing the Bin Sizes

As the running time of Alg_ε also depends on the number of bin sizes (see (2.16)), we only use a subset of the M bin sizes in C. Set $\tilde{C} := \{c \in C \mid c \geq \varepsilon\}$. If $M \leq \lfloor \frac{2}{\varepsilon} \ln \frac{1}{\varepsilon} \rfloor + 1$, set $C^{(1)} := \tilde{C}$. If on the other hand $M > \lfloor \frac{2}{\varepsilon} \ln \frac{1}{\varepsilon} \rfloor + 1$, partition $\tilde{C}$ into

$$\bigcup_{l=0}^{\lfloor \frac{2}{\varepsilon} \ln \frac{1}{\varepsilon} \rfloor} \tilde{C}_l \quad \text{with} \quad \tilde{C}_l := \left\{ c \mid c \in \left((1+\varepsilon)^{-(l+1)}, (1+\varepsilon)^{-l} \right] \right\} \ ,$$

and let $C^{(1)}$ consist of the largest c in every $\tilde{C}_l$, i.e.

$$C^{(1)} := \bigcup_{l=0}^{\lfloor \frac{2}{\varepsilon} \ln \frac{1}{\varepsilon} \rfloor} \max \left\{ c \mid c \in \tilde{C}_l \right\}$$

(This construction of $C^{(1)}$ is mentioned in [78].) We first show that the reduced bin set $C^{(1)}$ does not increase the value of an optimal or approximate solution too much.

Lemma 2.22. *$C^{(1)}$ has only $M_1 \leq \lfloor \frac{2}{\varepsilon} \ln \frac{1}{\varepsilon} \rfloor + 1$ bin sizes and satisfies*

$$\begin{aligned} \operatorname{OPT}(I^{(1)}, C^{(1)}) &\leq \operatorname{OPT}(I_{\text{large}}, C^{(1)}) \leq (1+\varepsilon)\operatorname{OPT}(I_{\text{large}}, \tilde{C}) \\ &\leq (1+4\varepsilon)\operatorname{OPT}(I, C) + 2 \ . \end{aligned}$$

Proof. The bound $\ln(1+z) \geq z - z^2$ holds for $z \geq -\frac{1}{2}$. The number of knapsack sizes in $C^{(1)}$ is therefore in

$$\begin{aligned} \left\lfloor \log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right) \right\rfloor + 1 &= \left\lfloor \frac{\ln \frac{1}{\varepsilon}}{\ln 1 + \varepsilon} \right\rfloor + 1 \overset{\varepsilon > 0}{\leq} \left\lfloor \frac{\ln \frac{1}{\varepsilon}}{\varepsilon - \varepsilon^2} \right\rfloor + 1 \\ &= \left\lfloor \frac{\ln \frac{1}{\varepsilon}}{\varepsilon \cdot (1-\varepsilon)} \right\rfloor + 1 \overset{\varepsilon \leq 1/2}{\leq} \left\lfloor \frac{\ln \frac{1}{\varepsilon}}{\frac{1}{2}\varepsilon} \right\rfloor + 1 \\ &= \left\lfloor \frac{2}{\varepsilon} \ln \frac{1}{\varepsilon} \right\rfloor + 1 \in \mathcal{O}\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right) \ . \end{aligned}$$

By definition, we have $(I^{(1)}, C^{(1)}) \leq (I_{\text{large}}, C^{(1)})$. If $C^{(1)} = \tilde{C}$, the second inequality is also obvious. If $C^{(1)} \subsetneq \tilde{C}$, choose an optimal solution to $(I_{\text{large}}, \tilde{C})$ and let $\text{OPT}(I_{\text{large}}, \tilde{C}) = \sum_l c_l z_l$, where z_l denotes the number of bins c_l the chosen optimal solution uses. Every c_l is contained in one $\tilde{C}_{j_l} = \{c \mid c \in ((1+\varepsilon)^{-(j_l+1)}, (1+\varepsilon)^{-j_l}]\}$, and $C^{(1)}$ contains the largest bin size $\tilde{c}_{j_l}$ of every $\tilde{C}_{j_l}$. Replace every c_l by the corresponding $\tilde{c}_{j_l}$. Then $c_l \leq \tilde{c}_{j_l} \leq (1+\varepsilon)c_l$ so that we obtain a feasible solution to $(I_{\text{large}}, C^{(1)})$ with the objective value $\sum_l \tilde{c}_{j_l} z_l \leq \sum_l (1+\varepsilon) c_l z_l \leq (1+\varepsilon)\text{OPT}(I_{\text{large}}, \tilde{C})$. As an optimal solution to $(I_{\text{large}}, C^{(1)})$ may have an even smaller objective value, we have

$$\text{OPT}(I_{\text{large}}, C^{(1)}) \leq (1+\varepsilon)\,\text{OPT}(I_{\text{large}}, \tilde{C}) \ . \tag{2.18}$$

For the last inequality, let (I_{large}, C) be a VBP instance with items I_{large} and all bin sizes C, and let $v = (v_j^{(l)})$ be an optimal integral packing of it. Let $\tilde{I}_1$ be the items packed in the bins $c_l \geq \varepsilon$, i.e. in bins in $\tilde{C}$, and let $(\tilde{I}_1, \tilde{C})$ be the corresponding instance. Let $(\tilde{I}_2, \tilde{C}_2)$ be the instance with the items packed in the remaining smaller bin sizes $\tilde{C}_2 = C \setminus \tilde{C}$.

Note that we have

$$\text{OPT}(I_{\text{large}}, C) = \text{OPT}(\tilde{I}_1, \tilde{C}) + \text{OPT}(\tilde{I}_2, \tilde{C}_2) \ . \tag{2.19}$$

In fact, an optimal packing of $(\tilde{I}_1, \tilde{C})$ combined with an optimal packing of $(\tilde{I}_2, \tilde{C}_2)$ is obviously a feasible packing of (I_{large}, C), i.e. we have $\text{OPT}(I_{\text{large}}, C) \leq \text{OPT}(\tilde{I}_1, \tilde{C}) + \text{OPT}(\tilde{I}_2, \tilde{C}_2)$. On the other hand, the packing of $\text{OPT}(I_{\text{large}}, C)$ restricted to the bins $\tilde{C}$ (or $\tilde{C}_2$) is by definition a feasible packing of $(\tilde{I}_1, \tilde{C})$ (or $(\tilde{I}_2, \tilde{C}_2)$). Let $Pack(\tilde{I}_1, \tilde{C})$ and $Pack(\tilde{I}_2, \tilde{C}_2)$ be the objective values of these packings where obviously $Pack(\tilde{I}_1, \tilde{C}) + Pack(\tilde{I}_2, \tilde{C}_2) = \text{OPT}(I_{\text{large}}, C)$ holds. Then, we have $\text{OPT}(\tilde{I}_1, \tilde{C}) + \text{OPT}(\tilde{I}_2, \tilde{C}_2) \leq Pack(\tilde{I}_1, \tilde{C}) + Pack(\tilde{I}_2, \tilde{C}_2) = \text{OPT}(I_{\text{large}}, C)$ so that (2.19) follows.

Now, the items in $\tilde{I}_2$ are re-packed into new unit-sized bins using First-Fit: an additional new bin is opened if all new unit-sized bins are full. Let $k+1$ be the number of these new bins. Note that $s(a) \leq \varepsilon$ for $a \in \tilde{I}_2$ because these items were packed in bins of size $c_l \leq \varepsilon$. Since a new bin is only opened if no new item fits into the old ones, the first k bins must be filled at least up to $1-\varepsilon$, i.e. $\text{Area}(\tilde{I}_2) \geq k(1-\varepsilon)$. We get

$$k+1 \leq \frac{1}{1-\varepsilon}\text{Area}(\tilde{I}_2) + 1 \leq (1+2\varepsilon)\text{Area}(\tilde{I}_2) + 1 \leq (1+2\varepsilon)\text{OPT}(\tilde{I}_2, \tilde{C}_2) + 1$$

where we have used Lemma 2.7 and $\frac{1}{1-\varepsilon} \leq 1+2\varepsilon$ for $\varepsilon \leq \frac{1}{2}$. Now note that this new packing of $\tilde{I}_2$ into $k+1$ unit-sized bins together with the original packing of $\tilde{I}_1$

is a packing of I_{large} not using the bin sizes $c_l \leq \varepsilon$, i.e. it is a packing of the instance $(I_{\text{large}}, \tilde{C})$ with

$$\begin{aligned}
\text{OPT}(I_{\text{large}}, \tilde{C}) &\leq \text{OPT}(\tilde{I}_1, \tilde{C}) + k + 1 \\
&\leq \text{OPT}(\tilde{I}_1, \tilde{C}) + (1+2\varepsilon)\text{OPT}(\tilde{I}_2, \tilde{C}_2) + 1 \\
&\leq (1+2\varepsilon)\left(\text{OPT}(\tilde{I}_1, \tilde{C}) + \text{OPT}(\tilde{I}_2, \tilde{C}_2)\right) + 1 \\
&\overset{(2.19)}{=} (1+2\varepsilon)\text{OPT}(I_{\text{large}}, C) + 1 \\
&\leq (1+2\varepsilon)\text{OPT}(I, C) + 1 \ . \qquad (2.20)
\end{aligned}$$

As we assume that $\varepsilon \leq \frac{1}{2}$, we get

$$\begin{aligned}
\text{OPT}(I^{(1)}, C^{(1)}) &\leq \text{OPT}(I_{\text{large}}, C^{(1)}) \overset{(2.18)}{\leq} (1+\varepsilon)\,\text{OPT}(I_{\text{large}}, \tilde{C}) \\
&\overset{(2.20)}{\leq} (1+\varepsilon)\left[(1+2\varepsilon)\text{OPT}(I, C) + 1\right] \leq (1+4\varepsilon)\,\text{OPT}(I, C) + 2 \ .
\end{aligned}$$

(The second part of the proof is only a slight modification of the corresponding proof by Murgolo [69, Proof of Thm. 1, case 1].) □

The general algorithm A_ε including the preprocessing is presented in Algorithm 2.2.

Algorithm 2.2: The general algorithm A_ε for VBP

Input: Accuracy $\varepsilon > 0$, instance (I, C)

1 Preprocess instance (I, C) to obtain $(I_{\text{huge}}, C^{(1)})$, $(I^{(1)}, C^{(1)})$, and $(I_{\text{small}}, C^{(1)})$;

2 Pack instance $(I^{(1)}, C^{(1)})$ with the algorithm $Alg_\varepsilon(I^{(1)}, C^{(1)})$ (see Section 2.5 and Alg. 2.1);

3 Replace the rounded-up and packed items of $(I^{(1)}, C^{(1)})$ by their original counterparts to obtain a packing of $I_{\text{large}} \setminus I_{\text{huge}}$;

4 Pack the items of I_{huge} into unit-sized bins using Next-Fit where the items in $(2^{-(r+1)}, 2^{-r}]$ for $r = 0$ are packed first, then the items for $r = 1, 2 \ldots, \lfloor \log_2(\frac{1}{\delta}) \rfloor$;

5 Add the small items I_{small} greedily to the existing bins using Next-Fit: open a new unit-sized bin if an item cannot be added to the existing bins;

2.6.3 Analysis of the Approximation Ratio

We sum the results up to obtain the final asymptotic approximation ratio of A_ε.

Lemma 2.23. *Before adding the small items I_{small} in Step 5, algorithm A_ε has found an integral packing $v = (v_j^{(l)})$ for (I_{large}, C) with $\sum_l c_l \sum_j v_j^{(l)} \leq (1+17\varepsilon)\text{OPT}(I, C) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$.*

Proof. $\delta = \varepsilon^2$ and $\log(d_1) \in \mathcal{O}(\log(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})) \subseteq \mathcal{O}(\log(\frac{1}{\varepsilon}))$ hold. Moreover, we have $\mathrm{LIN}(I^{(1)}, C^{(1)}) \leq \mathrm{OPT}(I^{(1)}, C^{(1)}) \leq \mathrm{OPT}(I_{\mathrm{large}}, C^{(1)}) \leq \mathrm{OPT}(I, C^{(1)})$ as can be seen in Lemma 2.7. Combining these properties with Theorem 2.15, we deduce that $(I^{(1)}, C^{(1)})$ and therefore $(I_{\mathrm{large}} \setminus I_{\mathrm{huge}}, C^{(1)})$ is packed in at most

$$\begin{aligned}(1+4\varepsilon)\mathrm{OPT}(I^{(1)}, C^{(1)}) + \mathcal{O}\left(\log\left(\frac{1}{\delta}\right)\log(d_1)\right) \leq &(1+4\varepsilon)\mathrm{OPT}(I^{(1)}, C^{(1)}) \\ &+ \mathcal{O}\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)\end{aligned}$$

bins. Using Lemma 2.22, we see that this is at most

$$\begin{aligned}(1+4\varepsilon)\left[(1+4\varepsilon)\mathrm{OPT}(I, C) + 2\right] + \mathcal{O}\left(\log^2\left(\frac{1}{\varepsilon}\right)\right) \leq &(1+16\varepsilon)\mathrm{OPT}(I, C) \\ &+ \mathcal{O}\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)\end{aligned}$$

because of $\varepsilon \leq \frac{1}{2}$. As I_{huge} fits into at most $\varepsilon\mathrm{Area}(I_{\mathrm{large}}) + \mathcal{O}(\log(\frac{1}{\varepsilon})) \leq \varepsilon\mathrm{OPT}(I, C) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$ unit-sized bins (see Subsection 2.6.1), we obtain the desired approximation ratio. □

Theorem 2.24. *Let (I, C) be a VBP instance with n items. For given $\varepsilon \in (0, \frac{1}{2}]$, the algorithm A_ε finds a solution with $A_\varepsilon(I, C) \leq (1+17\varepsilon)\mathrm{OPT}(I, C) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$.*

Proof. Because of Lemma 2.23, the theorem is obviously true if Step 5 does not create any new bins. Hence, let us assume that at least one new (unit-sized) bin has to be opened. Note that the solution constructed by the algorithm only uses bins in $C^{(1)} \subset \tilde{C}$, i.e. bins with $c_l \geq \varepsilon$. The items in I_{small} satisfy $s(a) \leq \delta = \varepsilon^2$. As at least one unit-sized bin is opened, all bins (with the possible exception of the last opened unit-sized bin) are filled at least up to $c_l - \delta = c_l - \varepsilon^2$ at the end of Step 5. Let z_l be the number of bins $c_l \in C^{(1)}$ the solution has after adding the last (small) item, and let $\bar{C} := C^{(1)} \setminus \{c_M\} = C^{(1)} \setminus \{1\}$. Then we have with $\frac{1}{1-\varepsilon} \leq 1 + 2\varepsilon$ for $\varepsilon \leq \frac{1}{2}$ that

$$\begin{aligned}\mathrm{Area}(I) &\geq \sum_{c_l \in \bar{C}} \left(c_l - \varepsilon^2\right) z_l + \left(1 - \varepsilon^2\right)(z_M - 1) \\ &\overset{c_l \geq \varepsilon}{\geq} \sum_{c_l \in \bar{C}} c_l\,(1-\varepsilon)\,z_l + (1-\varepsilon)\,(z_M - 1) \\ \Rightarrow \sum_{c_l \in \bar{C}} c_l z_l + c_M z_M &\leq \frac{1}{1-\varepsilon}\mathrm{Area}(I) + 1 \leq (1+2\varepsilon)\,\mathrm{OPT}(I, C) + 1\ .\end{aligned}$$

Since $\sum_{c_l \in \bar{C}} c_l z_l + c_M z_M = \sum_{c_l \in C^{(1)}} c_l z_l$ is the objective value of our solution, this shows the theorem. (The proof is only a slight modification of the corresponding proof by Murgolo [69, Proof of Thm. 1, case 2].) □

2.6.4 Analysis of the Running Time

Deriving the running time is a rather straightforward calculation.

In Step 1 of the algorithm, I_{small} and I_{large} can be constructed in time $\mathcal{O}(n)$. Furthermore, it is mentioned in [72, p. 298] that $I^{(1)}$ and I_{huge} can be obtained in time $\mathcal{O}(\log(d_1)n) = \mathcal{O}(\log(\frac{1}{\varepsilon})n)$ even when I_{large} is not sorted. For the reduction of the bin sizes, i.e. the construction of $C^{(1)}$, we have the following lemma.

Lemma 2.25. *The bins $C^{(1)}$ can be constructed and sorted in time $\mathcal{O}(M + \frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})$.*

Proof. The set $\tilde{C}$ can be found in time $\mathcal{O}(M)$. If $M \leq \lfloor \frac{2}{\varepsilon}\ln\frac{1}{\varepsilon}\rfloor + 1$, we have $\tilde{C} = C^{(1)}$ and can sort the items in $\mathcal{O}(M\log M) = \mathcal{O}\left(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon}\right)$. If $M > \lfloor \frac{2}{\varepsilon}\ln\frac{1}{\varepsilon}\rfloor + 1$, Lawler [63] has presented a technique to construct $C^{(1)}$: create a bucket for every interval $\tilde{C}_l$ and place every bin size $c \in \tilde{C}$ in its corresponding bucket. The largest bin size in every $\tilde{C}_l$ can then be found by a single scan of the corresponding bucket. Here, the right bucket for one c, i.e. the right exponent l, can be determined in $\mathcal{O}(1)$ because we assume that the logarithm can be calculated in $\mathcal{O}(1)$. Thus, adding all $c \in \tilde{C}$ to the right buckets can be done in $\mathcal{O}(M)$. The creation of the buckets themselves needs time in $\mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ and finding the largest c in every $\tilde{C}_l$ in total $\mathcal{O}(M)$. Note that the buckets can of course be created in increasing order of $l \in \{0, 1, \ldots\}$ such that $C^{(1)}$ is sorted in the end. □

The running time of Step 2 is already known (see Theorem 2.21 and (2.16)). It is not difficult to see that Steps 3 and 5 need time $\mathcal{O}(n)$. Finally, Step 4 consists of grouping the items in I_{huge} and then packing them with Next-Fit, which can be done in $\mathcal{O}(\log(\frac{1}{\delta})n) = \mathcal{O}(\log(\frac{1}{\varepsilon})n)$. (In fact, the items have normally already been grouped during the construction of I_{huge}.) As $C^{(1)}$ contains $M_1 = \min\{M, \lfloor \frac{2}{\varepsilon}\ln\frac{1}{\varepsilon}\rfloor + 1\}$ different bin sizes, we get the overall running time:

$$\begin{aligned}\mathcal{O}\Bigg(& \max\left\{UKPIP\left(d_1, M_1, c_{\min}, \frac{\bar{\varepsilon}}{6}\right), \mathcal{O}\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\varepsilon}\right)\log\log\left(\frac{1}{\varepsilon}\right)\right)\right\} \\ & \cdot \frac{1}{\varepsilon}\log\left(\frac{1}{\varepsilon}\right)\cdot\left[\log\left(\frac{1}{\varepsilon}\right) + \frac{1}{\varepsilon^2}\right] + \frac{1}{\varepsilon^{2.5356}}\cdot\log^{2.5356}\left(\frac{1}{\varepsilon}\right)\left[\log\left(\frac{1}{\varepsilon}\right) + \frac{1}{\varepsilon^2}\right] \\ & + \frac{1}{\varepsilon^3}\log\left(\frac{1}{\varepsilon}\right) + M + \log\left(\frac{1}{\varepsilon}\right)n\Bigg).\end{aligned} \tag{2.21}$$

Assume that $UKPIP(d_1, M_1, c_{\min}, \frac{\bar{\varepsilon}}{6}) \in \Omega(\frac{1}{\varepsilon^2})$, which is motivated by the running time of known UKPIP algorithms (as will be seen in this thesis). By simplifying the expression above, we see that the AFPTAS needs time in

$$\mathcal{O}\left(UKPIP\left(d_1, M_1, c_{\min}, \frac{\bar{\varepsilon}}{6}\right)\cdot\frac{1}{\varepsilon^3}\log\frac{1}{\varepsilon} + M + \log\left(\frac{1}{\varepsilon}\right)n\right) \quad .$$

This shows the Theorems 2.4 and 2.5. The time complexity for BP is obtained from (2.21) with an FPTAS for UKP instead of one for UKPIP and with $M = 1$.

The last missing part of the algorithm is an FPTAS for UKPIP and UKP. One possibility mentioned in the introduction is to employ Lawler's FPTAS for UKP [63]. For VBP, we get an FPTAS for UKPIP in the following way: we separately solve for every $c_l \in C^{(1)}$ the UKP instance with profits $\frac{p_j}{c_l}$ and return the maximum profit over all c_l. The correctness of this FPTAS for UKPIP follows from the lemmas in Section 3.4.

Since Lawler's FPTAS has a running time in $\mathcal{O}(d_1 + \frac{1}{\varepsilon^3})$ for one knapsack size, this method would yield a running time $UKPIP(d_1, M_1, c_{\min}, \frac{\bar{\varepsilon}}{6}) \in \mathcal{O}(M_1 \cdot (d_1 + \frac{1}{\varepsilon^3})) = \mathcal{O}(\frac{1}{\varepsilon^4} \log \frac{1}{\varepsilon})$ and therefore $\mathcal{O}(\frac{1}{\varepsilon^7} \log^2(\frac{1}{\varepsilon}) + M + \log(\frac{1}{\varepsilon})n)$ for the entire AFPTAS. Note that $\bar{\varepsilon} = \frac{\varepsilon}{4}$ as stated in Theorem 2.13 and that $c_{\min} \geq \varepsilon$ because $c_{\min} \in C^{(1)} \subseteq \tilde{C}$.

Theorem 2.26. *$A_\varepsilon(I, C)$ has for VBP a running time in $\mathcal{O}(\frac{1}{\varepsilon^7} \log^2(\frac{1}{\varepsilon}) + M + \log(\frac{1}{\varepsilon})n)$.*

This is indeed the running time our VBP algorithm originally had in [42]. Similarly, the FPTAS for UKP by Lawler [63] yields the following time complexity for Bin Packing:

Theorem 2.27. *$A_\varepsilon(I, C)$ is for $C = \{1\}$ an AFPTAS for BP with a running time in $\mathcal{O}(\frac{1}{\varepsilon^6} \log(\frac{1}{\varepsilon}) + \log(\frac{1}{\varepsilon})n)$.*

Remark 2.28. Note that we can even set $\delta = \mathcal{O}(\varepsilon)$ for BP as done e.g. by Karmarkar and Karp [56]. Moreover, the approximation ratio of $(1 + \varepsilon)$ for BP and VBP is achieved by replacing ε by $\frac{\varepsilon}{17}$, which does not change the asymptotic running time.

The running times of our AFPTAS for VBP and BP depend on the running time of the FPTAS for UKPIP or UKP. The next chapters will show how we can improve them and therefore the time complexity of the AFPTAS.

3 The Knapsack Problem with Inversely Proportional Profits

3.1 Introduction

This chapter presents algorithms for variants of the Knapsack Problem with Inversely Proportional Profits (KPIP), namely for the 0-1 variant (0-1 KPIP), the unbounded variant (UKPIP), and the bounded variant (BKPIP). They are formally defined in Subsection 1.3.3.

3.1.1 Known Results

The 0-1 Knapsack Problem and the other variants of KP like the Unbounded Knapsack Problem (UKP) and the Bounded Knapsack Problem (BKP) are well-known NP-hard problems [25]. They can be optimally solved in pseudo-polynomial time by dynamic programming [5, 61]. As KP is a maximization problem, the absolute ratio of any algorithm A for KP is defined by $\inf_I \frac{A(I)}{\mathrm{OPT}(I)}$, and it is bounded from above by 1. While Bin Packing has a bound of $\frac{3}{2}$ on the absolute approximation ratio for polynomial-time algorithms (see Subsection 2.1.1), there is no upper bound on the absolute approximation ratio of KP: there are several known PTAS and FPTAS.

The first FPTAS for 0-1 KP was presented by Ibarra and Kim [39] with a running time in $\mathcal{O}(n \log n + \frac{1}{\varepsilon^2} \cdot \min\{\frac{1}{\varepsilon^2} \log(\frac{1}{\varepsilon}), n\})$ and a space complexity in $\mathcal{O}(n + \frac{1}{\varepsilon^3})$. Lawler [63] improved the running time to $\mathcal{O}(\frac{1}{\varepsilon^4} + \log(\frac{1}{\varepsilon})n)$. In 1981, Magazine and Oguz [67] presented a method to decrease the space complexity of the dynamic program so that their FPTAS runs in time $\mathcal{O}(n^2 \log(n) \frac{1}{\varepsilon^2})$ and needs space in $\mathcal{O}(\frac{n}{\varepsilon})$. (The paper focuses on the improved space complexity without a partitioning and reduction of the items as done e.g. by Lawler. Without it, Lawler's basic algorithm has in fact a time and space complexity in $\mathcal{O}(\frac{n^2}{\varepsilon})$.) The currently fastest known algorithm is due to Kellerer and Pferschy [59, 60, 61, pp. 166–183] with a space bound in $\mathcal{O}(n + \frac{1}{\varepsilon^2})$ and a running time in $\mathcal{O}(n \min\{\log n, \log \frac{1}{\varepsilon}\} + \frac{1}{\varepsilon^2} \log(\frac{1}{\varepsilon}) \cdot \min\{n, \frac{1}{\varepsilon} \log(\frac{1}{\varepsilon})\})$. Assuming that $n \in \Omega(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$, this is in $\mathcal{O}(n \log(\frac{1}{\varepsilon}) + \frac{1}{\varepsilon^3} \log^2(\frac{1}{\varepsilon}))$.

For UKP, Ibarra and Kim [39] presented the first FPTAS by extending their 0-1 KP algorithm. This UKP algorithm has a running time in $\mathcal{O}(n + \frac{1}{\varepsilon^4} \log \frac{1}{\varepsilon})$ and a space complexity in $\mathcal{O}(n + \frac{1}{\varepsilon^3})$. Kellerer et al. [61, pp. 232–234] have moreover described an

FPTAS with a time complexity in $\mathcal{O}(n\log(n) + \frac{1}{\varepsilon^2}(n + \log\frac{1}{\varepsilon}))$ and a space bound in $\mathcal{O}(n + \frac{1}{\varepsilon^2})$. In 1979, Lawler [63] presented his FPTAS with a running time in $\mathcal{O}(n + \frac{1}{\varepsilon^3})$ and a space complexity in $\mathcal{O}(n + \frac{1}{\varepsilon^2})$. For $n \in \Omega(\frac{1}{\varepsilon})$, this is still the best known FPTAS.

For BKP, a basic FPTAS was presented by Kellerer et al. in their book [61, pp. 207–209] and a more sophisticated by Plotkin et al. [72, pp. 295–297], where the latter is derived from Lawler's 0-1 KP algorithm [63] and runs in $\mathcal{O}(\min\{n\frac{1}{\varepsilon^2}, n\log(\frac{1}{\varepsilon}) + \frac{1}{\varepsilon^4}\})$.

For further information on the Knapsack Problem, we refer to the books by Martello and Toth [68] as well as by Kellerer, Pferschy, and Pisinger [61].

3.1.2 Our Results

As mentioned in Chapter 2, the AFPTAS for VBP has to solve UKPIP instances as a subroutine (see e.g. Subsection 2.5.1). This motivated us to formally define the Knapsack Problem with Inversely Proportional Profits (KPIP) as a new problem and to study it.

Subsection 2.6.4 has already presented a simple way to solve an instance of 0-1 KPIP, BKPIP or UKPIP with n items and M knapsack sizes: take an FPTAS for the corresponding variant of the knapsack problem. For each knapsack c_l, solve the KP instance with item profits $\frac{p_j}{c_l}$. Return the solution and knapsack size of highest profit. Lawler's algorithm [63] for UKP yields a running time in $\mathcal{O}(M \cdot (n + \frac{1}{\varepsilon^3}))$ and a space bound in $\mathcal{O}(M + n + \frac{1}{\varepsilon^2})$ for UKPIP. (The correctness of this approach is formally shown in Section 3.4.)

We change and improve the techniques of the KPIP approximation algorithm above such that several knapsack sizes can be processed at once. Thus, we avoid redundancy and get the following theorem.

Theorem 3.1. *Let $\varepsilon > 0$. Let $c_{\min}$ be the smallest knapsack size of a UKPIP instance with M knapsacks and n item types. There is an FPTAS that solves this problem in time*

$$\mathcal{O}\left(n\log M + \min\left\{\left\lfloor\log\frac{1}{c_{\min}}\right\rfloor + 1, M\right\}n + \min\left\{\left\lfloor\log\frac{1}{c_{\min}}\right\rfloor + 1, M\right\}\frac{1}{\varepsilon^3} + \frac{M}{\varepsilon^2}\right)$$

and space $\mathcal{O}(M + n + \frac{1}{\varepsilon^2})$.

The running time of the column generation subroutine for our VBP AFPTAS improves so that the overall time complexity decreases by a factor of $\Theta\left(\frac{1}{\varepsilon}\right)$:

Theorem 3.2. *The AFPTAS for VBP only needs time in $\mathcal{O}(\frac{1}{\varepsilon^6}\log^2(\frac{1}{\varepsilon}) + M + \log(\frac{1}{\varepsilon})n)$ (see Theorem 2.4).*

The FPTAS for KPIP together with the improved running time of our AFPTAS for VBP were presented at the conference CSR 2013 [43].

3.2 Overview

We start with some basic definitions and remarks in Section 3.3. Important are the definition of $\mathrm{OPT}_{c_l}(I)$ (best solution for the KPIP instance with the knapsack size c_l and scaled profits $\frac{p_j}{c_l}$) and $\overline{\mathrm{OPT}}_{c_l}(I)$ (optimum solution for the normal KP instance with the knapsack size c_l and non-scaled basic profits p_j). This notation is used throughout this chapter where values without bar (like $\mathrm{OPT}_{c_l}(I)$) are obtained with scaled profits $\frac{p_j}{c_l}$ and values with bar (like $\overline{\mathrm{OPT}}_{c_l}(I)$) with the basic profits p_j.

Section 3.4 shows an important property: for a solution of KPIP, it is sufficient to solve the normal KP problem with basic profits p_j for every knapsack size c_l. By scaling every solution by the corresponding c_l, we get the solutions P_{c_l}. The maximum $\max_{c_l} P_{c_l}$ is the solution for the KPIP instance. This is also true for approximate solutions. The property allows us to work with the basic profits p_j: we could apply e.g. the UKP algorithm by Lawler [63] to every knapsack size c_l and basic profits p_j and scale the obtained profits afterwards. However, this results in using dynamic programming M times, which is time-consuming.

In Section 3.5, the basic FPTAS for 0-1 KPIP is introduced. Since it is an extension of Lawler's FPTAS [63], our presentation follows the one by Lawler. First, we introduce the well-known Dynamic Programming by Profits in Subsection 3.5.1, where $F_j(i)$ stands for the smallest total size to get profit i if only the items $a_1, \ldots, a_j$ are considered.

Subsection 3.5.2 presents the extended greedy approach to find first approximations $\bar{P}_{c_l} \geq \frac{1}{2}\overline{\mathrm{OPT}}_{c_l}(I)$ for every knapsack c_l. Because of the scaling property of Section 3.4, we get the approximations $P_{c_l} = \frac{\bar{P}_{c_l}}{c_l} \geq \frac{1}{2}\mathrm{OPT}_{c_l}(I)$ and $P_0 = \max_{l\in\{1,\ldots,M\}} P_{c_l} \geq \frac{1}{2}\mathrm{OPT}(I)$. The special item sets S_l are introduced, which are the items with a size $c_{l-1} < s(a) \leq c_l$, and the derived sets $\bar{S}_l = \bigcup_{l'=1}^{l} S_{l'}$, which are the items that fit into c_l, i.e. $s(a) \leq c_l$.

Subsection 3.5.3 explains the basic FPTAS that avoids redundancy for large values of M by determining the values $F_j(i)$ for several knapsack sizes at once. At the beginning, a threshold T based on the values $\bar{P}_{c_l}$ is defined, and the items are partitioned into large ones of profit $p_j > T$ and small ones of profit $p_j \leq T$. Let n_L be the number of large items. We assume without loss of generality that the first n_L items $a_1, \ldots, a_{n_L}$ are the large ones. Their profits are scaled to values $q_j = q(a_j)$ with a scaling constant K based on T. Dynamic Programming by Profits is applied to the scaled profits and yields candidates $(i, F_{n_L}(i))$ where every candidate represents a set of large items of scaled total profit i and total size $F_{n_L}(i)$. Then, we determine an approximate solution for every c_l by $\bar{P}_l^{(1)} = \max_{F_{n_L}(i)\leq c_l} K \cdot i + \phi_l(c_l - F_{n_L}(i))$. The value $K \cdot i$ is the (almost) unscaled profit of the corresponding set of large items, and $\phi_l(c_l - F_{n_L}(i))$ is the profit obtained by greedily adding the small items to the remaining knapsack capacity

$c_l - F_{n_L}(i)$. The definition of $\bar{S}_l$ is useful for these calculations. We set $P_l^{(1)} = \bar{P}_l^{(1)}/c_l$ and obtain with $P_1 = \max_l P_l^{(1)}$ an approximation to the optimum of our instance by the observation in Section 3.4.

Unfortunately, the overall running time depends on $\frac{c_M}{c_1}$, i.e. the ratio of the largest knapsack $c_M = c_{\max}$ to the smallest knapsack $c_1 = c_{\min}$. It is improved by first partitioning the knapsacks into intervals $C_b = \{c_l \in C \mid c_l \in (c_{\min}^{(b+1)w}, c_{\min}^{bw}]\}$ for $b \in \{0, \ldots, \lfloor \frac{1}{w} \rfloor + 1\}$ and a suitable $w \leq 1$ and then by separately executing the algorithm above for every C_b. Hence, a threshold T_b and a scaling constant K_b are introduced for every C_b. Now, the ratio $c_{\max}^{(b)}/c_{\min}^{(b)}$ of the largest to the smallest knapsack in C_b is bounded by $\frac{1}{c_{\min}^w}$. Thus, w is chosen in such a way that $\mathcal{O}(\frac{1}{c_{\min}^w} \cdot \frac{1}{w})$, i.e. the running time of one dynamic programming computation times the number of computations, is minimized.

Subsection 3.5.4 shows how the running time and storage space of the algorithm can be further reduced. In fact, only a limited number of large items with profits in $L^{(k,b)} = (2^k T_b, 2^{k+1} T_b]$ can be part of a solution for knapsacks in C_b (otherwise, the upper bound $\overline{\mathrm{OPT}}_{c_l}(I) \leq 2\bar{P}_{c_l}$ would be exceeded). Hence, we take a reduced number $n_{L,j}$ of items for every scaled profit q_j, which is sufficient for an approximate solution. Therefore, the total number of large items $n_b \leq n_L$ is bounded. (n_L now denotes the global upper bound on all n_b.) Furthermore, we redefine $F_j(i)$ to be the smallest total size to obtain the (scaled) total profit i if only the items with the first j scaled profits $q_1, \ldots, q_j$ are considered. By adapting the dynamic program accordingly, the total storage needed for the algorithms decreases, too.

The variants UKPIP and BKPIP are addressed in Section 3.6. Of special interest is the adaptation of our algorithm to UKPIP because it is the subproblem we have to solve for our VBP AFPTAS. For UKPIP, the instance is transformed (in parts) into a 0-1 KPIP instance. First, the item $\tilde{a}_j$ of smallest size is determined for every scaled profit q_j: these items are obviously sufficient to get an approximate solution for UKPIP. Second, items $\tilde{a}_j^{(r)}$ of profit $2^r p(\tilde{a}_j)$ and size $2^r s(\tilde{a}_j)$ are created, which can represent any choice of large items in a feasible solution of UKPIP. Only one of these items has to be kept for every scaled profit q_j, which (again) improves the running time. The space needed for the algorithm can be further decreased by adapting the dynamic program. Even the greedy algorithm to determine the values of $\phi_l(\cdot)$ is faster because we only have to take copies of the most efficient small item $a_{\mathrm{eff}}^{(l)}$ for c_l. Such improvements are not possible for BKPIP, but the basic idea to create items as above is also used.

3.3 Notation and Remarks

We introduce some useful notation. The basic profit of an item a is denoted by $p(a)$ and its size by $s(a)$. If $a = a_j$, we also write $p(a_j) = p_j$ and $s(a_j) = s_j$. A multi-set $V = \{x_a : a \mid a \in I\}$ of items is a subset of items in I with the item multiplicities. We assume that $x_a \in D_a$: note that $D_a = \{0, 1\}$ for 0-1 KPIP, $D_a = \{0, \ldots, d_a\}$ for BKPIP (with $d_a \in \mathbb{N}$), and $D_a = \mathbb{N}$ for UKPIP (see Subsection 1.3.3). We naturally define the total basic profit $p(V) := \sum_{x_a>0} p(a)x_a$ and the total size $s(V) := \sum_{x_a>0} s(a)x_a$.

$$\mathrm{OPT}_{c_l}(I) = \max\left\{\sum_{a\in I}\frac{p(a)}{c_l}x_a \;\middle|\; \sum_{a\in I} s(a)x_a \le c_l;\ x_a \in D_a \text{ for } a \in I\right\}$$

refers to the optimal solution for the current variant of KPIP when only knapsack size c_l is considered. $\mathrm{OPT}(I) = \max_{l\in\{1,\ldots,M\}} \mathrm{OPT}_{c_l}(I)$ denotes the global optimum.

We will also consider the case where the items only have their unscaled basic profit p_j, independent of the knapsack in which they are packed. Let $v \le c_M = 1$ be a part of any knapsack. The corresponding optimal basic profit for the volume v is denoted by

$$\overline{\mathrm{OPT}}\,(I, v) = \max\left\{\sum_{a\in I} p(a)x_a \;\middle|\; \sum_{a\in I} s(a)x_a \le v;\ x_a \in D_a \text{ for } a \in I\right\} \quad .$$

For $v = c_l$, we also write the optimum for knapsack size c_l as $\overline{\mathrm{OPT}}_{c_l}(I) = \overline{\mathrm{OPT}}(I, c_l)$. Let $\overline{\mathrm{OPT}}(I) = \max_{l\in\{1,\ldots,M\}} \overline{\mathrm{OPT}}_{c_l}(I)$ be the global optimum. (Note that $\overline{\mathrm{OPT}}(I) = \overline{\mathrm{OPT}}_{c_M}(I)$.) This notation will also be used in Chapters 4 and 5. As there is only one knapsack size in the case of UKP, Chapter 4 will use expressions without bar (e.g. $\mathrm{OPT}(I)$ instead of $\overline{\mathrm{OPT}}(I)$) for ease of notation.

Remark 3.3. For $I = \{a_1, \ldots, a_n\}$, we can of course write the optimum as $\mathrm{OPT}_{c_l}(I) = \max\{\sum_{j=1}^{n} \frac{p_j}{c_l}x_j \mid \sum_{j=1}^{n} s_j x_j \le c_l; x_j \in D_j\}$ etc. (a notation that will indeed be employed). The UKP algorithm in Chapter 4 and the UKPIP algorithm in Chapter 5 will however often use the optimum for different item sets in the analysis of the respective algorithms. Hence, we have chosen the more general notation of summing over all items in the instance $\sum_{a\in I}$ instead of the summation $\sum_{j=1}^{n}$.

As mentioned in Section 1.4, we assume that basic arithmetic operations as well as computing the logarithm can be done in $\mathcal{O}(1)$.

3.4 Observations

Two simple observations help us to handle that the profits depend on the knapsack size. In fact, they allow us to work only with the basic profits p_j.

Lemma 3.4. *For every variant of KPIP, it is sufficient to calculate for every knapsack c_l a $(1-\varepsilon)$ approximate solution $P_{c_l} \geq (1-\varepsilon)\mathrm{OPT}_{c_l}(I)$ with profits $\frac{p_j}{c_l}$ and take as final solution the maximum over all knapsack sizes* $\max_l P_{c_l}$.

Proof. Let $I = \{a_1, \ldots, a_n\}$. Determine for every knapsack c_l a $(1-\varepsilon)$ approximate solution $(x_1^{(l)}, \ldots, x_n^{(l)})$ such that

$$P_{c_l} = \sum_{j=1}^{n} \frac{p_j}{c_l} x_j^{(l)} \geq (1-\varepsilon)\mathrm{OPT}_{c_l}(I) \ . \tag{3.1}$$

Let $c_r \in \{c_1, \ldots, c_M\}$ be the knapsack size and $(x_1^{(r)}, \ldots, x_n^{(r)})$ be the $(1-\varepsilon)$ solution with the maximum profit over all knapsack sizes. Therefore, we have

$$\sum_{j=1}^{n} \frac{p_j}{c_r} x_j^{(r)} = \max_{l \in \{1,\ldots,M\}} \sum_{j=1}^{n} \frac{p_j}{c_l} x_j^{(l)} \ . \tag{3.2}$$

On the other hand, let $c_q \in \{c_1, \ldots, c_M\}$ be the knapsack size with $\mathrm{OPT}(I) = \mathrm{OPT}_{c_q}(I)$. Then obviously

$$\sum_{j=1}^{n} \frac{p_j}{c_r} x_j^{(r)} \overset{(3.2)}{\geq} \sum_{j=1}^{n} \frac{p_j}{c_q} x_j^{(q)} \overset{(3.1)}{\geq} (1-\varepsilon)\,\mathrm{OPT}(I) \ ,$$

and $(x_1^{(r)}, \ldots, x_n^{(r)})$ is a $(1-\varepsilon)$ approximate solution. □

Lemma 3.5. *Let c_l be a knapsack size. A $(1-\varepsilon)$ approximate solution for one variant of KP with the knapsack size c_l and the basic profits p_j is also a $(1-\varepsilon)$ solution for the corresponding variant of KPIP with profits $\frac{p_j}{c_l}$ and (only) the knapsack size c_l, and vice versa.*

Moreover, we have

$$\mathrm{OPT}_{c_l}(I) = \frac{\overline{\mathrm{OPT}}_{c_l}(I)}{c_l} \ . \tag{3.3}$$

Proof. Let (again) $I = \{a_1, \ldots, a_n\}$. First, every solution to the problem with basic profits p_j is a feasible solution to the problem with profits $\frac{p_j}{c_l}$ because only the item profits, but not the item sizes, change.

Let $(x_1, \ldots, x_n)$ be a $(1-\varepsilon)$ approximate solution and $(\tilde{x}_1, \ldots, \tilde{x}_n)$ be an optimal solution for knapsack size c_l and basic profits p_j. By definition, we have

$$\sum_{j=1}^{n} p_j x_j \geq (1-\varepsilon)\overline{\mathrm{OPT}}_{c_l}(I) = (1-\varepsilon) \sum_{j=1}^{n} p_j \tilde{x}_j \ . \tag{3.4}$$

On the other hand, we show that

$$\frac{1}{c_l} \sum_{j=1}^{n} p_j \tilde{x}_j = \mathrm{OPT}_{c_l}(I) \ . \tag{3.5}$$

The inequality $\sum_{j=1}^{n} \frac{p_j}{c_l} \tilde{x}_j \leq \mathrm{OPT}_{c_l}(I)$ is obvious. To show the inequality $\sum_{j=1}^{n} \frac{p_j}{c_l} \tilde{x}_j \geq \mathrm{OPT}_{c_l}(I)$, let $(\bar{x}_1, \ldots, \bar{x}_n)$ be an optimal solution for c_l with profits $\frac{p_j}{c_l}$. Then

$$\sum_{j=1}^{n} \frac{p_j}{c_l} \bar{x}_j = \mathrm{OPT}_{c_l}(I) \ ,$$

and

$$\sum_{j=1}^{n} p_j \bar{x}_j \leq \overline{\mathrm{OPT}}_{c_l}(I) = \sum_{j=1}^{n} p_j \tilde{x}_j \tag{3.6}$$

because $(\bar{x}_1, \ldots, \bar{x}_n)$ is also a feasible solution for the considered variant of KP with the knapsack size c_l. Dividing by $c_l > 0$ on both sides of (3.6) shows that $\mathrm{OPT}_{c_l}(I) = \sum_{j=1}^{n} \frac{p_j}{c_l} \bar{x}_j \leq \sum_{j=1}^{n} \frac{p_j}{c_l} \tilde{x}_j$ and therefore (3.5).

We get

$$\sum_{j=1}^{n} p_j x_j \overset{(3.4)}{\geq} (1-\varepsilon) c_l \sum_{j=1}^{n} \frac{p_j}{c_l} \tilde{x}_j \overset{(3.5)}{=} (1-\varepsilon) c_l \mathrm{OPT}_{c_l} \ \Rightarrow \ \sum_{j=1}^{n} \frac{p_j}{c_l} x_j \geq (1-\varepsilon) \mathrm{OPT}_{c_l}(I) \ .$$

The proof that an approximate solution for the profits $\frac{p_j}{c_l}$ is also an approximate solution for the basic profits p_j is almost the same.

Note that we have also shown that $\mathrm{OPT}_{c_l}(I) = \frac{\overline{\mathrm{OPT}}_{c_l}(I)}{c_l}$. □

We get the following result:

Algorithm 3.1: `MaxSolution`

Input: Item set I, knapsack sizes $c_1 < \ldots < c_M = 1$

Output: Solution of value $P \geq (1-\varepsilon)\mathrm{OPT}(I)$

1 **for** *all* c_l, $l \in \{1, \ldots, M\}$ **do**

2 Find a solution $(x_1^{(l)}, \ldots, x_n^{(l)})$ with $\sum_{j=1}^{n} p_j x_j^{(l)} \geq (1-\varepsilon)\, \overline{\mathrm{OPT}}(I, c_l)$;

3 **for** *all* c_l, $l \in \{1, \ldots, M\}$ **do**

4 Compute $P_{c_l} := \frac{1}{c_l} \sum_{j=1}^{n} p_j x_j^{(l)}$;

5 **return** $\max_{l \in \{1,\ldots,M\}} P_{c_l}$;

Theorem 3.6. *The algorithm* `MaxSolution` *presented in Algorithm 3.1 finds a* $(1-\varepsilon)$ *approximate solution for all considered KPIP variants.*

Proof. Lemma 3.5 shows that $P_{c_l} \geq (1-\varepsilon)\mathrm{OPT}_{c_l}(I)$. Taking the maximum over all P_{c_l} then yields a solution with an objective value of at least $(1-\varepsilon)\mathrm{OPT}(I)$ according to Lemma 3.4. □

3.5 Extending Lawler's Algorithm

We have seen that `MaxSolution` calculates the desired approximate solution. In this section, we will see how to extend Lawler's algorithm for 0-1 KP [63] to 0-1 KPIP such that we obtain the approximate solutions in Step 2 of the algorithm, and how we optimize the overall running time when calculating them. BKPIP and UKPIP will then be considered in Section 3.6. Our presentation follows the one by Lawler.

First, we will introduce basic techniques for the Knapsack Problem, which are designed for integral profits p_j, sizes s_j, and knapsack sizes c_l. Hence, we first assume that $p_j, s_j, c_l \in \mathbb{N}$ for all j and l. Then, we will see that the techniques and the FPTAS also work for $p_j, s_j, c_l \in (0, 1]$.

For ease of notation, this section assumes that $I = \{a_1, \ldots, a_n\}$.

3.5.1 Dynamic Programming

Dynamic Programming by Profits [5, 61] is a well-known technique to optimally solve a 0-1 KP instance in pseudo-polynomial time. Let $F_j(i) \in \mathbb{R}_{\geq 0}$ be the minimum size necessary to obtain the profit $i \in \mathbb{N}$ with the items $a_1, \ldots, a_j$. For instance, $F_1(0) = 0$ and $F_1(p_1) = s_1$ hold. If a profit i cannot be obtained with the first j items, $F_j(i) = \infty$ is set, i.e. $F_1(i) = \infty$ for $i \neq 0, p_1$ and $F_j(i) = \infty$ for $i < 0$ and all j. The $F_j(i)$ are computed by the Bellman recursion

$$F_j(i) = \min\{F_{j-1}(i), F_{j-1}(i - p_j) + s_j\} \ .$$

Let $c_1 < \ldots < c_M$ be several knapsack sizes. We obviously have the identity $\overline{\mathrm{OPT}}_{c_l}(I) = \max\{i \mid F_n(i) \leq c_l\}$ (see also [29]), and the corresponding items can be found by backtracking in the table of the $F_j(i)$. Note that it is possible that $i \leq i'$, but $F_n(i) \geq F_n(i')$: a larger profit i' is achieved with a smaller size $F_n(i')$. Then, we say like in [63] that $F_n(i)$ is *dominated* by $F_n(i')$. It is not difficult to remove dominated table entries $F_n(i)$.

Lemma 3.7. *Dominated table entries $F_n(i)$ can be discarded in $\mathcal{O}(\overline{PB})$, where $\overline{PB}$ is an upper bound on the maximum profit* $\max_{l \in \{1,\ldots,M\}} \overline{\mathrm{OPT}}_{c_l}(I) = \overline{\mathrm{OPT}}_{c_M}(I)$.

Proof. Let $F_n(1), \ldots, F_n(\overline{PB})$ be the table entries: they are already sorted in increasing order of their profit i. Moreover, the profits i are already different so that $F_n(i')$ can only dominate $F_n(i)$ if $i < i'$, but $F_n(i) \geq F_n(i')$. We define a partial ordering on the pairs $(i, F_n(i))$ where $(i, F_n(i)) < (i', F_n(i'))$ if $i < i'$ and $F_n(i) < F_n(i')$, i.e. $(i, F_n(i))$ has a smaller profit than $(i', F_n(i'))$ and $F_n(i)$ is not dominated by $F_n(i')$. We call a sequence of pairs $(i_1, F_n(i_1)), \ldots, (i_r, F_n(i_r))$ sorted if $(i_1, F_n(i_1)) < \ldots < (i_r, F_n(i_r))$. Hence, no $F_n(i)$ is dominated in a sorted sequence.

Obviously, the last sequence element $(\overline{PB}, F_n(\overline{PB}))$ cannot be dominated. We iteratively construct a subsequence of non-dominated pairs in the following way: let $(i, F_n(i)), \ldots, (\overline{PB}, F_n(\overline{PB}))$ be the current sequence such that $(i, F_n(i)) < \ldots < (\overline{PB}, F_n(\overline{PB}))$. Compare $(i-1, F_n(i-1))$ and $(i, F_n(i))$. If $(i-1, F_n(i-1)) < (i, F_n(i))$, add $(i-1, F_n(i-1))$ to the sequence as the new first element, otherwise discard it because $F_n(i-1) \geq F_n(i)$ holds. Continue until all items of the original sequence have been processed, i.e. until $(1, F_n(1))$ has been considered.

It is clear that the resulting sequence is sorted and therefore consists only of pairs that correspond to non-dominated $F_n(i)$. On the other hand, we have only removed pairs from the original sequence that are dominated. Therefore, the new sequence has to be the original sequence with the dominated entries removed. The running time of $\mathcal{O}(\overline{PB})$ is obvious. Note that we have implicitly removed entries $F_n(i) = \infty$: there is no item set with the total profit i if this is the case. □

Thus, all $\overline{\mathrm{OPT}}_{c_l}(I)$ can be found by a single scan of the $F_n(i)$ in time $\mathcal{O}(\overline{PB} + M)$.

Theorem 3.8. *The $F_j(i)$ can be determined in time and space $\mathcal{O}(\overline{PB} \cdot n)$. Afterwards, the optimal values $\overline{\mathrm{OPT}}_{c_l}(I)$ for $l \in \{1, \ldots, M\}$ can be found in time $\mathcal{O}(\overline{PB} + M)$ and space $\mathcal{O}(M)$. Dynamic programming also works for non-integral item sizes $s_j \in \mathbb{R}_{>0}$ and knapsack sizes $c_l \in \mathbb{R}_{>0}$.*

Proof. The $F_j(i)$ form a table with $n \cdot \overline{PB}$ entries. As one entry $F_j(i)$ can be determined in $\mathcal{O}(1)$, the overall running time to determine the $F_j(i)$ is therefore in $\mathcal{O}(n \cdot \overline{PB})$. Since $\overline{\mathrm{OPT}}_{c_l}(I) = \max\{i \mid F_n(i) \leq c_l\}$ holds as mentioned above, it is sufficient to calculate all entries $F_n(i)$ once and discard the dominated $F_n(i)$. Thus, all $\overline{\mathrm{OPT}}_{c_l}(I)$ for $l \in \{1, \ldots, M\}$ can be determined in additional time $\mathcal{O}(\overline{PB} + M)$ by one scan of the $F_n(i)$.

The overall space requirement can be bounded by $\mathcal{O}(n \cdot \overline{PB})$ to calculate and save the $F_j(i)$ and $\mathcal{O}(M)$ to save the $\overline{\mathrm{OPT}}_{c_l}(I)$.

Finally, dynamic programming does not use the fact that the item sizes s_j and knapsack sizes c_l are integral: the algorithm only assumes that the profits p_j are integral whereas it is sufficient that the sizes s_j, c_l are positive.

A trivial bound on $\overline{PB}$ (not used in this thesis) is $n \cdot p_{\max}$, where $p_{\max}$ is the largest item profit. □

3.5.2 Bounds for the Optimum: a Simple $\frac{1}{2}$ Algorithm

We present a simple algorithm for a bound on $\overline{\mathrm{OPT}}_{c_l}(I)$. Let

$$S_1 := \{a_i \mid s(a_i) \leq c_1\} \text{ and } S_l := \{a_i \mid c_{l-1} < s(a_i) \leq c_l\} \text{ for } l \in \{2, \ldots, M\} \ .$$

Note that we still have $c_1 < c_2 < \ldots < c_M$.

Moreover, let $\bar{S}_l := \bigcup_{l'=1}^{l} S_{l'} = \{a \mid s(a) \leq c_l\}$, which is the set of the items that have to be considered for a solution of knapsack size c_l.

The basic idea of the approximation for any c_l is simple: sort the items in $\bar{S}_l$ according to their efficiency $\frac{p(a)}{s(a)}$; if ties occur, we can e.g. consider the item of smaller index to have a smaller efficiency. Start with the item of largest efficiency and add items in $\bar{S}_l$ to the knapsack in non-increasing order of the efficiency until no more items can be added. Return $\bar{P}_{c_l}$, the maximum of this value and of the most precious item $a_{\max}^{(l)}$ with profit $p_{\max}^{(l)} := \max\{p_j \mid s_j \leq c_l\}$. We will see below how this can be done without sorting the items. This greedy approximation algorithm is also called `Ext-Greedy` [61].

For 0-1 KPIP, let

$$P_{c_l} := \frac{\bar{P}_{c_l}}{c_l} \ . \tag{3.7}$$

Finally, let $P_0 := \max_{l \in \{1,\ldots,M\}} P_{c_l}$ be the global maximum.

Theorem 3.9. *For 0-1 KPIP, we have* $\bar{P}_{c_l} \geq \frac{1}{2}\overline{\mathrm{OPT}}_{c_l}(I)$ *and* $P_{c_l} \geq \frac{1}{2}\mathrm{OPT}_{c_l}(I)$. *Furthermore,* P_0 *is a* $\frac{1}{2}$ *approximation to the global optimum* $\mathrm{OPT}(I)$.

Proof. For ease of notation, let us assume without loss of generality that we have $\bar{S}_l = \{a_1, \ldots, a_n\}$ and $\frac{p_1}{s_1} \geq \frac{p_2}{s_2} \geq \ldots \geq \frac{p_n}{s_n}$, i.e. the items are already sorted according to their efficiency $\frac{p(a)}{s(a)}$ in non-increasing order.

Determine now j_{c_l} such that $s_1 + \cdots + s_{j_{c_l}} \leq c_l$, but $s_1 + \cdots + s_{j_{c_l}} + s_{j_{c_l}+1} > c_l$. Then we have

$$p_1 + \cdots + p_{j_{c_l}} \leq \overline{\mathrm{OPT}}_{c_l}(I) < p_1 + \cdots + p_{j_{c_l}} + p_{j_{c_l}+1} \ . \tag{3.8}$$

Since $p_{\max}^{(l)} := \max\{p_i \mid s_i \leq c_l\} \leq \overline{\mathrm{OPT}}_{c_l}(I)$, the inequality

$$\begin{aligned}
\overline{\mathrm{OPT}}_{c_l}(I) &< p_1 + \cdots + p_{j_{c_l}} + p_{j_{c_l}+1} \leq p_1 + \cdots + p_{j_{c_l}} + p_{\max}^{(l)} \\
&\leq 2\max\left\{p_1 + \cdots + p_{j_{c_l}}, p_{\max}^{(l)}\right\} \\
&\leq 2\overline{\mathrm{OPT}}_{c_l}(I)
\end{aligned}$$

holds, i.e.

$$\frac{1}{2}\overline{\mathrm{OPT}}_{c_l}(I) \leq \bar{P}_{c_l} = \max\left\{p_1 + \cdots + p_{j_{c_l}}, p_{\max}^{(l)}\right\} \leq \overline{\mathrm{OPT}}_{c_l}(I) \ . \tag{3.9}$$

Thus, $\bar{P}_{c_l}$ is a $1 - \varepsilon = 1 - \frac{1}{2} = \frac{1}{2}$ approximation to $\overline{\mathrm{OPT}}_{c_l}(I)$ so that $\bar{P}_{c_l}$ is a suitable lower bound on $\overline{\mathrm{OPT}}_{c_l}(I)$ and $2\bar{P}_{c_l}$ an upper bound.

$P_{c_l} = \frac{\bar{P}_{c_l}}{c_l}$ is a $\frac{1}{2}$ approximate solution for c_l with profits $\frac{p_j}{c_l}$ according to Lemma 3.5. Lemma 3.4 shows that $P_0 = \max_{l \in \{1,\ldots,M\}} P_{c_l}$ is also a $\frac{1}{2}$ approximation to $\mathrm{OPT}(I)$ with $\frac{1}{2}\mathrm{OPT}(I) \leq P_0 \leq \mathrm{OPT}(I)$.

(The main part of this proof is taken from [63].) □

Corollary 3.10. *For 0-1 KPIP, we can discard knapsacks c_l with $P_{c_l} = \frac{\bar{P}_{c_l}}{c_l} < \frac{1}{2}P_0$.*

Proof. To avoid confusion, let $P^{1/2}_{c_l} := P_{c_l}$ be the $\frac{1}{2}$ approximate value for knapsack size c_l found by the greedy approach of this subsection. Moreover, let $P^{\varepsilon}_{c_l}$ be a $(1-\varepsilon)$ approximation to $\mathrm{OPT}_{c_l}(I)$. We show that there is at least one knapsack size $c_l \in C$ such that $P^{1/2}_{c_l} \geq \frac{1}{2}P_0$ and $P^{\varepsilon}_{c_l} \geq (1-\varepsilon)\mathrm{OPT}(I)$. In fact, let c_q be the optimum knapsack size so that $\mathrm{OPT}(I) = \mathrm{OPT}_{c_q}(I)$. We get

$$P^{1/2}_{c_q} \overset{\text{Thm. 3.9}}{\geq} \frac{1}{2}\mathrm{OPT}_{c_q}(I) = \frac{1}{2}\mathrm{OPT}(I) \overset{\text{Thm. 3.9}}{\geq} \frac{1}{2}P_0 \ .$$

Moreover, $P^{\varepsilon}_{c_q} \geq (1-\varepsilon)\mathrm{OPT}_{c_q}(I) = (1-\varepsilon)\mathrm{OPT}(I)$ holds, which finishes the proof. □

We show how to actually construct the S_l and to find the $\bar{P}_{c_l}$ and P_0. First, the $\bar{P}_{c_l}$ can be constructed without having to sort the items according to their efficiency by a median-based divide-and-conquer strategy (similar to [63]).

Lemma 3.11. *Suppose the item sets S_l have been created and the items $a^{(l)}_{\max}$ are known. Then, all $\bar{P}_{c_l}$ and therefore P_0 can be found in time $\mathcal{O}(M \cdot n)$ and space $\mathcal{O}(M+n)$ without sorting the items according to their efficiency. The procedure also works for non-integral profits and sizes $p_j, s_j, c_l \in \mathbb{R}_{>0}$.*[1]

Proof. For one c_l, the median-based binary search presented in Lemma B.2 of Section B is applied to the set $\bar{S}_l$ to determine the greedy solution in time and space $\mathcal{O}(|\bar{S}_l|)$. The item with $p(a) = p^{(l)}_{\max}$ is already known. Hence, one $\bar{P}_{c_l}$ can be determined in time and

[1] During the proofreading of the thesis, I have discovered that the running time can probably be improved to an expression in $o(M \cdot n)$. In Chapter B, it is explained how the remaining capacity $c_l - F_{n_b}(i)$ of a knapsack can be filled with a median-based divide-and-conquer strategy. These are the values $\phi_l(c_l - F_{n_b}(i))$ of Subsection 3.5.3. Assume that in iteration $s-1$ these values are known for profits $i_1 < \ldots < i_3 < \ldots < i_{2r-1} < i_{2r+1}$ and therefore for capacities $c_l - F_{n_b}(i_1) \geq c_l - F_{n_b}(i_3) \geq \ldots \geq c_l - F_{n_b}(i_{2r-1}) \geq c_l - F_{n_b}(i_{2r+1})$. In the next iteration s, the values are determined for values $i_0, i_2, \ldots, i_{2r}, i_{2r+2}$ with $i_0 < i_1 < i_2 < \ldots < i_{2r-1} < i_{2r} < i_{2r+1} < i_{2r+2}$ and therefore with sizes $c_l - F_{n_b}(i_0) \geq c_l - F_{n_b}(i_1) \geq c_l - F_{n_b}(i_2) \geq \ldots \geq c_l - F_{n_b}(i_{2r}) \geq c_l - F_{n_b}(i_{2r+1}) \geq c_l - F_{n_b}(i_{2r+2})$ where the median-based algorithm uses the information of the last iteration $s-1$. This procedure is iterated until the values are known for all pairs $(i, F_{n_b}(i))$. Similarly, it is probably possible to use the same approach to iteratively find the greedy solutions for the knapsack sizes: when the values for the knapsack sizes $c_{l_1} < c_{l_3} < \ldots < c_{l_{2r-1}} < c_{l_{2r+1}}$ are known in iteration $s-1$, the values for $c_{l_0}, c_{l_2} \ldots, c_{l_{2r}}, c_{l_{2r+2}}$ with $c_{l_0} < c_{l_1} < c_{l_2} < c_{l_3} < \ldots < c_{l_{2r-1}} < c_{l_{2r}} < c_{l_{2r+1}} < c_{l_{2r+2}}$ can be determined like above in iteration s. Each iteration only needs time in $\mathcal{O}(n)$. Instead of using the median-based approach for every c_l again (which needs $\mathcal{O}(n)$ for every c_l), this iterative approach may find all $\bar{P}_{c_l}$ in $\mathcal{O}(\log(M)n)$ because only $\mathcal{O}(\log M)$ iterations are necessary until all c_l have been considered. However, this does not change the asymptotic running time of the final algorithm (see Theorem 3.23), which still contains an expression that dominates $\mathcal{O}(M \cdot n)$. We leave it as an open question of this thesis whether the sketched idea is correct, and if yes, to develop the actual algorithm.

space $\mathcal{O}(|\bar{S}_l|) \subseteq \mathcal{O}(n)$ and therefore all $\bar{P}_{c_l}$ in time $\mathcal{O}(M \cdot n)$. As the storage needed to find one $\bar{P}_{c_l}$ can be emptied after $\bar{P}_{c_l}$ has been found, we only need space in $\mathcal{O}(n)$ to find all $\bar{P}_{c_l}$ and additionally space in $\mathcal{O}(M)$ to save the $\bar{P}_{c_l}$. As the median-based approach also works for non-integral profits and sizes, the lemma follows. □

Lemma 3.12. *Suppose the c_l are already sorted such that $c_1 < \ldots < c_M$. Algorithm 3.2 constructs the sets S_l and finds the items $a^{(l)}_{\max}$ of profit $p^{(l)}_{\max}$ as well as the values $\bar{P}_{c_l}$ and P_0 in time $\mathcal{O}(M \cdot n)$ and space $\mathcal{O}(M + n)$. Moreover, knapsack sizes c_l with $P_{c_l} = \frac{\bar{P}_{c_l}}{c_l} < \frac{1}{2}P_0$ are discarded. The algorithm also works for non-integral profits and sizes $p_j, s_j, c_l \in \mathbb{R}_{>0}$.*

Proof. The algorithm is based on an idea by Lawler [63]: create M stacks, one for each S_l. Every item $a \in I$ is then added to the right stack by binary search. Note that $a_{\max}$ is always the item of highest profit for $\bar{S}_l = \cup_{l'=0}^{l} S_{l'}$ so that $a^{(l)}_{\max}$ is correctly defined. The re-combination of sets S_l and S_{l+1} when c_l is discarded can be done in $\mathcal{O}(1)$ if the right data structure is used (e.g. linked lists). The running time and the space needed follow. It is obvious that the procedure also works for non-integral p_j, s_j and c_l. □

Remark 3.13. Algorithm 3.2 only creates $\mathcal{O}(n)$ sets S_l. This is sufficient: it can easily be deduced that $S_l = \emptyset$ if one set $S_{l'}$ for $l' < l$ is followed by $S_{l''}$ with $l'' > l$ such that we do not have to save the sets $S_l = \emptyset$.

3.5.3 Scaling and Dividing: the Basic FPTAS

We introduce the basic algorithm, an extension of Lawler's algorithm [63]. The basic idea of Lawler's FPTAS for the normal 0-1 KP with knapsack size c is as follows:

First, a threshold T, which will be defined later, is introduced: items a_j with $p(a_j) \geq T$ are *large*, the other items *small*. For ease of notation, let $a_1, \ldots, a_{n_L}$ be the large items. They are scaled as follows: if $p(a_j) = p_j \in (2^k T, 2^{k+1}T]$ for $k \in \mathbb{N}$, the item has the scaled profit

$$q(a_j) = q_j := 2^k \left\lfloor \frac{p_j}{2^k K} \right\rfloor \ ,$$

where K will be defined later. We assume that $q(a_j)$ can be computed in time $\mathcal{O}(1)$ because k can be found in $\mathcal{O}(1)$ by computing the logarithm. Then, dynamic programming is applied to the large items with profits q_j, but unchanged sizes s_j, and dominated $F_{n_L}(i)$ are discarded.

Moreover, let $\phi(c - F_{n_L}(i))$ be the profit obtained by greedily filling up the remaining capacity $c - F_{n_L}(i)$ with small items: as in Subsection 3.5.2, items are added in non-increasing order of their efficiency $\frac{p_j}{s_j}$ (and starting with the item of largest efficiency) until adding the next item would result in a packing of size more than $c - F_{n_L}(i)$.

Algorithm 3.2: This algorithm constructs the sets S_l and finds the items with profit $p_{\max}^{(l)}$ as well as the values $\bar{P}_{c_l}$ and P_0. Knapsack sizes c_l with $P_{c_l} < \frac{1}{2}P_0$ are discarded.

for $l = 1, \ldots, M$ **do**
  $a_{\max}^{(l)} := \emptyset$;
for *all* $a \in I$ **do**
  Determine by binary search the value l such that $c_{l-1} < s(a) \leq c_l$;
  if S_l *not defined* **then**
    Create $S_l = \emptyset$;
  $S_l := S_l \cup \{a\}$;
$a_{\max} := \emptyset$;
for $l = 1, \ldots, M$ **do**
  for $a \in S_l$ **do**
    if $a_{\max} = \emptyset$ *or* $p(a) > p(a_{\max})$ **then**
      $a_{\max} := a$;
  $a_{\max}^{(l)} := a_{\max}$;
Determine all $\bar{P}_{c_l}$ (see Lemma 3.11);
$P_0 := 0$;
for $l = 1, \ldots, M$ **do**
  if $\frac{\bar{P}_{c_l}}{c_l} > P_0$ **then**
    $P_0 := \frac{\bar{P}_{c_l}}{c_l}$;
for $l = 1, \ldots, M-1$ **do**
  if $\frac{\bar{P}_{c_l}}{c_l} < \frac{1}{2}P_0$ **then**
    $C := C \setminus \{c_l\}$;
    $S_{l+1} := S_l \cup S_{l+1}$;
    Discard S_l;
if $\frac{\bar{P}_{c_M}}{c_M} < \frac{1}{2}P_0$ **then**
  $C := C \setminus \{c_l\}$;
  Discard S_M;

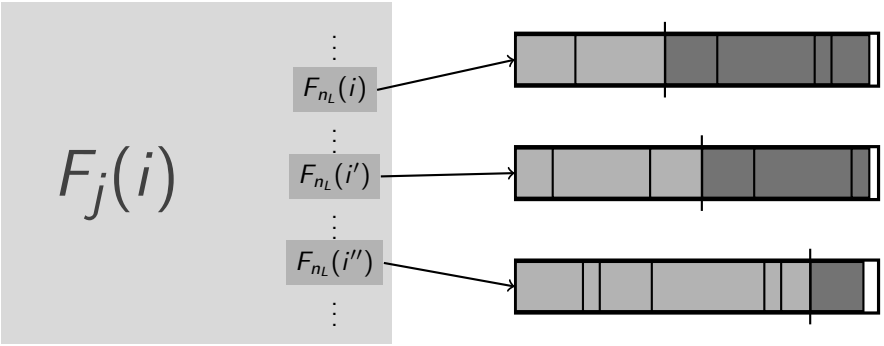


Figure 3.1: Principle of Lawler's algorithm: the values $F_{n_L}(i)$ are determined with the scaled profits of the large-profit items. Each $F_{n_L}(i)$ stands for a set of large-profit items (coloured medium gray). The remaining knapsack capacity $c - F_{n_L}(i)$ is greedily filled with small-profit items (coloured dark-grey). The best combination of large and small items is returned.

Lawler's algorithm for 0-1 KP then returns $\max_{F_{n_L}(i) \leq c} K \cdot i + \phi(c - F_{n_L}(i))$, the best combination of large and small items. The principle is shown in Figure 3.1.

We now want to use the same principle for KPIP and combine it with the idea from Subsection 3.5.1: first, the $F_{n_L}(i)$ are determined for the scaled large items and the dominated values discarded. For every c_l, we take

$$\bar{P}_l^{(1)} := \max_{F_{n_L}(i) \leq c_l} K \cdot i + \phi_l(c_l - F_{n_L}(i)) \ , \tag{3.10}$$

where ϕ_l only uses small items in $\bar{S}_l = \{a_i \mid s(a_i) \leq c_l\}$ to fill the remaining capacity $c_l - F_{n_L}(i)$. Therefore, the $F_{n_L}(i)$ are calculated only once and used to determine the $\bar{P}_l^{(1)}$ for all c_l similar to Subsection 3.5.1. Figure 3.2 illustrates the idea.

Lemma 3.14. *Let $\tilde{V}_{c_l}$ be the items chosen by the algorithm for knapsack c_l, and let $\overline{APP}_{c_l} := \sum_{a \in \tilde{V}_{c_l}} p(a)$ be their profit. Let $c_{\min} = c_1$ be the smallest knapsack size. By setting $T := \frac{1}{2}\varepsilon \bar{P}_{c_{\min}}$ and $K := \frac{1}{4}\varepsilon^2 \bar{P}_{c_{\min}}$, we have $\overline{APP}_{c_l} \geq \bar{P}_l^{(1)} \geq (1-\varepsilon)\overline{\mathrm{OPT}}_{c_l}(I)$ for all c_l. Moreover, the algorithm (including the dynamic program) also works for non-integral profits and sizes $p_j, s_j, c_l \in \mathbb{R}_{>0}$.*

Proof. We want to choose K and T such that $\bar{P}_l^{(1)} \geq (1-\varepsilon)\,\overline{\mathrm{OPT}}_{c_l}(I)$ holds for all c_l.

Let us fix one knapsack size c_l. First of all, the algorithm obviously determines a set of items that fits into c_l, i.e. a feasible solution.

Now, let $V = V_{c_l} = V_L \cup V_S \subseteq \{a_1, \ldots, a_n\}$ be the optimal choice of items for knapsack c_l, where $V_L = V_{L,c_l}$ are the large and $V_S = V_{S,c_l}$ the small items. Let

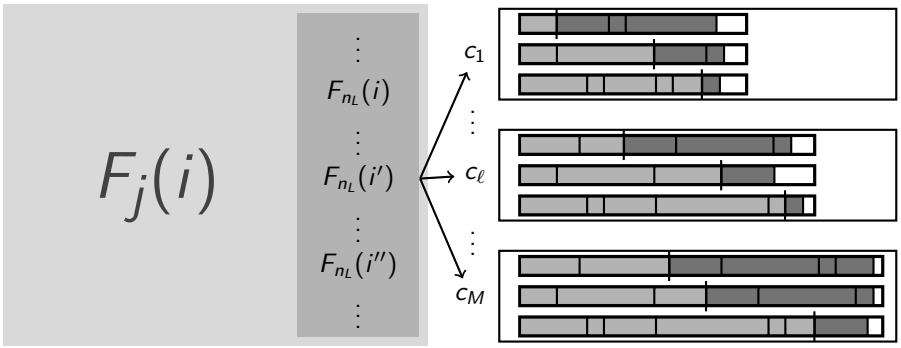


Figure 3.2: The table of the $F_j(i)$ is computed in such a way that the $F_{n_L}(i)$ can be used to determine the best combination of large- and small-profit items for all knapsack sizes c_l.

$\overline{OPT}_L = \overline{OPT}_{L,c_l}$ be the profit of the large and $\overline{OPT}_S = \overline{OPT}_{S,c_l}$ be the profit of the small items, i.e.

$$\overline{\mathrm{OPT}}_{c_l}(I) = \overline{OPT}_L + \overline{OPT}_S = \sum_{a \in V_L} p(a) + \sum_{a \in V_S} p(a) \ . \tag{3.11}$$

The large items in V_L have the total scaled profit

$$i_L := \sum_{a \in V_L} q(a) \ . \tag{3.12}$$

Since the dynamic program determines all optimal, non-dominated solutions for the scaled profits q_j, there is a pair $(i_{L,\mathrm{Dom}}, F_{n_L}(i_{L,\mathrm{Dom}}))$ that is equal to or that dominates $(i_L, s(V_L))$ such that

$$i_{L,\mathrm{Dom}} \geq i_L \quad \text{and} \quad c_l \geq s(V_L) \geq F_{n_L}(i_{L,\mathrm{Dom}}) \ . \tag{3.13}$$

On the other hand, let $\tilde{V} = \tilde{V}_L \cup \tilde{V}_S$ be the item set of the solution found by the algorithm. The total profit of the unscaled items is $\overline{APP}_{c_l} := p(\tilde{V}) = \sum_{a \in \tilde{V}} p(a)$. Let $\overline{APP}_L$ be the (unscaled) total profit of the large items and $\overline{APP}_S$ be the (unscaled) total profit of the small items so that

$$\overline{APP}_{c_l} = \overline{APP}_L + \overline{APP}_S = \sum_{a \in \tilde{V}_L} p(a) + \sum_{a \in \tilde{V}_S} p(a) \ . \tag{3.14}$$

We denote by

$$\tilde{i}_L := \sum_{a \in \tilde{V}_L} q(a) \tag{3.15}$$

the profit of the scaled large items in $\tilde{V}_L$. Since $\tilde{V}_L$ is chosen by the dynamic program with the scaled profits, we have $s(\tilde{V}_L) = F_{n_L}(\tilde{i}_L)$, and the profit of the small items in $\tilde{V}_S$ satisfies

$$p(\tilde{V}_S) = \phi_l(c_l - F_{n_L}(\tilde{i}_L)) \ . \tag{3.16}$$

Finally, let $k(a)$ be the exponent for a large item a such that $p(a) \in (2^{k(a)}T, 2^{k(a)+1}T]$. Thus, we have the following:

$$\begin{aligned}
\overline{APP}_{c_l} &\overset{(3.14)}{=} \overline{APP}_L + \overline{APP}_S = \sum_{a\in\tilde{V}_L} p(a) + \sum_{a\in\tilde{V}_S} p(a) \\
&\geq \sum_{a\in\tilde{V}_L} K \cdot \underbrace{\left\lfloor \frac{p(a)}{2^{k(a)}K} \right\rfloor 2^{k(a)}}_{=q(a)} + \sum_{a\in\tilde{V}_S} p(a) \\
&\overset{(3.16)}{=} \sum_{a\in\tilde{V}_L} K \cdot q(a) + \phi_l \left(c_l - F_{n_L}(\tilde{i}_L)\right) \\
&\overset{(3.15)}{=} K \cdot \tilde{i}_L + \phi_l \left(c_l - F_{n_L}(\tilde{i}_L)\right) \\
&= \max_{F_{n_L}(i)\leq c_l} K \cdot i + \phi_l(c_l - F_{n_L}(i)) \overset{(3.10)}{=} \bar{P}_l^{(1)} \\
&\overset{(3.13)}{\geq} K \cdot i_{L,\mathrm{Dom}} + \phi_l(c_l - F_{n_L}(i_{L,\mathrm{Dom}})) \\
&\overset{(3.13)}{\geq} K \cdot i_L + \phi_l(c_l - F_{n_L}(i_{L,\mathrm{Dom}})) \\
&\overset{(3.12)}{=} \sum_{a\in V_L} K \cdot \left\lfloor \frac{p(a)}{2^{k(a)}K} \right\rfloor 2^{k(a)} + \phi_l \left(c_l - F_{n_L}(i_{L,\mathrm{Dom}})\right) \\
&\geq \sum_{a\in V_L} K \cdot \left(\frac{p(a)}{2^{k(a)}K} - 1 \right) 2^{k(a)} + \phi_l \left(c_l - F_{n_L}(i_{L,\mathrm{Dom}})\right) \\
&= \sum_{a\in V_L} \left(p(a) - K \cdot 2^{k(a)} \right) + \phi_l \left(c_l - F_{n_L}(i_{L,\mathrm{Dom}})\right) =: (*) \ .
\end{aligned} \tag{3.17}$$

The Identity (3.17) holds because $\tilde{V}$ is exactly the solution set found by the algorithm. Since $p(a) \in (2^{k(a)}T, 2^{k(a)+1}T]$, i.e. $\frac{p(a)}{T} \geq 2^{k(a)}$, holds, we get

$$\begin{aligned}
(*) &\geq \sum_{a\in V_L} \left(p(a) - K \cdot \frac{p(a)}{T} \right) + \phi_l \left(c_l - F_{n_L}(i_{L,\mathrm{Dom}})\right) \\
&\overset{(3.11)}{=} \left(1 - \frac{K}{T}\right) \overline{OPT}_L + \phi_l \left(c_l - F_{n_L}(i_{L,\mathrm{Dom}})\right) \\
&\overset{(3.13)}{\geq} \left(1 - \frac{K}{T}\right) \overline{OPT}_L + \phi_l \left(c_l - s(V_L)\right) \ .
\end{aligned}$$

There are two possibilities: either ϕ_l manages to greedily fill the entire capacity $c_l - s(V_L)$ with small items. Then obviously $\phi_l(c_l - s(V_L)) = \overline{OPT}_S$. Otherwise, we have similar to Inequality (3.8) that $\phi_l(c_l - s(V_L)) + p(a_{\mathrm{next}}) \geq \overline{OPT}_S$, where a_{next} is the

item that would be greedily added if there were enough knapsack capacity left. Since a_{next} is a small item, we have $p(a_{\text{next}}) < T$ and therefore

$$\begin{aligned}\overline{APP}_{c_l} &\geq \left(1-\frac{K}{T}\right)\overline{OPT}_L + \phi_l\left(c_l - s(V_L)\right) \geq \left(1-\frac{K}{T}\right)\overline{OPT}_L + \overline{OPT}_S - T \\ &\geq \left(1-\frac{K}{T}\right)\left(\overline{OPT}_L + \overline{OPT}_S\right) - T \overset{(3.11)}{=} \left(1-\frac{K}{T}\right)\overline{\text{OPT}}_{c_l}(I) - T \ .\end{aligned}$$

As we want that $\overline{APP}_{c_l} \geq (1-\varepsilon)\overline{\text{OPT}}_{c_l}(I)$ for all c_l, we have to ensure that

$$\frac{K}{T}\overline{\text{OPT}}_{c_l}(I) \leq \lambda\varepsilon\overline{\text{OPT}}_{c_l}(I) \Leftrightarrow \frac{K}{T} \leq \lambda\varepsilon \text{ and } T \leq (1-\lambda)\,\varepsilon\overline{\text{OPT}}_{c_l}(I),\ \forall l \in \{1,\ldots,M\}$$

where $\lambda \in (0,1)$. Let

$$c_{\min} := \min\left\{c_l \;\middle|\; l \in \{1,\ldots,M\}\right\} = c_1 \ .$$

Since $\overline{\text{OPT}}_{c_{\min}}(I) \leq \overline{\text{OPT}}_{c_l}(I)$ for $l \in \{1,\ldots,M\}$, we see that we must have

$$T \leq (1-\lambda)\varepsilon\overline{\text{OPT}}_{c_{\min}}(I) \quad \text{and} \quad K \leq \lambda\varepsilon T = \varepsilon^2\lambda(1-\lambda)\overline{\text{OPT}}_{c_{\min}}(I) \ . \tag{3.18}$$

Note that a scaling factor K as large as possible yields smaller scaled profits q_j and therefore a faster running time as stated by Lawler [63]. Choosing $\lambda = \frac{1}{2}$ maximizes K and therefore minimizes the running time. Together with $\bar{P}_{c_{\min}} \leq \overline{\text{OPT}}_{c_{\min}}(I)$, we have proved the correctness if we set

$$T = \frac{1}{2}\varepsilon\bar{P}_{c_{\min}} \text{ and } K = \frac{1}{4}\varepsilon^2\bar{P}_{c_{\min}} \ . \tag{3.19}$$

Moreover, $\overline{APP}_{c_l} = \sum_{a\in\tilde{V}} p(a) \geq \bar{P}_l^{(1)} \geq (1-\varepsilon)\overline{\text{OPT}}_{c_l}$ holds because of (3.17).

Finally, it is obvious that the algorithm also works for non-integral $p_j, s_j, c_l \in \mathbb{R}_{>0}$ because the large-item profits are scaled to integral values, and the dynamic program also works for non-integral s_j and c_l (see Theorem 3.8). Moreover, the greedy algorithm presented in this proof does not use the fact that the profits are integral. (This proof is adapted from Lawler's proof [63].) □

After having found the $\bar{P}_l^{(1)}$, Algorithm `MaxSolution` yields a $(1-\varepsilon)$ solution for 0-1 KPIP.

Let us now determine the running time. We will see how to optimize it by a change to the calculation of the $F_{n_L}(i)$.

Running Time for the Large-Item Computation

We have already seen in Theorem 3.8 the general running time of $\mathcal{O}(n_L \cdot i_{\max})$ for the dynamic program, where $i_{\max}$ is the largest total profit i of scaled items we have to consider. Moreover, $\mathcal{O}(i_{\max} + M)$ is an upper bound on the time needed to discard dominated $F_{n_L}(i)$ and determine for all c_l the profits i such that $F_{n_L}(i) \leq c_l$. Therefore, we have to bound $i_{\max}$.

Let V_{L,c_l} be a choice of large items of a solution for knapsack c_l, and let $i_{c_l} := \sum_{a \in V_{L,c_l}} q(a)$ be the corresponding total scaled profit. Obviously, we have $i_{c_l} \leq i_{\max}$. As in the proof of Lemma 3.14, let $k(a)$ be the exponent for a large item a such that $p(a) \in (2^{k(a)}T, 2^{k(a)+1}T]$.

There is a general bound for every i_{c_l} and therefore for $i_{\max}$: we have

$$\begin{aligned} i_{c_l} = \sum_{a \in V_{L,c_l}} q(a) = \sum_{a \in V_{L,c_l}} \left\lfloor \frac{p(a)}{2^{k(a)}K} \right\rfloor 2^{k(a)} \leq \sum_{a \in V_{L,c_l}} \frac{p(a)}{K} \leq \frac{\overline{\mathrm{OPT}}_{c_l}(I)}{K} \\ \leq \frac{\overline{\mathrm{OPT}}_{c_{\max}}(I)}{K} = \frac{\overline{\mathrm{OPT}}(I)}{K} \ , \end{aligned} \tag{3.20}$$

where $c_{\max} := \max_{l \in \{1,\dots,M\}} c_l = c_M = 1$. As

$$\overline{\mathrm{OPT}}(I) = \overline{\mathrm{OPT}}_{c_{\max}}(I) \overset{(3.3)}{=} c_{\max} \cdot \mathrm{OPT}_{c_{\max}}(I) \overset{\text{Thm. 3.9}}{\leq} 2c_{\max}P_{c_{\max}} \leq 2c_{\max}P_0 \ , \tag{3.21}$$

we see that

$$i_{c_l} \leq i_{\max} \overset{(3.20)}{\leq} \frac{\overline{\mathrm{OPT}}_{c_{\max}}(I)}{K} = \frac{\overline{\mathrm{OPT}}(I)}{K} \leq \frac{2c_{\max}P_0}{K} \tag{3.22}$$

so that the large-item computation is done in $\mathcal{O}(i_{\max} \cdot n_L) = \mathcal{O}(\frac{2c_{\max}P_0}{K} \cdot n_L)$ as seen in Theorem 3.8.

We have $K \overset{(3.19)}{=} \frac{\varepsilon^2}{4}\bar{P}_{c_{\min}} \overset{(3.7)}{=} \frac{\varepsilon^2}{4}c_{\min}P_{c_{\min}}$. Since we assume without loss of generality that $P_{c_l} \geq \frac{P_0}{2}$ (see Corollary 3.10), we get

$$\begin{aligned} i_{\max} \leq \frac{\overline{\mathrm{OPT}}_{c_{\max}}(I)}{K} = \frac{\overline{\mathrm{OPT}}(I)}{K} \leq \frac{2c_{\max}P_0}{K} = \frac{2c_{\max}P_0}{\frac{\varepsilon^2}{4}c_{\min}P_{c_{\min}}} \\ \leq \frac{8c_{\max}P_0}{\varepsilon^2 c_{\min}\frac{P_0}{2}} = \frac{16}{\varepsilon^2}\frac{c_{\max}}{c_{\min}} \end{aligned} \tag{3.23}$$

so that the overall running time for the large-item computation is bounded by

$$\mathcal{O}\left(\frac{2c_{\max}P_0}{K} \cdot n_L\right) = \mathcal{O}\left(\frac{16}{\varepsilon^2}\frac{c_{\max}}{c_{\min}}n_L\right) \ .$$

The running time therefore depends on the ratio $\frac{c_{\max}}{c_{\min}}$, which may be quite large. Fortunately, there is a method to control it.

For $w > 0$, partition $[c_{\min}, c_{\max}]$ into intervals

$$C_b := \left\{ c \in C \;\middle|\; c \in \left(c_{\min}^{w\cdot(b+1)}, c_{\min}^{w\cdot b} \right] \right\} \text{ and } C_{\lfloor \frac{1}{w} \rfloor + 1} := \left\{ c \in C \;\middle|\; c \in \left[c_{\min}, c_{\min}^{w\cdot \lfloor \frac{1}{w} \rfloor} \right] \right\} \ .$$

The value for w will be chosen later.

For every C_b, we apply our algorithm only to the bin sizes $c_l \in C_b$. Therefore, we have an adapted threshold $T_b := \frac{1}{2}\varepsilon \bar{P}_{c_{\min}^{(b)}}$ and scaling factor $K_b := \frac{1}{4}\varepsilon^2 \bar{P}_{c_{\min}^{(b)}}$, where $c_{\min}^{(b)}$ is the smallest knapsack size in C_b. The disadvantage is obviously that we have to partition the items into large and small ones as well as scale the items again if we consider another C_b. The advantage is a decreased running time: if $c_{\max}^{(b)}$ is the largest knapsack size in C_b, we have

$$\frac{c_{\max}^{(b)}}{c_{\min}^{(b)}} \leq \frac{c_{\min}^{w\cdot b}}{c_{\min}^{w\cdot(b+1)}} = \frac{1}{c_{\min}^{w}} \ . \tag{3.24}$$

Should $M \leq \lfloor \frac{1}{w} \rfloor + 1$ hold, we instead set $C_b = C_l := \{c_l\}$ for $l \in \{1, \ldots, M\}$: we have $c_{\max}^{(b)}/c_{\min}^{(b)} = 1$ and less C_b than in the case where we partition into $(c_{\min}^{w\cdot(b+1)}, c_{\min}^{w\cdot b}]$.

Therefore, we get the following theorem. We assume for convenience that the first n_b items $a_1, \ldots, a_{n_b}$ are the large items for C_b.

Theorem 3.15. *Let the $F_{n_b}(i)$ be non-dominated. Let n_b be the number of large items for knapsack interval C_b, and let n_L be an upper bound on all n_b. The overall time and space bound for the large-item computation in one knapsack interval C_b is in $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)} n_b)$, which is in $\mathcal{O}(\frac{1}{\varepsilon^2}\frac{1}{c_{\min}^{w}} n_L)$ for $M > \lfloor \frac{1}{w} \rfloor + 1$ and in $\mathcal{O}(\frac{1}{\varepsilon^2} n_L)$ otherwise.*

Note that the running time for re-partitioning and re-scaling is not considered yet, as well as the optimal choice of w.

Adding the Small Items Efficiently

Let C_b be a knapsack interval and $F_{n_b}(i)$ the values $F_j(i)$ that consider all n_b large items. For every c_l and $F_{n_b}(i) \leq c_l$, we have to determine $\phi_l(c_l - F_{n_b}(i))$. As already stated in Subsection 3.5.2, we can avoid sorting the items according to their efficiency: we use median finding in a divide-and-conquer strategy on the small items in $\bar{S}_l$, similar to the idea by Lawler [63]. Details can again be found in Appendix B. The algorithm also returns item sets $\tilde{J}^{(i)}$ from which the item choice for every $c_l - F_{n_b}(i)$ can be reconstructed.

Theorem 3.16. *For one $c_l \in C_b$, let the small items $\bar{S}_{l,\text{small}}$ in $\bar{S}_l$ be given. The values $\phi_l(c_l - F_{n_b}(i))$ can then be determined in time $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)} + n \cdot \log(\frac{1}{\varepsilon} c_{\max}^{(b)}/c_{\min}^{(b)}))$ and in*

space $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)} + n)$. *The method employed also returns item sets* $\tilde{J}^{(i')}$ *such that the item set* $J^{(i)}$ *for profit* $\phi_l(c_l - F_{n_b}(i))$ *can be constructed by* $J^{(i)} = \bigcup_{i'=i_{c_l}}^{i} \tilde{J}^{(i')}$. *There are* $\mathcal{O}(i_{\max}) \overset{(3.23)}{=} \mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)})$ *sets* $\tilde{J}^{(i)}$, *which are stored in space* $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)} + n)$. *The method also works for non-integral profits and sizes* $p_j, s_j, c_l \in \mathbb{R}_{>0}$.

Putting the Basic FPTAS Together

Algorithm 3.3 presents the basic algorithm for 0-1 KPIP. The principle of the algorithm is illustrated in Figure 3.3.

Algorithm 3.3: The basic algorithm for 0-1 KPIP

Input: Item set I, sorted knapsacks $C = \{c_1, \ldots, c_M\}$
Output: Profit P, solution set V and knapsack size c_{sol}

1 Determine the sets S_l, the approximations $\bar{P}_{c_l}$ and P_0 and adapt C accordingly (Algorithm 3.2);
2 Partition C into the sets C_b and determine at the same time $c_{\min}^{(b)}$ and $c_{\max}^{(b)}$ for all C_b;
3 Set $P := 0$, $c_{\text{sol}} := \emptyset$, and $V := \emptyset$;
4 **for** *all* $C_b \neq \emptyset$ **do**
5 Define T_b and K_b;
6 Partition the items in every S_l for $c_l \leq c_{\max}^{(b)}$ into large and small items and scale the large ones;
7 Calculate the $F_j(i)$. Discard dominated $F_{n_b}(i)$;
8 **for** *all* $c_l \in C_b$ **do**
9 Find $\bar{P}_l^{(1)} = \max_{F_{n_b}(i) \leq c_l} K_b \cdot i + \phi_l(c_l - F_{n_b}(i))$ where the values of ϕ_l are determined as stated in Theorem 3.16;
10 **if** $\frac{1}{c_l}\bar{P}_l^{(1)} > P_1$ **then**
11 Set $P := \frac{\bar{P}_l^{(1)}}{c_l}$;
12 Determine the item set V_t for solution $\bar{P}_l^{(1)}$ and set $V := V_t$ as well as $c_{\text{sol}} := c_l$;
13 **return** *P, V and c_{sol};*

As the algorithm is an implementation of Algorithm `MaxSolution`, its correctness follows from Lemma 3.4, Lemma 3.5 and Lemma 3.14. Note that we keep the structure of the S_l to (heuristically) faster get the small items in $\bar{S}_l$ for the calculations in Step 9.

We can now state the running time and the space bound.

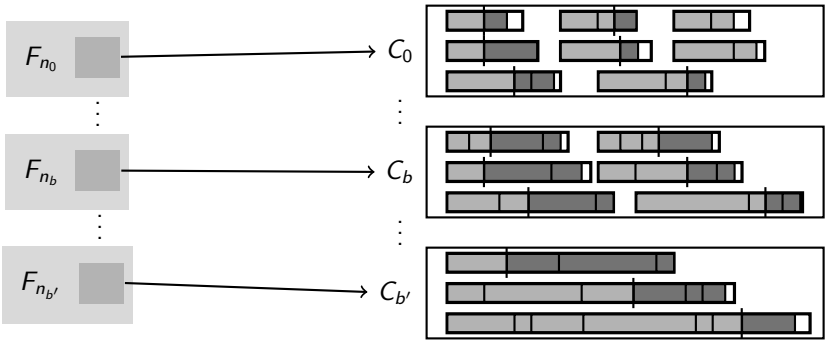


Figure 3.3: The principle of the 0-1 KPIP algorithm: the tables F_{n_b} of the entries $F_j(i)$ are determined for every C_b. Then, the $F_{n_b}(i)$ are only used for the knapsack sizes $c_l \in C_b$ to find the best combination of large items (coloured medium gray) and small items (coloured dark-grey).

Lemma 3.17. *Fix one $C_b \neq \emptyset$. The inner for-loop of Algorithm 3.3 (Steps 8–12) needs for all $c_l \in C_b$ time in $\mathcal{O}(|C_b| \cdot (\frac{1}{\varepsilon^2} c^{(b)}_{\max}/c^{(b)}_{\min} + n \cdot \log(\frac{1}{\varepsilon} c^{(b)}_{\max}/c^{(b)}_{\min})))$ and space in $\mathcal{O}(\frac{1}{\varepsilon^2} c^{(b)}_{\max}/c^{(b)}_{\min} + n)$.*

Proof. Fix one $c_l \in C_b$. We calculate $K_b \cdot i + \phi_l(c_l - F_{n_b}(i))$ for $i \in \{1, \ldots, i_{c_l}\}$ where $F_{n_b}(i) \leq c_l$ for $i \leq i_{c_l}$. Here, $i_{c_l} \leq i_{\max}$ is the largest profit i we have to consider, i.e. $i_{c_l} \in \mathcal{O}(\frac{1}{\varepsilon^2} c^{(b)}_{\max}/c^{(b)}_{\min})$ holds (see Bound (3.23)). We do not have to consider values $i > i_{c_l}$ because $F_{n_b}(i) > c_l$ holds for these i: dominated $F_{n_b}(i)$ have been removed. The set of small items $\bar{S}_{l,\text{small}}$ for c_l can be constructed and stored in $\mathcal{O}(n)$ from the small items in the $S_{l'}$, $l' \leq l$ (see Step 6). Thus, all $\phi_l(c_l - F_{n_b}(i))$ for $c_l \in C_b$ can be found in time $\mathcal{O}(\frac{1}{\varepsilon^2} c^{(b)}_{\max}/c^{(b)}_{\min} + n \cdot \log(\frac{1}{\varepsilon} c^{(b)}_{\max}/c^{(b)}_{\min}))$ according to Theorem 3.16. Therefore, finding $\bar{P}^{(1)}_l$ needs time in $\mathcal{O}(|C_b| \cdot (\frac{1}{\varepsilon^2} c^{(b)}_{\max}/c^{(b)}_{\min} + n \cdot \log(\frac{1}{\varepsilon} c^{(b)}_{\max}/c^{(b)}_{\min})))$ over all $c_l \in C_b$.

The check of the if-condition only needs time and space $\mathcal{O}(1)$. For one $c_l \in C_b$, the body of the if-condition needs $\mathcal{O}(n_b) \subseteq \mathcal{O}(n)$ for backtracking of the large items in the solution and $\mathcal{O}(n + \frac{1}{\varepsilon^2} c^{(b)}_{\max}/c^{(b)}_{\min})$ to find the small items from the $\tilde{J}^{(i')}$ (see Theorem 3.16). Hence, the if-condition with its body needs time in $\mathcal{O}(|C_b| \, (n + \frac{1}{\varepsilon^2} c^{(b)}_{\max}/c^{(b)}_{\min}))$ for all $c_l \in C_b$. As this is dominated by the time to find $\bar{P}^{(1)}_l$, the time complexity bound follows.

We now consider the space complexity. Only the values $\phi_l(F_{n_b}(i) - c_l)$ as well as the sets $\tilde{J}^{(i')}$ and V_t are newly determined in the inner for-loop. The calculation of $\bar{P}^{(1)}_l$ for one c_l therefore needs space in $\mathcal{O}(\frac{1}{\varepsilon^2} c^{(b)}_{\max}/c^{(b)}_{\min} + n)$ and the body of the if-condition space in $\mathcal{O}(\frac{1}{\varepsilon^2} c^{(b)}_{\max}/c^{(b)}_{\min} + n)$ (see Theorem 3.16). This also includes the space for backtracking to find the large items, which is bounded by $\mathcal{O}(n)$. The newly calculated values (except of P_1, V and c_{sol}) can be discarded after one iteration of the inner for-loop, i.e. after having processed one c_l, and the values P_1, V and c_{sol} are overwritten if

a new optimum is found. The overall space bound over all $c_l \in C_b$ is therefore in $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)} + n)$. □

Lemma 3.18. *The outer for-loop of Algorithm 3.3 (Steps 4–12) needs time in*

$$\mathcal{O}\left(\frac{1}{\varepsilon^2}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}} n_L + |C_b| \cdot \left(\frac{1}{\varepsilon^2}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}} + n \cdot \log\left(\frac{1}{\varepsilon}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}\right)\right)\right)$$

for one $C_b \neq \emptyset$. The space needed is in $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b_0)}/c_{\min}^{(b_0)} n_L + n)$. Here, $n_L \leq n$ is an upper bound on the number of the large items for all C_b, and $c_{\max}^{(b_0)}/c_{\min}^{(b_0)} := \max_{C_b} c_{\max}^{(b)}/c_{\min}^{(b)}$.

Proof. Fix $C_b \neq \emptyset$. Defining T_b and K_b can be done in time and space $\mathcal{O}(1)$.

The partition of the S_l into large and small can be done in time $\mathcal{O}(n)$ and in space $\mathcal{O}(n)$: we only have $\mathcal{O}(n)$ sets S_l as seen in Remark 3.13.

We need time and space in $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)} n_L)$ for the dynamic program, i.e. to calculate and save the $F_j(i)$ according to Theorem 3.15. Discarding the dominated $F_{n_b}(i)$ can be done in time $\mathcal{O}(i_{\max}) \overset{(3.23)}{=} \mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)})$ as seen in Lemma 3.7.

Lemma 3.17 states that we need time in $\mathcal{O}(|C_b| \cdot (\frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)} + n \cdot \log(\frac{1}{\varepsilon} c_{\max}^{(b)}/c_{\min}^{(b)})))$ and space in $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)} + n)$ for the inner for-loop.

The stated time and space complexity follow. □

Theorem 3.19. *The basic FPTAS (Algorithm 3.3) determines a solution of 0-1 KPIP in time*

$$\mathcal{O}\left(\min\left\{\left(\left\lfloor\frac{1}{w}\right\rfloor + 1\right)\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}, M\right\} \cdot \frac{1}{\varepsilon^2} n_L + M \cdot \left(\frac{1}{\varepsilon^2}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}} + n \cdot \log\frac{1}{\varepsilon}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}\right)\right)$$

and in space $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b_0)}/c_{\min}^{(b_0)} n_L + M + n)$. Note that $c_{\max}^{(b_0)}/c_{\min}^{(b_0)} \leq \frac{1}{c_{\min}^{(b)}}$ for $M > \lfloor\frac{1}{w}\rfloor + 1$ and $c_{\max}^{(b_0)}/c_{\min}^{(b_0)} = 1$ otherwise. The FPTAS also works for non-integral profits and sizes $p_j, s_j, c_l \in \mathbb{R}_{>0}$ and therefore for the formal definition of 0-1 KPIP in Subsection 1.3.3.

Proof. Algorithm 3.2 needs time in $\mathcal{O}(M \cdot n)$ and space in $\mathcal{O}(n + M)$ as seen in Lemma 3.12.

The partitioning of the knapsacks into C_b can be done in time and space $\mathcal{O}(M)$ because the knapsack sizes are already sorted and the logarithm—necessary to find for one c_l the right b such that $c_l \in C_b$—needs time in $\mathcal{O}(1)$. Moreover, we only save the sets $C_b \neq \emptyset$. During the partitioning, all values $c_{\min}^{(b)}$ and $c_{\max}^{(b)}$ can also be found, which additionally needs time and space in $\mathcal{O}(M)$. Details can be found in Subsection 5.5.1. Note that $C_b = \{c \in C \mid c \in (2^{-(b+1)}, 2^{-b}]\}$ holds in Subsection 5.5.1, which does not change the reasoning here.

The creation of P, c_{sol} and V is in $\mathcal{O}(1)$.

For the outer for-loop, remember that C_b contains the knapsacks $c_l \in (c_{\min}^{w\cdot(b+1)}, c_{\min}^{w\cdot b}]$ for $M > \lfloor \frac{1}{w} \rfloor + 1$ and $C_b = C_l = \{c_l\}$ for $M \leq \lfloor \frac{1}{w} \rfloor + 1$. There are therefore at most $\min\left\{\lfloor \frac{1}{w} \rfloor + 1, M\right\}$ sets C_b so that the outer for-loop is executed $\min\left\{\lfloor \frac{1}{w} \rfloor + 1, M\right\}$ times. Hence, its time complexity is in

$$\mathcal{O}\left(\min\left\{\left(\left\lfloor\frac{1}{w}\right\rfloor + 1\right)\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}, M\right\}\frac{1}{\varepsilon^2}n_L + M \cdot \left(\frac{1}{\varepsilon^2}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}} + n \cdot \log\frac{1}{\varepsilon}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}\right)\right)$$

as seen in Lemma 3.18. We have used that $c_{\max}^{(b_0)}/c_{\min}^{(b_0)} \leq \frac{1}{c_{\min}^{w}}$ holds for $M > \lfloor \frac{1}{w} \rfloor + 1$ according to Inequality (3.24) and that $c_{\max}^{(b_0)}/c_{\min}^{(b_0)} = 1$ for $M \leq \lfloor \frac{1}{w} \rfloor + 1$. The space needed is bounded by

$$\mathcal{O}\left(\frac{1}{\varepsilon^2}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}n_L + M + n\right) \ ,$$

again because the newly calculated values except of P_1, V and c_{sol} can be discarded after one iteration of the for-loop.

It is easy to see that the FPTAS also works for non-integral profits p_j, sizes s_j, and knapsacks c_l. Note that this has already been stated for the more complicated steps like Algorithm 3.2 (see Lemma 3.12), the dynamic program to determine the $F_j(i)$ with the scaled profits q_j, and the calculation of the $\phi_l(c_l - F_{n_b}(i))$ (see Lemma 3.14 and Theorem 3.16). □

Assumption 3.1. For the remaining thesis, we assume that the profits $p(a)$, sizes $s(a)$, and knapsacks c, c_l for KP and KPIP are not necessarily integral (or better $p(a), s(a), c_l \in (0, 1]$). This is the case for the column generation of our AFPTAS for Bin Packing and Variable-sized Bin Packing (see Remark 2.11 in Subsection 2.5.1). In fact, this assumption is in line with the formal definition of KPIP in Subsection 1.3.3.

3.5.4 Improved FPTAS: Reducing Running Time and Storage Space

Lawler [63] presented two techniques to decrease the running time and the storage space of the basic FPTAS algorithm. We adapt the notation in this section to explain them: let $q_1, \ldots, q_{m_b}$ from now on be all different scaled profits that occur for a fixed knapsack interval C_b. We assume that they are sorted in non-increasing order, i.e. $q_1 \geq \ldots \geq q_{m_b}$.[2]

[2]It should be noted that $p(a_1) < p(a_2)$ does not always imply $q(a_1) \leq q(a_2)$. Let $p(a_1) = 2^{k+1}T_b$ and $p(a_2) = (2^{k+1} + 2^{k+1}\delta)T_b$ for $\delta > 0$ small enough such that $q(a_1) = \lfloor \frac{4}{\varepsilon} \rfloor 2^k > q(a_2) = \lfloor \frac{2+2\delta}{\varepsilon} \rfloor 2^{k+1} = \lfloor \frac{2}{\varepsilon} \rfloor 2^{k+1}$, i.e. $\lfloor \frac{4}{\varepsilon} \rfloor > 2\lfloor \frac{2}{\varepsilon} \rfloor$. Values $\varepsilon > 0$ for which the latter condition holds can be easily found. However, the scaling monotonicity can be guaranteed by assuming without loss of generality that $\varepsilon = \frac{1}{2^\kappa}$ for $\kappa \in \mathbb{N}$. Even if this is not assumed, the possible non-monotonicity does not change the

First, it is quite obvious that an optimal solution cannot contain too many items of large profit. Let a be a large-profit item with $p(a) \in L^{(k,b)} := (2^k T_b, 2^{k+1} T_b]$ and therefore with the scaled profit

$$q_j \in \tilde{L}^{(k,b)} := \left(\left\lfloor \frac{2}{\varepsilon} \right\rfloor 2^k, \left\lfloor \frac{4}{\varepsilon} \right\rfloor 2^k \right] .$$

We can show that only a subset of items for every scaled profit $q_j \in \tilde{L}^{(k,b)}$ has to be kept. Moreover, an optimal solution using r items of profit q_j obviously uses the r items of smallest size. This allows us to efficiently choose the large-profit items we have to consider.

Lemma 3.20. *For one C_b, there are at most $m_b \in \mathcal{O}(\frac{2}{\varepsilon} \cdot \log(\frac{1}{\varepsilon} c^{(b)}_{\max}/c^{(b)}_{\min}))$ different scaled profits q_j. We need at most $n_b \leq n_L \in \mathcal{O}(\frac{1}{\varepsilon^2} c^{(b)}_{\max}/c^{(b)}_{\min})$ items of large profit, which can be found in time and space $\mathcal{O}(n + \frac{2}{\varepsilon} \log(\frac{1}{\varepsilon} c^{(b)}_{\max}/c^{(b)}_{\min}))$.*

Proof. Let C_b be a fixed interval of knapsack sizes. Note that an item of profit $p(a) \in L^{(k,b)}$ has a scaled profit $q_j = \lfloor \frac{p_j}{2^k K_b} \rfloor 2^k \in \tilde{L}^{(k,b)} = (\lfloor \frac{2}{\varepsilon} \rfloor 2^k, \lfloor \frac{4}{\varepsilon} \rfloor 2^k]$ so that each interval $L^{(k,b)}$ provides only $\mathcal{O}(\frac{2}{\varepsilon})$ different scaled profits q_j.

On the other hand, at most $\mathcal{O}(\log(\frac{1}{\varepsilon} c^{(b)}_{\max}/c^{(b)}_{\min}))$ intervals $L^{(k,b)}$ are needed to cover the interval of the large items $[T_b, \overline{\mathrm{OPT}}_{c^{(b)}_{\max}}(I)] \subseteq [T_b, 2c^{(b)}_{\max} P_0]$: obviously, an item a with $s(a) \leq c^{(b)}_{\max}$ cannot have a profit $p(a)$ larger than

$$\overline{\mathrm{OPT}}_{c^{(b)}_{\max}}(I) \overset{\text{Thm. 3.9}}{\leq} 2\bar{P}_{c^{(b)}_{\max}} \overset{(3.7)}{=} 2c^{(b)}_{\max} P_{c^{(b)}_{\max}} \leq 2c^{(b)}_{\max} P_0 \ . \tag{3.25}$$

Thus, there are at most $m_b \in \mathcal{O}(\frac{2}{\varepsilon} \cdot \log(\frac{1}{\varepsilon} c^{(b)}_{\max}/c^{(b)}_{\min}))$ different scaled values q_j, which proves the first part of the lemma.

An upper bound on the total profit that can be obtained with the scaled items is

$$\frac{\overline{\mathrm{OPT}}_{c^{(b)}_{\max}}(I)}{K_b} \leq \frac{2c^{(b)}_{\max} P_0}{K_b} = \frac{2c^{(b)}_{\max} P_0}{\frac{1}{4}\varepsilon^2 c^{(b)}_{\min} P_{c^{(b)}_{\min}}} \overset{\text{Coroll. 3.10}}{\leq} \frac{16 P_0}{\varepsilon^2 P_0} \frac{c^{(b)}_{\max}}{c^{(b)}_{\min}} \in \mathcal{O}\left(\frac{1}{\varepsilon^2} \frac{c^{(b)}_{\max}}{c^{(b)}_{\min}} \right) \tag{3.26}$$

(see also Inequality (3.23)).

Fix one scaled profit q_j. Let $\tilde{a} \in I$ be an item with $p(\tilde{a}) \in L^{(k,b)}$ such that $q(\tilde{a}) = q_j = \lfloor \frac{p(\tilde{a})}{2^k K_b} \rfloor 2^k$. Thus, we have $q_j = q(\tilde{a}) \geq (\frac{p(\tilde{a})}{2^k K_b} - 1) 2^k \geq \frac{p(\tilde{a})}{K_b} - \frac{p(\tilde{a})}{T_b}$ and therefore

$$K_b q_j = K_b q(\tilde{a}) \geq \left(1 - \frac{K_b}{T_b}\right) p(\tilde{a}) = \left(1 - \frac{\varepsilon}{2}\right) p(\tilde{a}) \geq \frac{1}{2} p(\tilde{a}) \geq 2^{k-1} T_b \tag{3.27}$$

results and proofs in this subsection because the monotonicity is not used. This subsection then uses another order (see e.g. Lemma 3.21) where $q_1, \ldots, q_{j'}$ are the scaled profits in decreasing order for the largest interval $(2^k T_b, 2^{k+1} T_b]$, the items $q_{j'+1}, \ldots, q_{j''}$ are the scaled profits in decreasing order for $(2^{k-1} T_b, 2^k T_b]$ etc.

where we use the trivial bound $\varepsilon \leq 1$.

Hence, there cannot be more than

$$n_{L,j} \leq \left\lfloor \frac{2c_{\max}^{(b)} P_0 / K_b}{q_j} \right\rfloor \overset{(3.27)}{\leq} \left\lfloor \frac{2c_{\max}^{(b)} P_0}{2^{k-1} T_b} \right\rfloor \leq \frac{2c_{\max}^{(b)} P_0}{2^{k-1} T_b} = \frac{2}{2^{k-2}} \frac{c_{\max}^{(b)}}{c_{\min}^{(b)}} \frac{P_0}{\varepsilon P_{c_{\min}^{(b)}}}$$
$$\leq 2^{4-k} \frac{1}{\varepsilon} \frac{c_{\max}^{(b)}}{c_{\min}^{(b)}} \in \mathcal{O}\left(\frac{2^{-k}}{\varepsilon} \frac{c_{\max}^{(b)}}{c_{\min}^{(b)}} \right) \tag{3.28}$$

items of scaled profit $q_j \in \tilde{L}^{(k,b)}$ in a solution. We have again used that $P_{c_l} \geq \frac{1}{2} P_0$ (see Corollary 3.10). Since an optimal solution that uses $n' \leq n_{L,j}$ of these items will obviously use the n' items of smallest size, only the smallest $n_{L,j}$ items have to be kept.

Remember that every interval $\tilde{L}^{(k,b)}$ has only $\mathcal{O}(\frac{2}{\varepsilon})$ different values q_j and that there are only $\mathcal{O}(\log(\frac{1}{\varepsilon} c_{\max}^{(b)} / c_{\min}^{(b)}))$ intervals $L^{(k,b)}$, which is also the number of intervals $\tilde{L}^{(k,b)}$. Thus, we get the following bound on the number of the large items:

$$n_L \leq \sum_{\tilde{L}^{(k,b)}} \sum_{q_j \in \tilde{L}^{(k,b)}} n_{L,j} \leq \sum_{\tilde{L}^{(k,b)}} \mathcal{O}\left(\frac{2}{\varepsilon} \cdot \frac{2^{-k}}{\varepsilon} \frac{c_{\max}^{(b)}}{c_{\min}^{(b)}} \right) \subseteq \mathcal{O}\left(\frac{1}{\varepsilon^2} \frac{c_{\max}^{(b)}}{c_{\min}^{(b)}} \right) \quad .$$

Lawler [63] has presented a technique to obtain these items: create a bucket for every scaled profit q_j and place every large item in the bucket with the corresponding scaled profit. The right bucket for one item a, i.e. the value $q(a) = q_j$, can be determined in $\mathcal{O}(1)$: the exponent $k(a)$ can be computed in time $\mathcal{O}(1)$ because we assume that the logarithm can be determined in $\mathcal{O}(1)$. All other operations to compute $q(a)$ then also need time in $\mathcal{O}(1)$. The $n_{L,j}$ items of smallest size in one bucket can be found in linear time of the bucket size with a median-finding routine similar to the one of Appendix B. In total, we need time and space in $\mathcal{O}(n + \frac{2}{\varepsilon} \log(\frac{1}{\varepsilon} c_{\max}^{(b)} / c_{\min}^{(b)}))$ for the entire procedure.

(This proof is a slight modification of the proof by Lawler [63].) □

Let $\tilde{a}_1, \ldots, \tilde{a}_r$ be items that have the same scaled profit q_j and a non-scaled profit in $L^{(k,b)}$. As seen above, we may assume $r \leq n_{L,j} \in \mathcal{O}(\frac{2^{-k}}{\varepsilon} c_{\max}^{(b)} / c_{\min}^{(b)})$ because of Bound (3.28), and these r items are the smallest items with the scaled profit q_j. Assume without loss of generality that the items are sorted according to their size such that $s(\tilde{a}_1) \leq \ldots \leq s(\tilde{a}_r)$.

Let us change the large-item computation: $F_j(i)$ stands for the smallest size to get the profit i with items of profit in $q_1, \ldots, q_j$. The recursion becomes

$$\begin{aligned} F_j(i) = \min\{ & F_{j-1}(i), F_{j-1}(i - q_j) + s(\tilde{a}_1), \ldots, \\ & F_{j-1}(i - r' \cdot q_j) + s(\tilde{a}_1) + \cdots + s(\tilde{a}_{r'}), \ldots, \\ & F_{j-1}(i - r \cdot q_j) + s(\tilde{a}_1) + \cdots + s(\tilde{a}_r) \} \quad , \end{aligned}$$

which only needs space in $\mathcal{O}(1)$ and not more time than before. For backtracking, it must be stored how many items of profit q_j are added to obtain $F_j(i)$. The correctness is obvious: if $r' \leq r$ items are used, the solution will use the smallest r' items.

This new recursion allows us to improve the space bound for the large-item computation.

Lemma 3.21. *Redefine the $F_j(i)$ as above. By using this definition of the large-item computation over the scaled profits, we need space in $\mathcal{O}(\frac{1}{\varepsilon^3} c_{\max}^{(b)}/c_{\min}^{(b)})$ to find and store the $F_j(i)$ of one C_b. The running time is in $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)} n_L) \subseteq \mathcal{O}(\frac{1}{\varepsilon^4}(c_{\max}^{(b)}/c_{\min}^{(b)})^2)$. Note that the items have to be sorted according to their size $s(a)$ to determine the $F_j(i)$ and that we have $j \in \{1, \ldots, m_b\}$, where m_b is the number of different scaled profits (see Lemma 3.20).*

Proof. Let k be such that $\overline{\mathrm{OPT}}_{c_l}(I) \in L^{(k,b)}$, i.e. k is the largest k' that occurs for the item intervals $L^{(k',b)} = (2^{k'}T_b, 2^{k'+1}T_b]$. Since $\overline{\mathrm{OPT}}_{c_l}(I) \leq 2c_{\max}^{(b)} P_0$ holds (see the Bound (3.25)), we have

$$k \in \mathcal{O}\left(\log\left(\frac{8}{\varepsilon}\frac{c_{\max}^{(b)}}{c_{\min}^{(b)}}\right)\right) \quad . \tag{3.29}$$

Inequality (3.26) shows that the maximum profit for the scaled items that has to be considered is $\overline{\mathrm{OPT}}_{c_{\max}^{(b)}}(I)/K_b \leq \frac{16}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)}$. On the other hand, the items a with $p(a) \in L^{(k,b)}$, i.e. the items of largest profits, have scaled profits $q(a) = \lfloor \frac{p(a)}{2^k K_b} \rfloor 2^k \in \{r \cdot 2^k \mid r \in \mathbb{N}\}$.

Start the calculation of the $F_j(i)$ with these items. Because of $q(a) \in \{r \cdot 2^k \mid r \in \mathbb{N}\}$, only scaled profits $i = s \cdot 2^k$, $s \in \mathbb{N}$, may have $F_j(i) < \infty$, all other entries must have $F_j(i) = \infty$. It is therefore sufficient to calculate and save the $F_j(i)$ for $i = s \cdot 2^k$. There are at most

$$\mathcal{O}\left(\frac{\overline{\mathrm{OPT}}_{c_{\max}^{(b)}}(I)/K_b}{2^k}\right) \subseteq \mathcal{O}\left(\frac{16}{\varepsilon^2}\frac{c_{\max}^{(b)}}{c_{\min}^{(b)}}\frac{1}{2^k}\right) \overset{(3.29)}{\subseteq} \mathcal{O}\left(\frac{16}{\varepsilon^2}\cdot\frac{\varepsilon}{8}\right) = \mathcal{O}\left(\frac{2}{\varepsilon}\right)$$

values $i = s \cdot 2^k$ in $\{0, 1, \ldots, \overline{\mathrm{OPT}}_{c_{\max}^{(b)}}(I)/K_b\}$, and only $\mathcal{O}(\frac{2}{\varepsilon})$ different profits $q_{j'} \in \tilde{L}^{(k,b)}$ as seen in the proof of Lemma 3.20. Thus, we only have to save $\mathcal{O}(\frac{2}{\varepsilon}) \cdot \mathcal{O}(\frac{2}{\varepsilon}) = \mathcal{O}(\frac{4}{\varepsilon^2})$ table entries $F_j(i)$ for all $q_{j'} \in \tilde{L}^{(k,b)}$.

We continue the large-item computation for items in $\tilde{L}^{(k',b)}$ from the largest to the smallest $k' \in \{k-1, k-2, \ldots, 0\}$. The number of scaled profits $q_{j'}$ remains $\mathcal{O}(\frac{2}{\varepsilon})$ (see the proof of Lemma 3.20), whereas the number of profits i to be considered doubles from $k'+1$ to k' because the items now have profits $r \cdot 2^{k'}$ and the table entries $F_j(i) < \infty$ therefore have scaled profits $i = s \cdot 2^{k'}$, i.e. $\mathcal{O}(\frac{2}{\varepsilon}) \cdot \mathcal{O}(\frac{2}{\varepsilon} 2^{k-k'}) = \mathcal{O}(\frac{4}{\varepsilon^2} 2^{k-k'})$ entries have to be saved. (Note that the scaled profits $i = s \cdot 2^{k''}$ for $k'' > k'$ are a subset of the entries $i = s \cdot 2^{k'}$.)

Hence, we only need space in

$$\sum_{k'=0}^{k} \mathcal{O}\left(\frac{4}{\varepsilon^2}2^{k-k'}\right) = \mathcal{O}\left(\frac{4}{\varepsilon^2}2^{k}\right) \overset{(3.29)}{=} \mathcal{O}\left(\frac{4}{\varepsilon^2}\cdot\frac{8}{\varepsilon}\frac{c_{\max}^{(b)}}{c_{\min}^{(b)}}\right) = \mathcal{O}\left(\frac{1}{\varepsilon^3}\frac{c_{\max}^{(b)}}{c_{\min}^{(b)}}\right)$$

to compute and save all $F_j(i) < \infty$ if we determine the $F_j(i)$ from the largest to the smallest $k' \in \{k, k-1, \ldots, 0\}$.

To sum up, all profits $i \in \{0, \ldots, i_{\max}\}$, i.e. all values $F_j(i)$, have to be considered only for $k' = 0$. Since there are only $\mathcal{O}(\frac{2}{\varepsilon})$ values $q_{j'}$ for $k' = 0$, we get $\mathcal{O}(\frac{2}{\varepsilon}i_{\max}) = \mathcal{O}(\frac{2}{\varepsilon}\frac{16}{\varepsilon^2}c_{\max}^{(b)}/c_{\min}^{(b)}) = \mathcal{O}(\frac{1}{\varepsilon^3}c_{\max}^{(b)}/c_{\min}^{(b)})$. For $k' \in \{1, 2, \ldots\}$, the number of the scaled profits $q_{j'}$ stays the same, but only half as many profits i as for the preceding k' have to be considered, which yields the bound above.

The asymptotic running time for determining the table of the $F_j(i)$ does not change. Similar to Theorem 3.15, it is still dominated for one C_b by

$$\mathcal{O}\left(\frac{1}{\varepsilon^2}\frac{c_{\max}^{(b)}}{c_{\min}^{(b)}}n_b\right) = \mathcal{O}\left(\frac{1}{\varepsilon^4}\left(\frac{c_{\max}^{(b)}}{c_{\min}^{(b)}}\right)^2\right) \quad ,$$

where we have used $n_b \leq n_L \in \mathcal{O}(\frac{1}{\varepsilon^2}c_{\max}^{(b)}/c_{\min}^{(b)})$ as seen in Lemma 3.20. The time to sort all n_b large items according to their size (which is needed for the modified computation) is dominated by the dynamic program because only $n_b \leq n_L \in \mathcal{O}(\frac{1}{\varepsilon^2}c_{\max}^{(b)}/c_{\min}^{(b)})$ large items have to be sorted. □

Lemma 3.20 and 3.21 can now be applied to the basic FPTAS for 0-1 KPIP. Algorithm 3.4 presents the new method.

Lemma 3.22. *The improved FPTAS needs time in*

$$\mathcal{O}\left(\min\left\{\left(\left\lfloor\frac{1}{w}\right\rfloor+1\right)\left(\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}\right)^2, M\right\}\cdot\frac{1}{\varepsilon^4} + M\cdot\left(\frac{1}{\varepsilon^2}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}} + n\cdot\log\frac{1}{\varepsilon}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}\right)\right)$$

and space in $\mathcal{O}(\frac{1}{\varepsilon^3}c_{\max}^{(b_0)}/c_{\min}^{(b_0)} + M + n)$.

Proof. Lemma 3.17 still holds: for one $C_b \neq \emptyset$, the inner for-loop (Steps 8–12) needs time in $\mathcal{O}(|C_b| \cdot (\frac{1}{\varepsilon^2}c_{\max}^{(b)}/c_{\min}^{(b)} + n \cdot \log(\frac{1}{\varepsilon}c_{\max}^{(b)}/c_{\min}^{(b)})))$ and space in $\mathcal{O}(\frac{1}{\varepsilon^2}c_{\max}^{(b)}/c_{\min}^{(b)} + n)$. The main difference is that the backtracking for the large items needs $\mathcal{O}(m_b + n_b) \subseteq \mathcal{O}(m_b + n_L) \subseteq \mathcal{O}(\frac{1}{\varepsilon^2}c_{\max}^{(b_0)}/c_{\min}^{(b_0)})$, which however does not change the time and space complexity.

Let us now consider the outer for-loop (Steps 4–12). We will derive the running time for the Steps 6 and 7 over an iteration of one $C_b \neq \emptyset$.

Algorithm 3.4: The improved algorithm for 0-1 KPIP

Input: Item set I, sorted knapsacks $C = \{c_1, \ldots, c_M\}$

Output: Profit P, solution set V and knapsack size c_{sol}

1 Determine the sets S_l, the approximations $\bar{P}_{c_l}$ and P_0 and adapt C accordingly (Algorithm 3.2);

2 Partition C into the sets C_b and determine at the same time $c_{\min}^{(b)}$ and $c_{\max}^{(b)}$ for all C_b;

3 Set $P := 0$, $c_{\text{sol}} := \emptyset$, and $V := \emptyset$;

4 **for** *all* $C_b \neq \emptyset$ **do**

5 Define T_b and K_b;

6 Partition the items in every S_l for $c_l \leq c_{\max}^{(b)}$ into large and small items. Reduce the number of the large items as seen in Lemma 3.20 to have only n_b of them. Scale the large items and sort them according to their size;

7 Calculate the $F_j(i)$ with the new method of Lemma 3.21. Discard dominated $F_{m_b}(i)$;

8 **for** *all* $c_l \in C_b$ **do**

9 Find $\bar{P}_l^{(1)} = \max_{F_{m_b}(i) \leq c_l} K_b \cdot i + \phi_l(c_l - F_{m_b}(i))$ where the values of ϕ_l are determined as stated in Theorem 3.16;

10 **if** $\frac{1}{c_l}\bar{P}_l^{(1)} > P_1$ **then**

11 Set $P := \frac{\bar{P}_l^{(1)}}{c_l}$;

12 Determine the item set V_t for solution $\bar{P}_l^{(1)}$ and set $V := V_t$ as well as $c_{\text{sol}} := c_l$;

13 **return** *P, V and c_{sol};*

Step 6 still needs time in $\mathcal{O}(n)$ and space in $\mathcal{O}(n)$ to group the items into large and small. Additionally, it determines the $n_{L,j}$ items for every scaled item profit q_j in time $\mathcal{O}(n + \frac{2}{\varepsilon}\log \frac{1}{\varepsilon} c_{\max}^{(b_0)}/c_{\min}^{(b_0)})$ as seen in Lemma 3.20. Moreover, the large items are sorted according to their size for the space-saving calculation of the table entries $F_j(i)$ presented in Lemma 3.21. This increases the running time by $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b_0)}/c_{\min}^{(b_0)} \log(\frac{1}{\varepsilon^2} c_{\max}^{(b_0)}/c_{\min}^{(b_0)}))$. The space complexity of Step 6 is now bounded by $\mathcal{O}(n_L + n + \frac{2}{\varepsilon}\log(\frac{1}{\varepsilon} c_{\max}^{(b_0)}/c_{\min}^{(b_0)})) = \mathcal{O}(n + \frac{2}{\varepsilon}\log(\frac{1}{\varepsilon} c_{\max}^{(b_0)}/c_{\min}^{(b_0)}))$, which also includes the space to sort the $\mathcal{O}(n_L)$ large items.

The running time of Step 7 (together with discarding dominated $F_{m_b}(i)$) decreases to $\mathcal{O}((c_{\max}^{(b_0)}/c_{\min}^{(b_0)})^2 \frac{1}{\varepsilon^4})$ and only needs space in $\mathcal{O}(\frac{1}{\varepsilon^3} c_{\max}^{(b_0)}/c_{\min}^{(b_0)})$ as seen in Lemma 3.21.

Thus, the running time of the Steps 6 and 7 now is in $\mathcal{O}(n + (c_{\max}^{(b_0)}/c_{\min}^{(b_0)})^2 \frac{1}{\varepsilon^4})$ for one iteration of the outer for-loop, and the space needed is in $\mathcal{O}(n + \frac{1}{\varepsilon^3} c_{\max}^{(b_0)}/c_{\min}^{(b_0)})$. The overall time complexity in

$$\mathcal{O}\left(\left(\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}\right)^2 \cdot \frac{1}{\varepsilon^4} + |C_b| \cdot \left(\frac{1}{\varepsilon^2}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}} + n \cdot \log\frac{1}{\varepsilon}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}\right)\right)$$

and space complexity in $\mathcal{O}(\frac{1}{\varepsilon^3} c_{\max}^{(b_0)}/c_{\min}^{(b_0)} + n)$ follow for one iteration of the outer for-loop similar to the proof of Lemma 3.18.

The new running time and space bound can now be derived like in the proof of Theorem 3.19. □

w has still to be chosen so that the running time is minimized. Note that $c_{\max}^{(b_0)}/c_{\min}^{(b_0)} \leq \frac{1}{c_{\min}^w}$ and that $c_{\min} \leq c_{\min}^w \leq 1$ so that $w \leq 1$, i.e. $\lfloor \frac{1}{w} \rfloor + 1 \leq \frac{2}{w}$. For simplicity, we minimize the expression $(\lfloor \frac{1}{w} \rfloor + 1)(c_{\max}^{(b_0)}/c_{\min}^{(b_0)})^2 \frac{1}{\varepsilon^4}$ by minimizing the dominating expression $\frac{2}{w}\frac{1}{\varepsilon^4 c_{\min}^{2w}}$. A short calculation shows that the minimum is attained at $w = -\frac{1}{2\ln c_{\min}}$. Since $c_{\min}^{2w}|_{w=-\frac{1}{2\ln c_{\min}}} = e^{-\frac{2\ln c_{\min}}{2\ln c_{\min}}} = e^{-1} \in \mathcal{O}(1)$, we get the following result:

Theorem 3.23. *0-1 KPIP can be solved in time*

$$\mathcal{O}\left(\min\left\{M, \left\lfloor 2\log\frac{1}{c_{\min}}\right\rfloor + 1\right\}\frac{1}{\varepsilon^4} + M \cdot \left(\frac{1}{\varepsilon^2} + n\log\frac{1}{\varepsilon}\right)\right)$$

and in space $\mathcal{O}(\frac{1}{\varepsilon^3} + M + n)$.

Remark 3.24. By setting $w = -\frac{1}{2\ln c_{\min}}$, the knapsack intervals C_b are contained in $(e^{-\frac{1}{2}(b+1)}, e^{-\frac{1}{2}b}]$. We can of course also set e.g. $w = -\frac{1}{\log_2(c_{\min})}$ instead. Then we have $C_b \subset (2^{-(b+1)}, 2^{-b}]$, which is easier to implement and yields the same asymptotic running time and space bound.

3.6 Variants of KPIP

3.6.1 The Unbounded KPIP

Here, it is allowed to take an arbitrary number of copies of each item $a \in I$. The unboundedness allows to improve the running time and space complexity of the algorithm. The ideas below to adapt our FPTAS for 0-1 KPIP to UKPIP are again taken from [63], and the modified FPTAS is presented in Algorithm 3.5. Note that some ideas are re-used in Chapter 5 so that they are formally presented there.

First of all, the greedy procedures are easier to do. For the first approximations $\bar{P}_{c_l}$, we only need the most efficient item $a^{(l)}_{\text{meff}}$ in $\bar{S}_l$. As remarked in [63, 61, p. 232], the greedy procedure can fill at least half of c_l with $a^{(l)}_{\text{meff}}$, i.e. the item of largest profit does not need to be considered. A proof similar to Theorem 3.9 shows that we have $\frac{1}{2}\overline{\text{OPT}}_{c_l} \leq \bar{P}_{c_l} = \lfloor \frac{c_l}{s(a^{(l)}_{\text{meff}})} \rfloor \cdot p(a^{(l)}_{\text{meff}})$. This will be formally proved in Theorem 5.4. Similarly, we only need the most efficient small item $a^{(l)}_{\text{eff}}$ for c_l when we determine the $\phi_l(c_l - F_{n_b}(i))$ in $\max_{F_{n_b}(i) \leq c_l} K_b \cdot i + \phi_l(c_l - F_{n_b}(i))$.

Fix $C_b \neq \emptyset$. As we are in the unbounded case, it is obviously sufficient to keep only the smallest item $\tilde{a}_j$ for each scaled profit q_j. Then, we copy these large items for the dynamic program. One possibility is to take $n_{L,j}$ copies of each item $\tilde{a}_j$. However, it is not the most efficient way: instead of $n_{L,j}$ items, we create items $\tilde{a}^{(r)}_j$ of profit $2^r p(\tilde{a}_j)$ and size $2^r s(\tilde{a}_j)$ for $r \in \{0, \ldots, \lfloor \log_2(n_{L,j}) \rfloor\}$ as done in [63]. Note that the item $\tilde{a}^{(r)}_j$ has also the scaled profit $2^r q_j$. Obviously, these copies are sufficient to represent any choice of large items in a feasible solution of UKPIP. However, there may be items $\tilde{a}^{(r_1)}_{j_1}, \tilde{a}^{(r_2)}_{j_2}$ with the same scaled profit q_j. For every scaled profit q_j, we again have only to keep the item copy of smallest size because of the following lemma:

Lemma 3.25. *Let q_j be a fixed scaled profit. In UKPIP, it is sufficient to have only one item $\tilde{a}^{(r)}_{j_1}$ of smallest size with $q(\tilde{a}^{(r)}_{j_1}) = q_j$.*

Proof. Let $\tilde{a}^{(r_1)}_{j_1}$ and $\tilde{a}^{(r_2)}_{j_2}$ be two item copies with the same scaled profit q_j. Assume without loss of generality that $s(\tilde{a}^{(r_1)}_{j_1}) \leq s(\tilde{a}^{(r_2)}_{j_2})$ and that both items are used in a solution. Replace $\tilde{a}^{(r_2)}_{j_2}$ by a second copy of $\tilde{a}^{(r_1)}_{j_1}$. These two copies of $\tilde{a}^{(r_1)}_{j_1}$ with the profit $q_j + q_j = 2q_j$ can again be replaced by $\tilde{a}^{(r_1+1)}_{j_1}$ with the profit $2q_j$. By iterating, we get a new solution of the same profit as the old one and which may have an even smaller total size, but only one item of every scaled profit q_j. Thus, it is sufficient to consider one item of smallest size for every scaled profit. □

Note that we therefore have (at most) one item for every scaled profit in the end, i.e. $n_b \leq m_b$ holds.

Finally, we can take advantage of the unboundedness for the space complexity [63]: the table entries we save are of the form $(F_j(i), j')$, where j' is the index of the scaled profit $q_{j'}$ who was used to form the entry $F_j(i)$, i.e. the index where $F_j(i)$ changed for the last time so that $F_j(i) = F_{j'}(i) = F_{j'-1}(i - q_{j'}) + s(\bar{a}_{j'})$ holds. When the entries $(F_j(i), j')$ have been determined, the old entries $(F_{j-1}(i), j'')$ can be discarded so that we only have the entries $(F_{n_b}(i), j)$ at the end. These are sufficient for backtracking: if $F_{n_b}(i)$ has been obtained by adding the item with the scaled profit $q_{j'}$, this item is part of the solution, and we continue the backtracking with the entry $(F_{n_b}(i - q_{j'}), j'')$. It is of course possible that one item $\bar{a}_{j'}$ is used several times in the resulting solution, which is not a problem because of the unboundedness.

Algorithm 3.5 presents the entire FPTAS for UKPIP. We will derive the time and space complexity.

Lemma 3.26. *Fix one $C_b \neq \emptyset$ and assume that the most efficient small item $a_{\text{eff}}^{(l)}$ is already known for each $c_l \in C_b$. The inner for-loop of Algorithm 3.5 (Steps 10–14) needs for all $c_l \in C_b$ time in $\mathcal{O}(|C_b| \cdot \frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)})$ and space in $\mathcal{O}(1)$.*

Proof. As already explained, $\phi_l(c_l - F_{n_b}(i))$ can be determined in $\mathcal{O}(1)$ simply by computing $p(a_{\text{eff}}^{(l)}) \cdot \lfloor \frac{c_l - F_{n_b}(i)}{s(a_{\text{eff}}^{(l)})} \rfloor$. As the $a_{\text{eff}}^{(l)}$ are already known, we only need time in $\mathcal{O}(|C_b| \cdot i_{\max}) = \mathcal{O}(|C_b| \frac{1}{\varepsilon^2} c_{\max}^{(b)}/c_{\min}^{(b)})$ for the $\bar{P}_{c_l}$ (see the Bound (3.26)). In each iteration of the inner for-loop, we only have to store the new values of P_1, c_{sol}, NewBestValueFound as well as the saved tuple $(F_{n_b}(i), j')$, and we can discard the old values. Hence, the space is bounded by $\mathcal{O}(1)$ over all $c_l \in C_b$. □

Lemma 3.27. *The outer for-loop of Algorithm 3.5 (Steps 4–16) needs time in $\mathcal{O}(\frac{M}{\varepsilon^2} c_{\max}^{(b_0)}/c_{\min}^{(b_0)} + \min\{\lfloor \frac{1}{w} \rfloor + 1, M\} \frac{1}{\varepsilon^3} c_{\max}^{(b_0)}/c_{\min}^{(b_0)} + \min\{\lfloor \frac{1}{w} \rfloor + 1, M\} n)$ and space in $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b_0)}/c_{\min}^{(b_0)} + M + n)$ over all $C_b \neq \emptyset$.*

Proof. Fix one $C_b \neq \emptyset$. Obviously, Steps 5 and 6 need time and space in $\mathcal{O}(1)$.

In Step 7, the S_l can be partitioned into large and small items in time and space $\mathcal{O}(n)$. We use Remark 3.13 so that we only have at most n sets S_l. The most efficient small item $a_{\text{eff}}^{(l)}$ for every $c_l \in C_b$ can be found by a single scan of the small items: $a_{\text{eff}}^{(l)}$ is either $a_{\text{eff}}^{(l-1)}$ or a small item in S_l. We therefore additionally need time and space in $\mathcal{O}(n + |C_b|)$. Saving the $a_{\text{eff}}^{(l)}$ needs space in $\mathcal{O}(|C_b|)$. Algorithm 5.3 shows a possible implementation, where time and space complexity are proved in Theorem 5.11.

Step 8 preprocesses the large items. The item $\tilde{a}_j$ of smallest size for every scaled profit q_j can be found by creating a bucket for every scaled profit, adding every item to its corresponding bucket and taking the item of smallest size from every bucket. Together with the creation of the item copies $\tilde{a}_j^{(r)}$, this can be done in time

Algorithm 3.5: The algorithm for UKPIP

Input: Item set I, sorted knapsacks $C = \{c_1, \ldots, c_M\}$

Output: Profit P, solution set V and knapsack size c_{sol}

Determine $a_{\text{meff}}^{(l)}$, the sets S_l, the approximations $\bar{P}_{c_l}$ and P_0 and adapt C accordingly (Algorithm 5.1);

Partition C into the sets C_b and determine the $c_{\min}^{(b)}$ and $c_{\max}^{(b)}$ similar to Algorithm 5.2;

Let $P_1 := 0$, $V := \emptyset$ and $c_{\text{sol}} := \emptyset$;

for *all* $C_b \neq \emptyset$ **do**

  Set NewBestValueFound := *false*;

  Compute T_b and K_b;

  Partition the items in every S_l for $c_l \leq c_{\max}^{(b)}$ into large and small items and save the most efficient small item $a_{\text{eff}}^{(l)}$ for every $c_l \in C_b$ (see Algorithm 5.3);

  Take for every scaled profit q_j a large item $\tilde{a}_j$ with $q(\tilde{a}_j) = q_j$ and of smallest weight. Create the item copies $\tilde{a}_j^{(r)}$ and reduce the items again to have only one item for every scaled profit. Scale these large items;

  Calculate the $(F_j(i), j')$. Discard dominated $(F_{n_b}(i), j')$;

  for *every* $c_l \in C_b$ **do**

    Find $\bar{P}_l^{(1)} = \max_{F_{n_b}(i) \leq c_l} K_b \cdot i + \phi_l(c_l - F_{n_b}(i))$;

    if $\frac{1}{c_l}\bar{P}_l^{(1)} > P_1$ **then**

      Save the tuple $(F_{n_b}(i), j')$ for $\bar{P}_l^{(1)}$;

      Set $P_1 := \frac{1}{c_l}\bar{P}_l^{(1)}$, $c_{\text{sol}} := c_l$ and NewBestValueFound := *true*;

  if NewBestValueFound = *true* **then**

    Determine the item set V for solution $\bar{P}_l^{(1)}$ by backtracking (for the large items) and by taking $\lfloor \frac{c_{\text{sol}} - F_{n_b}(i)}{s(a_{\text{eff}}^{(l)})} \rfloor$ copies of $a_{\text{eff}}^{(l)}$;

return P_1, V *and* c_{sol};

and space $\mathcal{O}(n + \frac{2}{\varepsilon}\log^2(\frac{1}{\varepsilon}c_{\max}^{(b)}/c_{\min}^{(b)}))$, and we have $\mathcal{O}(\frac{2}{\varepsilon}\log^2(\frac{1}{\varepsilon}c_{\max}^{(b)}/c_{\min}^{(b)}))$ item copies in total. For these bounds, we use that $m_b \in \mathcal{O}(\frac{2}{\varepsilon}\cdot\log(\frac{1}{\varepsilon}c_{\max}^{(b)}/c_{\min}^{(b)}))$ (Lemma 3.20) and the generous bound $n_{L,j} \leq \mathcal{O}(\frac{1}{\varepsilon}c_{\max}^{(b)}/c_{\min}^{(b)})$ (see Bound (3.28)) so that each $\tilde{a}_j$ yields $\mathcal{O}(\log(\frac{1}{\varepsilon}c_{\max}^{(b_0)}/c_{\min}^{(b_0)}))$ item copies $\tilde{a}_j^{(r)}$. Moreover, we assume that the scaled profit of an item can be determined in $\mathcal{O}(1)$ (see the proof of Lemma 3.20). After having created the item copies, they are reduced again to have only one item for every scaled profit. This can be done in time and space $\mathcal{O}(\frac{2}{\varepsilon}\log^2(\frac{1}{\varepsilon}c_{\max}^{(b)}/c_{\min}^{(b)}))$ by the bucket procedure.

Thus, Step 8 has a running time in $\mathcal{O}(\frac{1}{\varepsilon}\log^2(\frac{1}{\varepsilon}c_{\max}^{(b)}/c_{\min}^{(b)}) + n)$ and needs space in $\mathcal{O}(\frac{1}{\varepsilon}\log^2(\frac{1}{\varepsilon}c_{\max}^{(b)}/c_{\min}^{(b)}) + n)$. We only have one item for every scaled profit q_j with $n_b \leq m_b \in \mathcal{O}(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}c_{\max}^{(b)}/c_{\min}^{(b)}))$.

The dynamic program is executed in Step 9. The space for Step 9, i.e. the space to store the pairs $(F_j(i), j')$ and finally $(F_{n_b}(i), j')$, is always in $\mathcal{O}(i_{\max}) = \mathcal{O}(\frac{1}{\varepsilon^2}c_{\max}^{(b)}/c_{\min}^{(b)})$ (see Bound (3.23)) because the dynamic program only saves the consecutive entries $(F_{j-1}(i), j')$ and $(F_j(i), j')$ for $i \in \{0, \ldots, i_{\max}\}$. As we only have one item $\bar{a}_{j'}$ for every scaled profit $q_{j'}$, the running time is equal to the space bound of Lemma 3.21, i.e. in $\mathcal{O}(\frac{1}{\varepsilon^3}c_{\max}^{(b)}/c_{\min}^{(b)})$.

The time and space complexity of the inner for-loop are stated in Lemma 3.26.

Finally, the time needed for the new backtracking in Step 16 can be bounded by $\mathcal{O}(i_{\max}) = \mathcal{O}(\frac{1}{\varepsilon^2}c_{\max}^{(b)}/c_{\min}^{(b)})$, while the number of item copies of $a_{\text{eff}}^{(l)}$ to be taken can be re-computed in $\mathcal{O}(1)$ if necessary. The space required is obviously in $\mathcal{O}(\frac{1}{\varepsilon^2}c_{\max}^{(b)}/c_{\min}^{(b)})$.

To sum up, the outer for-loop needs over all $C_b \neq \emptyset$ time in $\mathcal{O}(\frac{M}{\varepsilon^2}c_{\max}^{(b_0)}/c_{\min}^{(b_0)} + \min\{\lfloor\frac{1}{w}\rfloor + 1, M\}\frac{1}{\varepsilon^3}c_{\max}^{(b_0)}/c_{\min}^{(b_0)} + \min\{\lfloor\frac{1}{w}\rfloor + 1, M\}n)$. Since new values (except of P_1, c_{sol} and V) can be discarded after one execution of the for-loop, we have a space complexity in $\mathcal{O}(\frac{1}{\varepsilon^2}c_{\max}^{(b_0)}/c_{\min}^{(b_0)} + M + n)$ over all iterations of the algorithm. □

We get the overall running time and space bound.

Lemma 3.28. *UKPIP can be solved in time*

$$\mathcal{O}\left(n\log M + \min\left\{\left\lfloor\frac{1}{w}\right\rfloor + 1, M\right\}n + \frac{M}{\varepsilon^2}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}} + \min\left\{\left(\left\lfloor\frac{1}{w}\right\rfloor + 1\right)\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}, M\right\}\frac{1}{\varepsilon^3}\right)$$

and space $\mathcal{O}(\frac{1}{\varepsilon^2}c_{\max}^{(b_0)}/c_{\min}^{(b_0)} + M + n)$.

Proof. Step 1 is similar to the corresponding Step 1 in Algorithm 3.4. The main difference to Algorithm 3.2 is that we find the most efficient item $a_{\text{meff}}^{(l)}$ for each c_l instead of $a_{\max}^{(l)}$. Then, each $\bar{P}_{c_l}$ can be determined in $\mathcal{O}(1)$ as explained above. A

proper implementation of the modified Step 1 is shown in Algorithm 5.1, and we need time in $\mathcal{O}(M + n \log M)$ and space in $\mathcal{O}(M + n)$ (see the proof of Lemma 3.12 or Theorem 5.7). We can assume that we have only $\mathcal{O}(n)$ sets S_l (see Remark 3.13 and Remark 5.8).

Step 2 has not changed either and can be done in time and space $\mathcal{O}(M)$ (see the proof of Theorem 3.19 as well as Algorithm 5.2 with Theorem 5.10).

The lemma now follows with Lemma 3.27. □

Similar to Theorem 3.23, we get $w = -\frac{1}{\ln c_{\min}}$ and Theorem 3.1. Note that we can also set $w = -\frac{1}{\log_2(c_{\min})}$ like in Remark 3.24.

3.6.2 The Bounded KPIP

Contrary to UKPIP, only a bounded number $d_j \in \mathbb{N}$ of copies of every item a_j, $j \in \{1, \ldots, n\}$, can be taken. Plotkin, Shmoys, and Tardos [72, pp. 295–297] explain how the 0-1 FPTAS by Lawler [63] can be transformed into one for the Bounded Knapsack Problem. We adapt their reasoning and therefore slightly modify our 0-1 KPIP FPTAS for the Bounded KPIP (BKPIP).

The median-based divide-and-conquer strategy can easily be adapted to still run in time linear in the number of item sizes that are considered. It is used to get the values of $\bar{P}_{c_l}$ and P_0 in Step 1 of the 0-1 FPTAS (Algorithm 3.4), the subset of large items in Step 6 as well as the small-item values $\phi_l(c_l - F_{n_b}(i))$ in Step 9 together with the small items in V_t in Step 12. As proposed in [72, p. 296], item copies of the large items can be taken similar to UKPIP. However, the additional modifications for UKPIP are not possible, i.e. the running time and space bound are identical to the improved 0-1 KPIP algorithm.

Details can be found in Appendix C.

Theorem 3.29. *BKPIP can be solved in time*

$$\mathcal{O}\left(\min\left\{\left(\left\lfloor\frac{1}{w}\right\rfloor+1\right)\left(\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}\right)^2, M\right\}\cdot\frac{1}{\varepsilon^4}+M\cdot\left(\frac{1}{\varepsilon^2}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}+n\cdot\log\frac{1}{\varepsilon}\frac{c_{\max}^{(b_0)}}{c_{\min}^{(b_0)}}\right)\right)$$

and space $\mathcal{O}(\frac{1}{\varepsilon^3}c_{\max}^{(b_0)}/c_{\min}^{(b_0)} + M + n)$. *By choosing* $w = -\frac{1}{2\ln c_{\min}}$, *we obtain the same bounds as in Theorem 3.23 (see also Remark 3.24).*

4 The Unbounded Knapsack Problem

In this chapter, we focus on the unbounded variant of the Knapsack Problem (UKP) where an arbitrary number of copies of every item is allowed, i.e. we want to determine $\max\{\sum_{j=1}^{n} p_j x_j \mid \sum_{j=1}^{n} s_j x_j \leq c; x_j \in \mathbb{N}\ \forall j\}$. Note that the chapter can be read independently of the other knapsack results so that some parts of its content overlap with the other chapters.

4.1 Our Result

We have derived an improved FPTAS for UKP that is faster and needs less space than previously known algorithms.

Theorem 4.1. *There is an FPTAS for UKP with a running time in $\mathcal{O}(n + \frac{1}{\varepsilon^2}\log^3(\frac{1}{\varepsilon}))$ and a space complexity in $\mathcal{O}(n + \frac{1}{\varepsilon}\log^2(\frac{1}{\varepsilon}))$.*

Not only the improved running time, but also the improved space complexity is interesting because "for higher values of $\frac{1}{\varepsilon}$ the space requirement is usually considered to be a more serious bottleneck for practical applications than the running time" [61, p. 168].

Moreover, the new FPTAS can be used as the column generation subroutine of our Bin Packing AFPTAS. Its running time is in $\mathcal{O}(UKP(d_1, \frac{\bar{\varepsilon}}{6}) \cdot \frac{1}{\varepsilon^3}\log\frac{1}{\varepsilon} + \log(\frac{1}{\varepsilon})n)$ for $d_1 \in \mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ and $\bar{\varepsilon} \in \Theta(\varepsilon)$ according to Theorem 2.5, which yields with the new FPTAS the following result:

Theorem 2.2. *There is an AFPTAS $(A_\varepsilon)_{\varepsilon>0}$ for Bin Packing that finds for $\varepsilon \in (0, \frac{1}{2}]$ a packing of I in $A_\varepsilon(I) \leq (1+\varepsilon)\mathrm{OPT}(I) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$ bins. Its running time is in*

$$\mathcal{O}\left(\frac{1}{\varepsilon^5}\log^4\frac{1}{\varepsilon} + \log\left(\frac{1}{\varepsilon}\right)n\right) \quad .$$

Furthermore, Section 4.9 shows how the running time of an AFPTAS for Strip Packing is improved by the FPTAS.

4.2 Overview

Section 4.3 presents the notation used in this chapter. The value $\mathrm{OPT}(I, v)$ is especially important and stands for the optimum profit that can be achieved with the items I in the knapsack volume $v \leq c$.

Section 4.4 shows how the greedy $\frac{1}{2}$ approximation algorithm of Subsection 3.5.2 can be simplified in the unbounded case to get the approximation $P_0 \geq \frac{1}{2}\mathrm{OPT}(I)$. The threshold T and constant K are defined based on P_0.

In Section 4.5, the items are first partitioned into the set of large(-profit) items I_L with $p_j \geq T$ and the set of small(-profit) items I_S with $p_j < T$. The basic idea of the FPTAS is (again) to find the best combination of large and small items: we have $\mathrm{OPT}(I) = \max_{0 \leq v \leq c} \mathrm{OPT}(I_L, v) + \mathrm{OPT}(I_S, c - v)$. The solution values $\mathrm{OPT}(I_L, v)$ for the large items are approximated by dynamic programming and the small items are added greedily to the volume $c - v$. In fact, only the most efficient small item a_{eff} needs to be used for this, i.e. a small item that maximizes $\frac{p_j}{s_j}$. It is also shown that only a subset $I_{L,\mathrm{red}}$ of the large items is necessary for an approximation, which reduces the overall running time and space complexity. The set $I_{L,\mathrm{red}}$ is found by first partitioning the interval of large item profits $[T, 2P_0]$ into $L^{(k)} = [2^k T, 2^{k+1} T)$ for $k \in \{0, \ldots, \kappa + 1\}$ and each $L^{(k)}$ into sub-intervals $L_\gamma^{(k)}$ of length $2^k K$. For $I_{L,\mathrm{red}}$, only the smallest item $a_\gamma^{(k)}$ is kept for every $L_\gamma^{(k)}$. Finally, it is also remarked that one item in $I_{L,\mathrm{red}}$ of profit $2P_0$ already attains the largest possible value such that the final algorithm does not need to continue if it finds such an item. Hence, it can be assumed that $k \leq \kappa$.

Most algorithms for UKP [39, 61, 63] take copies of the items in a reduced item set like $I_{L,\mathrm{red}}$ to transform the UKP instance into a 0-1 KP instance. However, we further preprocess the large items in Section 4.6 by taking advantage of the unboundedness: the items in $I_{L,\mathrm{red}}$ with profits in $L^{(k)}$ are denoted by $I^{(k)}$ for $k \in \{0, \ldots, \kappa\}$. Starting with $I^{(0)} = \tilde{I}^{(0)}$, the items in $\tilde{I}^{(k)}$ are iteratively combined ("glued") together to larger items $\tilde{\tilde{I}}^{(k+1)}$. The next set $\tilde{I}^{(k+1)}$ is then obtained by keeping from $\tilde{\tilde{I}}^{(k+1)} \cup I^{(k+1)}$ only the smallest item $\tilde{a}_\gamma^{(k+1)}$ of each profit sub-interval $L_\gamma^{(k+1)}$. The sets $\tilde{I}^{(k)}$ form the new set $\tilde{I}$, which has approximate *structured* solutions for $k = \kappa$: except of a special case that can be easily checked, at most one large item in $\tilde{I}^{(k)}$ is used for every $k \in \{0, \ldots, \kappa\}$. The value $\mathrm{OPT}_{\leq \kappa}(\tilde{I}, v)$ denotes the optimal profit of these structured solutions for the knapsack volume $v \leq c$. (For the analysis, $\overline{\mathrm{OPT}}_{\leq k_0}(\cdot, v)$ is introduced for structured solutions for $k = k_0$ that use at most one item in $\tilde{I}^{(k,b)}$ for every $k \in \{0, \ldots, k_0\}$, where $k_0 \leq \kappa$.) Then, a large item $a_{\mathrm{eff-c}}$ that consists of several copies of the most efficient small item a_{eff} is constructed. There are now approximate *structured solutions* to the large items $\tilde{I} \cup \{a_{\mathrm{eff-c}}\}$ *with a lower bound* (on the profit). They consist of at most one item of every set $\tilde{I}^{(0)}, \ldots, \tilde{I}^{(\kappa)}, \{a_{\mathrm{eff-c}}\}$ and additionally use at least one item of

profit at least $\frac{1}{4}P_0$. The value $\mathrm{OPT}_{\mathrm{St}}(\tilde{I} \cup \{a_{\mathrm{eff-c}}\}, v)$ denotes the optimal profit with this structure for the knapsack volume $v \leq c$. (In fact, the approximate structured solutions with a lower bound are approximations to the $\mathrm{OPT}(I_L, v)$.)

How can a solution with such a structure for the large items be found for the relevant $v \leq c$? This question is answered in Section 4.7 where approximate dynamic programming is explained. It generates tuples, where a tuple (p, s, k) stands for a set of items with the profit p and size s. Only items in $\bigcup_{k'=k}^{\kappa} \tilde{I}^{(k')} \cup \{a_{\mathrm{eff-c}}\}$ are considered, and the structure above is respected. The dynamic program iteratively generates tuples for $k = \kappa, \ldots, 0$. The number of tuples is bounded by dividing the interval of the possible profits p into sub-intervals $\tilde{L}_{\xi}^{(\kappa-2)}$ and by keeping at most one tuple of smallest size for each sub-interval. As at least one item of profit $\frac{1}{4}P_0$ is used, the profit interval is $[\frac{1}{4}P_0, 2P_0]$. Moreover, the number of large items can be bounded by $\mathcal{O}(\log \frac{1}{\varepsilon})$ in each structured solution. Hence, it is sufficient for the overall approximation ratio that the profit interval is divided into sub-intervals $\tilde{L}_{\xi}^{(\kappa-2)}$ of length $2^{\kappa-2}K$. This considerably reduces the time and space complexity of the dynamic program, and is the reason for the introduction of the structured solutions with a lower bound.

Section 4.8 puts the entire algorithm together. It is shown that the best combination of large items (packed by the dynamic program) and the small item a_{eff} (added greedily), i,.e. $\max_{(p,s,0)} p + \mathrm{OPT}(\{a_{\mathrm{eff}}\}, c - s)$, has a profit of at least $(1 - \varepsilon)\mathrm{OPT}(I)$.

4.3 Notation and Remarks

We use the notation presented in (or close to the one in) Section 3.3. The profit of an item a is denoted by $p(a)$ and its size by $s(a)$. If $a = a_j$, we also write $p(a_j) = p_j$ and $s(a_j) = s_j$. Let $V = \{x_a : a \mid a \in I, x_a \in \mathbb{N}\}$ be a multiset of items, i.e. a subset of items in I with their multiplicities. We naturally define the total profit $p(V) := \sum_{x_a>0} p(a)x_a$ and the total size $s(V) := \sum_{x_a>0} s(a)x_a$.

Let $v \leq c = 1$. The corresponding optimum profit for the volume v is denoted by $\mathrm{OPT}(I, v) = \max\{\sum_{a\in I} p(a)x_a \mid \sum_{a\in I} s(a)x_a \leq v; x_a \in \mathbb{N}\}$. Obviously, $\mathrm{OPT}(I) = \mathrm{OPT}(I, c)$ holds.

Recall that we assume throughout the thesis that basic arithmetic operations as well as computing the logarithm can be performed in $\mathcal{O}(1)$.

Finally, we have a remark about the use of "item" and "item copy" when we consider a solution to a UKP instance.

Remark 4.2. Let $I, \tilde{I}$ be two sets of knapsack items with $\tilde{I} \subseteq I$. In the 0-1 Knapsack Problem, a sentence like "the solution to I uses at most one item in $\tilde{I}$" is obvious: if the solution uses one item in $\tilde{I}$, all other items of the solution are in $I \setminus \tilde{I}$.

Consider now UKP. When we talk about solutions, we would formally have to distinguish between an item $a' \in I$ in the instance and the item copies of a' that a solution $V = \{x_a : a \mid a \in I, x_a \in \mathbb{N}\}$ uses. In this thesis, we however use the expressions "item" and "item copy" interchangeably when talking about solutions. As an example, let us consider the sentence "the solution to I uses at most one item in $\tilde{I}$." It means that the solution contains item copies of items in I, but at most one item copy whose corresponding item is in $\tilde{I}$. To be more precise, the multiset V uses only one item $a \in \tilde{I}$ with a multiplicity $x_a > 0$. We have $x_a \leq 1$, but $x_{a''} = 0$ for all other $a'' \in \tilde{I}$, i.e. $\sum_{a' \in \tilde{I}} x_{a'} \leq 1$. Similarly, "the solution V uses at most $n' \in \mathbb{N}$ items in $\tilde{I}$" means that there are only n' item copies whose corresponding item(s) are in $\tilde{I}$: we have $\sum_{a' \in \tilde{I}} x_{a'} \leq n'$.

The interchangeable use of "item" and "item copy" allows for shorter sentences. Moreover, it is based upon 0-1 KP where "item" and "item copy" are in fact identical.

4.4 A First Approximation

The simple greedy approximation algorithm of Subsection 3.5.2 for $\mathrm{OPT}(I)$ now becomes even easier. Take the most efficient item $a_{\mathrm{meff}} := \arg\max_{a \in I} \frac{p(a)}{s(a)}$. Fill the knapsack with as many copies of a_{meff} as possible, i.e. take $\lfloor \frac{c}{s(a_{\mathrm{meff}})} \rfloor \overset{c=1}{=} \lfloor \frac{1}{s(a_{\mathrm{meff}})} \rfloor$ copies of a_{meff}. Then the following holds:

Theorem 4.3. *We have $P_0 := p(a_{\mathrm{meff}}) \cdot \lfloor \frac{c}{s(a_{\mathrm{meff}})} \rfloor \geq \frac{1}{2}\mathrm{OPT}(I)$. The value P_0 can be found in time $\mathcal{O}(n)$ and space $\mathcal{O}(1)$.*

Proof. Suppose first that a_{meff} can greedily fill the knapsack completely. Then $p(a_{\mathrm{meff}}) \cdot \lfloor \frac{c}{s(a_{\mathrm{meff}})} \rfloor = \mathrm{OPT}(I)$. Otherwise, one additional item a_{meff} exceeds the capacity c. Then $p(a_{\mathrm{meff}}) \cdot \lfloor \frac{c}{s(a_{\mathrm{meff}})} \rfloor + p(a_{\mathrm{meff}}) \geq \mathrm{OPT}(I)$. If $p(a_{\mathrm{meff}}) \leq \frac{1}{2}\mathrm{OPT}(I)$, then $p(a_{\mathrm{meff}}) \cdot \lfloor \frac{c}{s(a_{\mathrm{meff}})} \rfloor \geq \mathrm{OPT}(I) - p(a_{\mathrm{meff}}) \geq \frac{1}{2}\mathrm{OPT}(I)$, and the theorem follows. Otherwise $p(a_{\mathrm{meff}}) \cdot \lfloor \frac{c}{s(a_{\mathrm{meff}})} \rfloor \geq p(a_{\mathrm{meff}}) \geq \frac{1}{2}\mathrm{OPT}(I)$, which also proves the theorem.

To determine P_0, we only have to check all items (which can be done in $\mathcal{O}(n)$) and to save the most efficient item (which only needs time in $\mathcal{O}(1)$).

(The proof is taken from [61, p. 232, 63]) □

Assumption 4.1. From now on, we assume without loss of generality that $\varepsilon \leq \frac{1}{4}$ and $\varepsilon = \frac{1}{2^{\kappa-1}}$ for $\kappa \in \mathbb{N}$. Otherwise, we replace ε by the corresponding $\frac{1}{2^{\kappa-1}}$ such that $\frac{1}{2^{\kappa-1}} \leq \varepsilon < \frac{1}{2^{\kappa-2}}$. Note that $\log_2(\frac{2}{\varepsilon}) = \kappa$ holds.

Similar to Lawler [63] and to Subsection 3.5.3, we introduce the threshold T and a constant K:

$$T := \frac{1}{2}\varepsilon P_0 = \frac{1}{2}\frac{1}{2^{\kappa-1}}P_0 \tag{4.1}$$

and

$$K := \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon})+1}T = \frac{1}{4}\frac{1}{\kappa+1}\frac{1}{2^{\kappa-1}}T = \frac{1}{8}\frac{1}{\kappa+1}\left(\frac{1}{2^{\kappa-1}}\right)^2 P_0 \ . \tag{4.2}$$

We will see later that these values are indeed a good choice for the algorithm. Note that Chapter 5 will additionally derive the corresponding values T_b and K_b similar to Chapter 3, which will justify the choice of T and K in this chapter.

4.5 Reducing the Items

We first partition the items into large(-profit) and small(-profit) items, and only keep the most efficient small item:

$$I_L := \{a \in I \mid p(a) \geq T\}\,,\ I_S := I \setminus I_L,\ \text{ and } a_{\text{eff}} := \arg\max\left\{\frac{p(a)}{s(a)} \;\middle|\; p(a) < T\right\} \ .$$

Theorem 4.4. *The sets I_L, I_S and the item a_{eff} can be found in time $\mathcal{O}(n)$ and space $\mathcal{O}(n)$. This is also the space needed to save I_L.*

Proof. Obvious. □

Similar to Lawler, we now reduce the item set I_L. Note that we have $\text{OPT}(I) \leq 2P_0$ according to Theorem 4.3 so that one item cannot have a profit larger than $\text{OPT}(I) \leq 2P_0$. Hence, the large item profits are in the interval $[T, 2P_0]$. We partition this interval into

$$L^{(k)} := [2^k T, 2^{k+1}T) \ \text{ for } k \in \{0, \ldots, \kappa+1\} \ . \tag{4.3}$$

Note that

$$L^{(\kappa)} = \left[2^\kappa T,\ 2^{\kappa+1}T\right) = \left[2^\kappa \frac{1}{2}\frac{1}{2^{\kappa-1}}P_0,\ 2^{\kappa+1}\frac{1}{2}\frac{1}{2^{\kappa-1}}P_0\right) = [P_0, 2P_0) \ .$$

For convenience, we directly set $L^{(\kappa+1)} := \{2P_0\}$.

We further split the $L^{(k)}$ into disjoint sub-intervals, each of length $2^k K$:

$$L_\gamma^{(k)} := \left[2^k T + \gamma \cdot 2^k K,\ 2^k T + (\gamma+1)2^k K\right) \ \text{ for } \ \gamma \in \left\{0, \ldots, 2^{\kappa+1}(\kappa+1) - 1\right\} \ . \tag{4.4}$$

Note that indeed $L^{(k)} = \bigcup_\gamma L_\gamma^{(k)}$ holds because

$$\begin{aligned} 2^k T + (\gamma+1)2^k K|_{\gamma = 2^{\kappa+1}(\kappa+1)-1} &= 2^k T + 2^{\kappa+1}(\kappa+1)2^k K \\ &\overset{(4.2)}{=} 2^k T + 2^{\kappa+1}(\kappa+1)2^k \frac{1}{4}\frac{1}{\kappa+1}\frac{1}{2^{\kappa-1}}T \\ &= 2^k T + 2^k T = 2^{k+1}T \ . \end{aligned}$$

Similar to above, we set $L_0^{(\kappa+1)} := \{2P_0\}$.

The idea is to keep only the smallest item a for every profit interval $L_\gamma^{(k)}$. We will see that these items are sufficient to determine an approximate solution. Indeed, this is the reasoning used by Lawler [63], which we also employed for our UKPIP FPTAS (see Subsection 3.6.1). The difference is that the profit of an item in $L_\gamma^{(k)}$ was scaled to $q(a) = 2^k \lfloor \frac{p(a)}{2^k K} \rfloor$ (see Subsection 3.5.3) while the original profit $p(a)$ is kept here. Note that the items in one $L_\gamma^{(k)}$ would indeed have the same scaled profit.

Definition 4.5. *For an item a with $p(a) \geq T$, let $k(a) \in \mathbb{N}$ be the interval such that $p(a) \in L^{(k(a))}$ and $\gamma(a) \in \mathbb{N}$ be the sub-interval such that $p(a) \in L_{\gamma(a)}^{(k(a))}$. Let $a_\gamma^{(k)}$ be the smallest item for the profit interval $L_\gamma^{(k)}$, i.e.*

$$a_\gamma^{(k)} := \arg\min \left\{ s(a) \mid a \in I_L \text{ and } p(a) \in L_\gamma^{(k)} \right\} \text{ for all } k \text{ and } \gamma \ .$$

Algorithm 4.1 shows the algorithm to determine the $a_\gamma^{(k)}$. They form the reduced set of large items

$$I_{L,\text{red}} := \bigcup_k \bigcup_\gamma \left\{ a_\gamma^{(k)} \right\} \ .$$

As in [63], we now prove that $I_{L,\text{red}}$ is sufficient for an approximation.

Algorithm 4.1: The algorithm to determine the $a_\gamma^{(k)}$.

```
for k = 0, ..., κ do
    for γ = 0, ..., 2^{κ+1}(κ+1) − 1 do
        a_γ^{(k)} := ∅;
a_0^{(κ+1)} := ∅;
for a ∈ I_L do
    Determine (k(a), γ(a));
    if s(a_{γ(a)}^{(k(a))}) > s(a) or a_{γ(a)}^{(k(a))} = ∅ then
        a_{γ(a)}^{(k(a))} := a;
```

Output: $I_{L,\text{red}} := \bigcup_k \bigcup_\gamma \{a_\gamma^{(k)}\}$

Lemma 4.6. *Let $0 \leq v \leq c = 1$. Then*

$$\text{OPT}\left(\{a_{\text{eff}}\}, c - v\right) \geq \text{OPT}\left(I_S, c - v\right) - T$$

and

$$\text{OPT}\left(I_{L,\text{red}}, v\right) \geq \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right) \text{OPT}\left(I_L, v\right) \ .$$

Proof. For the first inequality, there are two possibilities: either copies of a_{eff} can be taken such that the entire capacity $c - v$ is used. Then obviously $\text{OPT}(\{a_{\text{eff}}\}, c - v) = \text{OPT}(I_S, c - v)$ holds. Otherwise, we have similar to the proof of Theorem 4.3 that $\text{OPT}(\{a_{\text{eff}}\}, c - v) + p(a_{\text{eff}}) = \lfloor \frac{c-v}{s(a_{\text{eff}})} \rfloor \cdot p(a_{\text{eff}}) + p(a_{\text{eff}}) \geq \text{OPT}(I_S, c - v)$. Thus, $\text{OPT}(\{a_{\text{eff}}\}, c - v) \geq \text{OPT}(I_S, c - v) - p(a_{\text{eff}}) \geq \text{OPT}(I_S, c - v) - T$. The first inequality follows.

For the second inequality, take an optimal solution $(x_a)_{a \in I}$ such that $\text{OPT}(I_L, v) = \sum_{a \in I_L} p(a) x_a$. Replace now every item a by its counterpart $a_{\gamma(a)}^{(k(a))}$ in $I_{L,\text{red}}$. Obviously, the solution stays feasible, i.e. the volume v will not be exceeded, because an item may only be replaced by a smaller one. This solution has the total profit $\sum_{a \in I_L} p(a_{\gamma(a)}^{(k(a))}) x_a$. Moreover, we have

$$\begin{aligned} p(a_{\gamma(a)}^{(k(a))}) &\geq p(a) - 2^{k(a)} K \overset{(4.2)}{=} p(a) - \frac{1}{4} \frac{1}{\kappa + 1} \frac{1}{2^{\kappa - 1}} 2^{k(a)} T \\ &\overset{p(a) \geq 2^{k(a)} T}{\geq} p(a) - \left(\frac{1}{4} \frac{1}{\kappa + 1} \frac{1}{2^{\kappa - 1}} \right) p(a) = p(a) \cdot \left(1 - \frac{1}{4} \frac{1}{\kappa + 1} \frac{1}{2^{\kappa - 1}} \right) \end{aligned} \tag{4.5}$$

by the definition of the $L_\gamma^{(k)}$. We get

$$\begin{aligned} \text{OPT}(I_{L,\text{red}}, v) &\geq \sum_{a \in I_L} p(a_{\gamma(a)}^{(k(a))}) x_a \overset{(4.5)}{\geq} \sum_{a \in I_L} \left(1 - \frac{1}{4} \frac{1}{\kappa + 1} \frac{1}{2^{\kappa - 1}} \right) \cdot p(a) x_a \\ &= \left(1 - \frac{1}{4} \frac{1}{\kappa + 1} \frac{1}{2^{\kappa - 1}} \right) \text{OPT}(I_L, v) \\ &= \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1} \right) \text{OPT}(I_L, v) \quad . \end{aligned}$$

(The reasoning is partially taken directly from or close to the one by Lawler in [63].)

Note that we have used the expressions "items" and "item copies" interchangeably in this proof as described in Remark 4.2. □

Theorem 4.7. *$I_{L,\text{red}}$ has $\mathcal{O}(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})$ items. Algorithm 4.1 needs time in $\mathcal{O}(n + \frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})$ and space in $\mathcal{O}(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})$ for the construction and for saving $I_{L,\text{red}}$.*

Proof. The number of items $a_\gamma^{(k)}$, including the item $a_0^{(\kappa+1)}$, is bounded by $\mathcal{O}((\kappa + 1) \cdot (2^{\kappa+1}(\kappa + 1) - 1 + 1)) = \mathcal{O}(\log \frac{1}{\varepsilon} \cdot (\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})) = \mathcal{O}(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})$. The space needed is asymptotically bounded by the space required to save the $a_\gamma^{(k)}$. Finally, the running time is obviously bounded by $\mathcal{O}(n + \frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})$: the values $k(a)$ and $\gamma(a)$ can be found in $\mathcal{O}(1)$ because we assume that the logarithm can be determined in $\mathcal{O}(1)$. □

Remark 4.8. If there is one item a with the profit $p(a) = 2P_0$, i.e. whose profit attains the upper bound, one optimum solution obviously consists of this single item. During

the partition of I into I_L and I_S, it can easily be checked whether such an item is contained in I. Since the algorithm can directly stop if this is the case, we will from now on assume without loss of generality that such an item does not exist and that $a_0^{(\kappa+1)} = \emptyset$.

4.6 A Simplified Solution Structure

In this section, we will transform $I_{L,\text{red}}$ into a new instance $\tilde{I}$ whose optimum $\text{OPT}(\tilde{I}, v)$ is only slightly smaller than $\text{OPT}(I_{L,\text{red}}, v)$ and where the corresponding solution has a special structure. This new transformation will allow us later to faster construct the approximate solution. First, we define

$$I^{(k)} := \left\{ a \in I_{L,\text{red}} \mid p(a) \in L^{(k)} \right\} = \left\{ a \in I_{L,\text{red}} \mid p(a) \in \left[2^k T, 2^{k+1} T \right) \right\} \ .$$

Note that the items are already partitioned into the sets $I^{(k)}$ because of the way $I_{L,\text{red}}$ has been constructed.

Definition 4.9. *Let a_1, a_2 be two knapsack items with $s(a_1) + s(a_2) \leq c$. The gluing operation $\oplus$ combines them into a new item $a_1 \oplus a_2$ with $p(a_1 \oplus a_2) = p(a_1) + p(a_2)$ and $s(a_1 \oplus a_2) = s(a_1) + s(a_2)$.*

Thus, the gluing operation is only defined on pairs of items whose combined size does not exceed c.

The basic idea for the new instance $\tilde{I}$ is as follows: we first set $\tilde{I}^{(0)} := I^{(0)}$. Then, we construct $a_1 \oplus a_2$ for all $a_1, a_2 \in \tilde{I}^{(0)}$ (including the case $a_1 = a_2$), which yields the item set $\tilde{\tilde{I}}^{(1)} := \{a_1 \oplus a_2 \mid a_1, a_2 \in \tilde{I}^{(0)}\}$. Note that $p(a_1 \oplus a_2) \in [2T, 4T) = L^{(1)}$. For every profit interval $L_\gamma^{(1)}$, we keep only the item of smallest size in $I^{(1)} \cup \tilde{\tilde{I}}^{(1)}$, which yields the item set $\tilde{I}^{(1)}$. This procedure is iterated for $k = 1, \ldots, \kappa - 1$: the set $\tilde{I}^{(k)}$ contains the items with a profit in $[2^k T, 2^{k+1} T) = L^{(k)}$ (see Fig. 4.1(a)). Gluing like above yields the item set $\tilde{\tilde{I}}^{(k+1)}$ with profits in $[2^{k+1} T, 2^{k+2} T) = L^{(k+1)}$ (see Fig. 4.1(b)). By taking again the smallest item in $\tilde{\tilde{I}}^{(k+1)} \cup I^{(k+1)}$ for every $L_\gamma^{(k+1)}$, the set $\tilde{I}^{(k+1)}$ is derived (see Fig. 4.1(c)). The item in $\tilde{I}^{(k)}$ with a profit in $L_\gamma^{(k)}$ is denoted by $\tilde{a}_\gamma^{(k)}$ for every k and γ.

We finish when $\tilde{I}^{(\kappa)}$ has been constructed from $\tilde{I}^{(\kappa-1)}$. We are in the case where $I^{(\kappa+1)} = \emptyset$, i.e. $a_0^{(\kappa+1)} = \emptyset$, and it is explained at the beginning of Section 4.7 that it is not necessary to construct $\tilde{I}^{(\kappa+1)}$ from the items in $\tilde{I}^{(\kappa)}$. Hence, we also have $\tilde{a}_0^{(\kappa+1)} = \emptyset$.

Note that we may glue items together that already consist of glued items. For backtracking, we save for every $\tilde{a}_\gamma^{(k)}$ which two items in $\tilde{I}^{(k-1)}$ have formed it or whether $\tilde{a}_\gamma^{(k)}$ has already been an item in $I^{(k)}$. Algorithm 4.2 presents one way to construct the sets $\tilde{I}^{(k)}$.

Remark 4.10. One item $\tilde{a}_\gamma^{(k)}$ is in fact the combination of several items in $I_{L,\text{red}}$. The profit and size of $\tilde{a}_\gamma^{(k)}$ is equal to the total profit and size of these items. The $\tilde{a}_\gamma^{(k)}$ represent feasible item combinations because an arbitrary number of item copies can be taken in UKP.

Algorithm 4.2: The construction of the item sets $\tilde{I}^{(k)}$.

```
for k = 0, ..., κ do
    for γ = 0, ..., 2^{κ+1}(κ+1) − 1 do
        ã_γ^(k) := a_γ^(k);
        Backtrack(ã_γ^(k)) := a_γ^(k);
Ĩ^(0) := I^(0);
for k = 0, ..., κ − 1 do
    for γ = 0, ..., 2^{κ+1}(κ+1) − 1 do
        for γ' = γ, ..., 2^{κ+1}(κ+1) − 1 do
            if s(ã_γ^(k)) + s(ã_γ'^(k)) ≤ c then
                ã := ã_γ^(k) ⊕ ã_γ'^(k);
                if s(ã) < s(ã_γ(ã)^(k+1)) or ã_γ(ã)^(k+1) = ∅ then
                    ã_γ(ã)^(k+1) := ã;
                    Backtrack(ã_γ(ã)^(k+1)) := (ã_γ^(k), ã_γ'^(k));
    Ĩ^(k+1) := {ã_0^(k+1), ..., ã_{2^{κ+1}(κ+1)−1}^(k+1)};
```

The item set

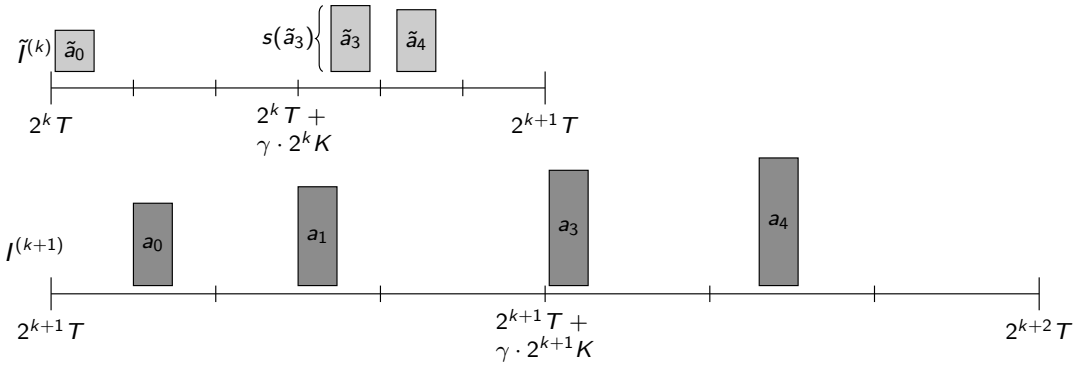
$$\tilde{I} := \bigcup_{k=0}^{\kappa} \tilde{I}^{(k)}$$

has for every $0 \leq v \leq c$ a solution near the original optimum $\text{OPT}\,(I_{L,\text{red}}, v)$ as shown below in Theorem 4.12. It is additionally proved that at most one item of every $\tilde{I}^{(k)}$ for $k \in \{0, \ldots, \kappa - 1\}$ is needed. First, we introduce a definition for the proof.

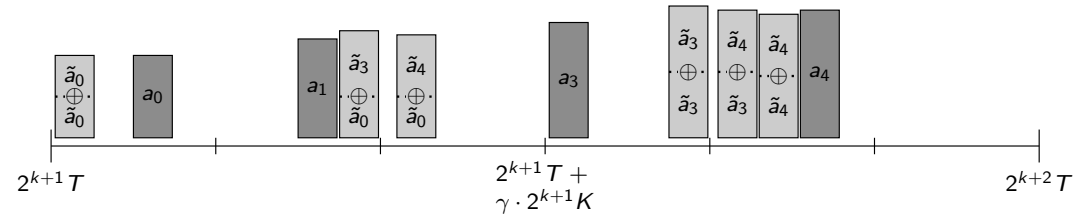
Definition 4.11. *Let I' be a set of knapsack items with $p(a) \geq T$ for every $a \in I'$. For a knapsack volume $v \leq c$ and $k_0 \in \{0, \ldots, \kappa\}$, a solution is* structured for $k = k_0$ *if it fits into v and uses for every $k \in \{0, \ldots, k_0\}$ at most one item copy with a profit in $L^{(k)} = [2^k T, 2^{k+1} T)$. We denote by $\text{OPT}_{\leq k_0}\,(I', v)$ the corresponding optimum profit.*

For instance, the solution for

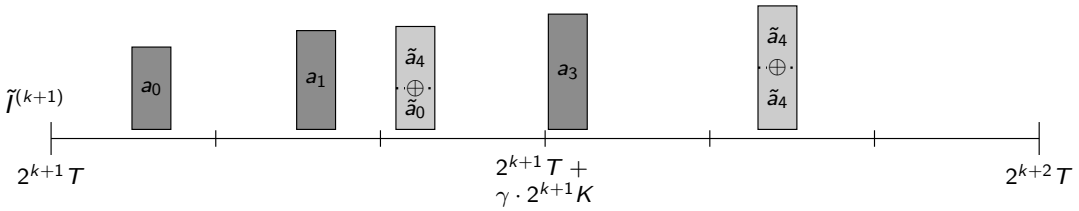
$$\text{OPT}_{\leq k_0}\left(\tilde{I}^{(0)} \cup \ldots \cup \tilde{I}^{(k_0)} \cup \tilde{I}^{(k_0+1)} \cup I^{(k_0+2)} \cup \ldots \cup I^{(\kappa)}, v\right)$$



(a) The items in $\tilde{I}^{(k)}$ and $I^{(k+1)}$. The height of every item a corresponds to its size $s(a)$ while its position on the axis corresponds to its profit $p(a)$. The axis is partitioned into the profit sub-intervals $L_\gamma^{(k)} = [2^k T + \gamma 2^k K, 2^k T + (\gamma + 1)2^k K)$.



(b) The set $I^{(k+1)}$ together with the newly constructed items in $\tilde{\tilde{I}}^{(k+1)}$



(c) The new set $\tilde{I}^{(k+1)}$ after keeping only the smallest item with a profit in $L_\gamma^{(k+1)}$. For instance, $\tilde{a}_4 \oplus \tilde{a}_4$ is kept because it is the smallest item in its profit sub-interval $L_\gamma^{(k+1)}$.

Figure 4.1: Principle of deriving $\tilde{I}^{(k+1)}$ from $\tilde{I}^{(k)}$ and $I^{(k+1)}$

fits into the volume v, and it uses only one item from every $\tilde{I}^{(k)}$ for $k \in \{0, \dots, k_0\}$. It may however use an arbitrary number of item copies e.g. in $\tilde{I}^{(k_0+1)}$ or $I^{(k_0+2)}$.

Theorem 4.12. *For $v \leq c$ and $k_0 \in \{0, \dots, \kappa - 1\}$, we have*

$$\mathrm{OPT}_{\leq k_0}\left(\bigcup_{k=0}^{k_0+1} \tilde{I}^{(k)} \cup \bigcup_{k=k_0+2}^{\kappa} I^{(k)}, v\right) \geq \left(1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{k_0+1} \mathrm{OPT}\left(I_{L,\mathrm{red}}, v\right) \ .$$

Proof. The proof idea is quite simple: we iteratively replace the items in $I^{(k_0+1)}$ by their counterpart in $\tilde{I}^{(k_0+1)}$ and also replace every pair of item copies in $\tilde{I}^{(k_0)}$ by the counterpart in $\tilde{I}^{(k_0+1)}$. This directly follows the way to construct the item sets $\tilde{I}^{(k)}$ presented in Algorithm 4.2.

Formally, the statement is proved by induction over k_0. Let $k_0 = 0$. Take an optimum solution to $\tilde{I}^{(0)} \cup I^{(1)} \cup \ldots \cup I^{(\kappa)} = I^{(0)} \cup I^{(1)} \cup \ldots \cup I^{(\kappa)} = I_{L,\mathrm{red}}$. For ease of notation, we directly write each item as often as it appears in the solution, i.e. we directly consider the item copies. We have three sub-sequences:

- Let $\bar{a}_1, \dots, \bar{a}_\eta$ ($\eta \in \mathbb{N}$) be the item copies from $\tilde{I}^{(0)} = I^{(0)}$ in the optimal solution for $\mathrm{OPT}(I_{L,\mathrm{red}}, v)$. We assume that η is odd (the case where η is even is easier and handled below.)
- Let $\bar{a}_{\eta+1}, \dots, \bar{a}_{\eta+\xi}$ ($\xi \in \mathbb{N}$) be the item copies from $I^{(1)}$ in the optimal solution for $\mathrm{OPT}(I_{L,\mathrm{red}}, v)$.
- Let $\bar{a}'_1, \dots, \bar{a}'_\lambda$ ($\lambda \in \mathbb{N}$) be the remaining item copies from $I^{(2)} \cup \ldots \cup I^{(\kappa)}$ in the optimal solution for $\mathrm{OPT}(I_{L,\mathrm{red}}, v)$. This set is denoted by Λ. As defined above, the total profit of these items is written as $p(\Lambda)$.

Figure 4.2(a) illustrates the packing. (Figure 4.2 shows the case for general k.) We have

$$\mathrm{OPT}\left(\tilde{I}^{(0)} \cup I^{(1)} \cup \ldots \cup I^{(\kappa)}, v\right) = \sum_{i=1}^{\eta} p(\bar{a}_i) + \sum_{j=\eta+1}^{\eta+\xi} p(\bar{a}_j) + p(\Lambda) \ . \tag{4.6}$$

In the first step, every pair of items $\bar{a}_{2i-1}$ and $\bar{a}_{2i}$ from $\tilde{I}^{(0)}$ for $i \in \{1, \dots, \lfloor \frac{\eta}{2} \rfloor\}$ is replaced by $\bar{a}_{2i-1} \oplus \bar{a}_{2i} \in \tilde{\tilde{I}}^{(1)}$ (see Fig. 4.2(b)). In the second step, every item $\bar{a}_{2i-1} \oplus \bar{a}_{2i}$ is again replaced by the corresponding item $\tilde{a}^{(1)}_{\gamma(\bar{a}_{2i-1} \oplus \bar{a}_{2i})} =: \tilde{a}^{(1)}_{\rho(i)}$ in $\tilde{I}^{(1)}$ (for $i \in \{1, \dots, \lfloor \frac{\eta}{2} \rfloor\}$). Only the item $\bar{a}_\eta$ remains unchanged. Moreover, $\bar{a}_j$ from $I^{(1)}$ is replaced by the corresponding $\tilde{a}^{(1)}_{\gamma(\bar{a}_j)} =: \tilde{a}^{(1)}_{\rho(j)}$ for $j \in \{\eta + 1, \dots, \eta + \xi\}$ (see Fig. 4.2(c)). Note that this new solution is indeed feasible because the replacing items $\tilde{a}^{(1)}_\gamma$ are at most as large as the original ones. Moreover, the corresponding items $\tilde{a}^{(1)}_{\rho(i)}$ and $\tilde{a}^{(1)}_{\rho(j)}$ must exist by the construction of $\tilde{I}^{(1)}$. Thus, we have a (feasible) solution that consists

of the item $\bar{a}_\eta \in \tilde{I}^{(0)}$, the items $\tilde{a}^{(1)}_{\rho(i)}$ and $\tilde{a}^{(1)}_{\rho(j)}$ in $\tilde{I}^{(1)}$, and the remaining items $\bar{a}'_1, \ldots, \bar{a}'_\lambda$ in $I^{(2)}, \ldots, I^{(\kappa)}$: this solution respects the structure of $\mathrm{OPT}_{\leq k_0}(\cdot, v)$ for $k_0 = 0$. (If η is even, no item in $\tilde{I}^{(0)}$ is used.)

Let now $\bar{a}$ be an item $\bar{a}_{2i-1} \oplus \bar{a}_{2i}$ or $\bar{a}_j$. It can be proved like for Inequality (4.5) that

$$p(\tilde{a}^{(1)}_{\gamma(\bar{a})}) \geq \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right) p(\bar{a}) \ . \tag{4.7}$$

Thus, we have

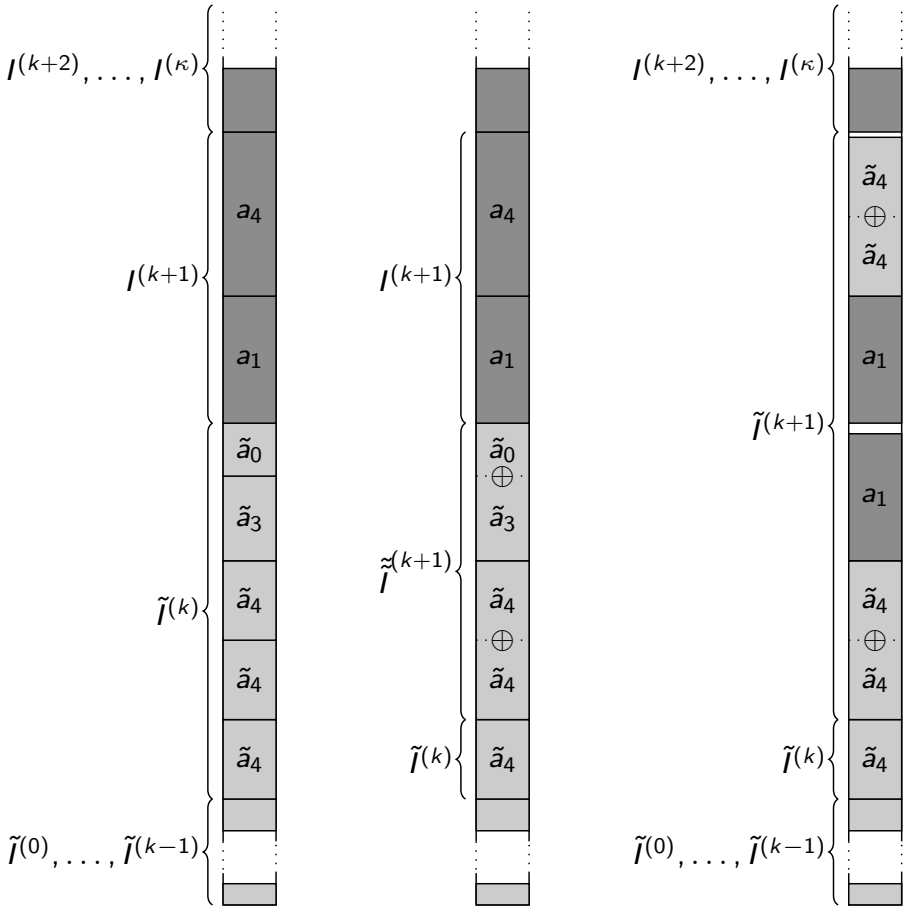
$$\begin{aligned}
\mathrm{OPT}_{\leq 0}\left(\tilde{I}^{(0)} \cup \tilde{I}^{(1)} \cup I^{(2)} \cup \ldots \cup I^{(\kappa)}, v\right) &\geq p(\bar{a}_\eta) + \sum_{i=1}^{\lfloor \frac{\eta}{2} \rfloor} p(\tilde{a}^{(1)}_{\rho(i)}) + \sum_{j=\eta+1}^{\eta+\xi} p(\tilde{a}^{(1)}_{\rho(j)}) + p(\Lambda) \\
&\overset{(4.7)}{\geq} p(\bar{a}_\eta) + \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right) \sum_{i=1}^{\lfloor \frac{\eta}{2} \rfloor} p(\bar{a}_{2i-1} \oplus \bar{a}_{2i}) \\
&\quad + \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right) \sum_{j=\eta+1}^{\eta+\xi} p(\bar{a}_j) + p(\Lambda) \\
&\geq \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right) \left(\sum_{i=1}^{\eta} p(\bar{a}_i) + \sum_{j=\eta+1}^{\eta+\xi} p(\bar{a}_j) + p(\Lambda)\right) \\
&\overset{(4.6)}{=} \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right) \mathrm{OPT}\left(\tilde{I}^{(0)} \cup I^{(1)} \cup \ldots \cup I^{(\kappa)}, v\right) \\
&= \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right) \mathrm{OPT}\left(I_{L,\mathrm{red}}, v\right) \ .
\end{aligned}$$

The statement for $k_0 = 1, \ldots, \kappa - 1$ now follows by induction. The proof is almost identical to the case $k_0 = 0$ above, the only difference is that there are additionally the items in $\tilde{I}^{(0)}, \ldots, \tilde{I}^{(k_0-1)}$ that remain unchanged like the items in $I^{(k_0+2)}, \ldots, I^{(\kappa)}$. Only items in $\tilde{I}^{(k_0)}$ and $I^{(k_0)}$ are replaced.

Note that we have again used the expressions "items" and "item copies" interchangeably in this proof as described in Remark 4.2. □

Lemma 4.13. $\mathrm{OPT}(\tilde{I} \cup \{a_{\mathrm{eff}}\}) \leq \mathrm{OPT}(I_{L,\mathrm{red}} \cup I_S) \leq \mathrm{OPT}(I_L \cup I_S) = \mathrm{OPT}(I) \leq 2P_0$ *holds.*

Proof. $\tilde{I}$ consists of items in $I_{L,\mathrm{red}}$ or of items that can be obtained by gluing several items in $I_{L,\mathrm{red}}$ together. Every combination of items in $\tilde{I}$ can therefore be represented by items in $I_{L,\mathrm{red}}$ (see also Remark 4.10). Moreover, we have $a_{\mathrm{eff}} \in I_S$. The first inequality follows. Since $I_{L,\mathrm{red}} \subseteq I_L$, the second inequality is obvious. The last inequality follows from Theorem 4.3. □



(a) The current solution to $\tilde{I}^{(0)} \cup \cdots \cup \tilde{I}^{(k)} \cup I^{(k+1)} \cup \cdots \cup I^{(\kappa)}$. The structure of $\mathrm{OPT}_{\leq k-1}(\cdot, v)$ is respected, i.e. at most one item from every $\tilde{I}^{(0)}, \ldots, \tilde{I}^{(k-1)}$ is used.

(b) The items in $\tilde{I}^{(k)}$ are pairwise glued together with the possible exception of one item.

(c) The items in $\tilde{\tilde{I}}^{(k+1)} \cup I^{(k+1)}$ are replaced by their counterparts in $\tilde{I}^{(k+1)}$. Now, at most one item in $\tilde{I}^{(k)}$ is part of the solution, and the structure for $\mathrm{OPT}_{\leq k}(\cdot, v)$ is respected.

Figure 4.2: The principle of the proof for Theorem 4.12

Up to now, we have (only) reduced the original item set I to $\tilde{I} \cup \{a_{\text{eff}}\}$.

Lemma 4.14. *Assume as mentioned in Remark 4.8 that $a_0^{(\kappa+1)} = \emptyset$. Consider the optimum structured solutions to $\tilde{I} \cup \{a_{\text{eff}}\}$ for $k_0 = \kappa - 1$ (see Definition 4.11). This means that at most one item is used from every $\tilde{I}^{(k)}$ for $k \in \{0, \ldots, \kappa - 1\}$. (The item a_{eff} has a profit $p(a_{\text{eff}}) < T$ such that it does not have to satisfy any structural conditions.) Then there are two possible cases:*

- *One solution uses (at least) two items in $\tilde{I}^{(\kappa)}$. This is the case if and only if the optimum for $\tilde{I} \cup \{a_{\text{eff}}\}$ is $2P_0$, and the solution consists of two item copies of the item $\tilde{a}_0^{(\kappa)}$ with $p(\tilde{a}_0^{(\kappa)}) = P_0$.*
- *Every solution uses at most one item in $\tilde{I}^{(\kappa)}$. Then, $\text{OPT}_{\leq \kappa-1}(\tilde{I}, v') = \text{OPT}_{\leq \kappa}(\tilde{I}, v')$ holds for all values $0 \leq v' \leq c$, and there is a value $0 \leq v \leq c$ such that*

$$\begin{aligned}\text{OPT}_{\leq \kappa}\left(\tilde{I}, v\right) + \text{OPT}\left(\{a_{\text{eff}}\}, c - v\right) &= \text{OPT}_{\leq \kappa-1}\left(\tilde{I}, v\right) + \text{OPT}\left(\{a_{\text{eff}}\}, c - v\right) \\ &\geq \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa+1} \text{OPT}(I) - T \ .\end{aligned}$$

Moreover, $\text{OPT}_{\leq \kappa}(\tilde{I}, v)$ uses at least one item in $\tilde{I}^{(\kappa-2)} \cup \tilde{I}^{(\kappa-1)} \cup \tilde{I}^{(\kappa)}$, and/or we have $\text{OPT}\left(\{a_{\text{eff}}\}, c - v\right) \geq \frac{1}{4} P_0$.

Proof. Note that $I_{L,\text{red}}$ does not contain any item with the profit $2P_0$ (see Remark 4.8). By construction, this is still the case for $\tilde{I}$. Suppose that one solution to $\tilde{I} \cup \{a_{\text{eff}}\}$ uses more than one item in $\tilde{I}^{(\kappa)}$. Since items in $\tilde{I}^{(\kappa)}$ have profits in $[P_0, 2P_0)$, only two copies of the item $\tilde{a}_0^{(\kappa)}$ can be used, and we have $p(\tilde{a}_0^{(\kappa)}) = P_0$. In fact, $2P_0$ is the maximum possible profit because $\text{OPT}(\tilde{I} \cup \{a_{\text{eff}}\}) \leq \text{OPT}(I) \leq 2P_0$ holds as we have seen in Lemma 4.13. Thus, the "only if" direction has been proved. The "if" direction is obvious.

Suppose now that every structured solution to $\tilde{I} \cup \{a_{\text{eff}}\}$ for $k_0 = \kappa - 1$ uses at most one item in $\tilde{I}^{(\kappa)}$. Thus, $\text{OPT}_{\leq \kappa-1}(\tilde{I}, v') = \text{OPT}_{\leq \kappa}(\tilde{I}, v')$ holds for all $0 \leq v' \leq c$.

Let $v \leq c$ now be the volume the large items I_L occupy in an optimum solution to I. Then obviously $\text{OPT}(I) = \text{OPT}\left(I_L, v\right) + \text{OPT}\left(I_S, c - v\right)$ holds. We have the following inequality:

$$\begin{aligned}\text{OPT}_{\leq \kappa}\left(\tilde{I}, v\right) + \text{OPT}\left(\{a_{\text{eff}}\}, c - v\right) &= \text{OPT}_{\leq \kappa-1}\left(\tilde{I}, v\right) + \text{OPT}\left(\{a_{\text{eff}}\}, c - v\right) \\ &\overset{\text{Thm. 4.12}}{\geq} \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa} \text{OPT}\left(I_{L,\text{red}}, v\right) + \text{OPT}\left(\{a_{\text{eff}}\}, c - v\right) \\ &\overset{\text{Lem. 4.6}}{\geq} \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa+1} \text{OPT}\left(I_L, v\right) + \text{OPT}\left(I_S, c - v\right) - T\end{aligned}$$

$$\begin{aligned} &\geq \left(1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon})+1}\right)^{\kappa+1} (\mathrm{OPT}(I_L, v) + \mathrm{OPT}(I_S, c-v)) - T \\ &= \left(1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon})+1}\right)^{\kappa+1} \mathrm{OPT}(I) - T \ . \end{aligned} \tag{4.8}$$

For the final property, suppose that no item in $\tilde{I}^{(\kappa-2)} \cup \tilde{I}^{(\kappa-1)} \cup \tilde{I}^{(\kappa)}$ is used in a solution for $\mathrm{OPT}_{\leq\kappa}(\tilde{I}, v)$. Then we have

$$\mathrm{OPT}_{\leq\kappa}\left(\tilde{I}, v\right) \leq \sum_{k=0}^{\kappa-3} \max\left\{p(a) \mid a \in \tilde{I}^{(k)}\right\} \leq \sum_{k=0}^{\kappa-3} 2 \cdot 2^k T < 2^{\kappa-1} T \overset{(4.1)}{=} \frac{1}{2} P_0 \ .$$

On the other hand, Inequality (4.8) together with $(1-\delta)^k \geq (1 - k \cdot \delta)$ for $\delta < 1$ yields

$$\begin{aligned} &\mathrm{OPT}_{\leq\kappa}\left(\tilde{I}, v\right) + \mathrm{OPT}\left(\{a_{\text{eff}}\}, c - v\right) \\ &\qquad \geq \left(1 - \frac{\varepsilon}{4}\frac{\kappa+1}{\log_2(\frac{2}{\varepsilon})+1}\right) \mathrm{OPT}(I) - T \\ &\qquad \overset{(4.1)}{=} \left(1 - \frac{\varepsilon}{4}\right) \mathrm{OPT}(I) - \frac{1}{2}\varepsilon P_0 \\ &\qquad \geq \left(1 - \frac{\varepsilon}{4}\right) \mathrm{OPT}(I) - \frac{1}{2}\varepsilon\, \mathrm{OPT}(I) \overset{\varepsilon \leq 1/4}{\geq} \frac{3}{4}\mathrm{OPT}(I) \geq \frac{3}{4} P_0 \ . \end{aligned}$$

Hence, $\mathrm{OPT}(\{a_{\text{eff}}\}, c - v) \geq \frac{1}{4} P_0$ holds. The final property of the second case follows.□

Definition 4.15. *Take $\lceil \frac{P_0/4}{p(a_{\text{eff}})} \rceil$ items a_{eff}. If their total size is at most c, they are glued together to $a_{\text{eff}-\text{c}}$.*

Obviously, $a_{\text{eff}-\text{c}}$ consists of the smallest number of items a_{eff} whose total profit is at least $\frac{P_0}{4}$. Moreover, $a_{\text{eff}-\text{c}}$ is a large item.

Definition 4.16. *Take a knapsack volume $v \leq c$. Consider the following solutions to $\tilde{I} \cup \{a_{\text{eff}-\text{c}}\}$ of size at most v:*

- *They are structured for $k = \kappa$, i.e. they use for every $k \in \{0, \ldots, \kappa\}$ at most one item in $\tilde{I}^{(k)}$.*
- *They additionally use the item $a_{\text{eff}-\text{c}}$ at most once and at least one item $\tilde{a} \in \tilde{I}^{(\kappa-2)} \cup \tilde{I}^{(\kappa-1)} \cup \tilde{I}^{(\kappa)} \cup \{a_{\text{eff}-\text{c}}\}$.*

Hence, these solutions have a profit of at least $p(\tilde{a}) \geq 2^{\kappa-2} T = \frac{1}{4} P_0$. These special solutions are called structured solutions with a lower bound *(on the profit).*

The value $\mathrm{OPT}_{\text{St}}(\tilde{I} \cup \{a_{\text{eff}-\text{c}}\}, v)$ *denotes the optimal profit for such solutions of total size at most v. If v is too small so that such a solution does not exist, we set* $\mathrm{OPT}_{\text{St}}(\tilde{I} \cup \{a_{\text{eff}-\text{c}}\}, v) = 0$.

Theorem 4.17. *In the second case of Lemma 4.14, there is a value $0 \leq v \leq c$ such that*

$$\mathrm{OPT}_{\mathrm{St}}\left(\tilde{I} \cup \{a_{\mathrm{eff-c}}\}, v\right) + \mathrm{OPT}\left(\{a_{\mathrm{eff}}\}, c - v\right) \geq \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa+1} \mathrm{OPT}(I) - T \ .$$

Proof. Like in the proof of Lemma 4.14, let v' be the volume the large items I_L occupy in an optimum solution to I so that $\mathrm{OPT}(I_L, v') + \mathrm{OPT}(I_S, c - v') = \mathrm{OPT}(I)$. Consider an optimum structured solution for $\mathrm{OPT}_{\leq\kappa}(\tilde{I}, v')$ and suppose that it does not use any items in $\tilde{I}^{(\kappa-2)} \cup \tilde{I}^{(\kappa-1)} \cup \tilde{I}^{(\kappa)}$. Lemma 4.14 states that $\mathrm{OPT}(\{a_{\mathrm{eff}}\}, c - v')$ has a profit of at least $\frac{1}{4}P_0$. Thus, a subset of the item copies of a_{eff} can be replaced by $a_{\mathrm{eff-c}}$, and $c - v' \geq s(a_{\mathrm{eff-c}})$. We set $v := v' + s(a_{\mathrm{eff-c}})$. Note that $\mathrm{OPT}_{\mathrm{St}}(\tilde{I} \cup \{a_{\mathrm{eff-c}}\}, v) \geq \mathrm{OPT}_{\leq\kappa}(\tilde{I}, v') + p(a_{\mathrm{eff-c}})$. Moreover, $\mathrm{OPT}_{\leq\kappa}(\tilde{I}, v') = \mathrm{OPT}_{\leq\kappa-1}(\tilde{I}, v')$ holds because we are in the second case of Lemma 4.14. We get the following inequalities:

$$\begin{aligned}
&\mathrm{OPT}_{\mathrm{St}}\left(\tilde{I} \cup \{a_{\mathrm{eff-c}}\}, v\right) + \mathrm{OPT}\left(\{a_{\mathrm{eff}}\}, c - v\right) \\
&\quad \geq \quad \mathrm{OPT}_{\leq\kappa}\left(\tilde{I}, v'\right) + p(a_{\mathrm{eff-c}}) + \mathrm{OPT}\left(\{a_{\mathrm{eff}}\}, c - v' - s(a_{\mathrm{eff-c}})\right) \\
&\quad = \quad \mathrm{OPT}_{\leq\kappa-1}\left(\tilde{I}, v'\right) + \mathrm{OPT}\left(\{a_{\mathrm{eff}}\}, c - v'\right) \\
&\quad \overset{\text{Thm. 4.12}}{\geq} \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa} \mathrm{OPT}\left(I_{L,\mathrm{red}}, v'\right) + \mathrm{OPT}\left(\{a_{\mathrm{eff}}\}, c - v'\right) \\
&\quad \overset{\text{Lem. 4.6}}{\geq} \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa+1} \mathrm{OPT}\left(I_L, v'\right) + \mathrm{OPT}\left(I_S, c - v'\right) - T \\
&\quad \geq \quad \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa+1} \left(\mathrm{OPT}\left(I_L, v'\right) + \mathrm{OPT}\left(I_S, c - v'\right)\right) - T \\
&\quad = \quad \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa+1} \mathrm{OPT}(I) - T \ .
\end{aligned}$$

Note that $\mathrm{OPT}(\{a_{\mathrm{eff}}\}, c - v)$ is well-defined—and therefore the entire chain of inequalities feasible—because $c - v = c - v' - s(a_{\mathrm{eff-c}}) \geq 0$.

Suppose now that the optimal solution uses at least one item in $\tilde{I}^{(\kappa-2)} \cup \tilde{I}^{(\kappa-1)} \cup \tilde{I}^{(\kappa)}$. We can then directly set $v := v'$, and the proof is similar to the first case above.

Roughly speaking, a solution in the first case of this proof satisfies the lower bound of the theorem on the profit and uses at most one item in every $\tilde{I}^{(k)}$, but no item in $\tilde{I}^{(\kappa-2)}$, $\tilde{I}^{(\kappa-1)}$ or $\tilde{I}^{(\kappa)}$. This implies that enough items a_{eff} are part of the solution such that a subset of them can be replaced by $a_{\mathrm{eff-c}}$. □

So far, we have not constructed an actual solution. We have only shown in Theorem 4.17 that there is a solution to $\tilde{I} \cup \{a_{\mathrm{eff-c}}\} \cup \{a_{\mathrm{eff}}\}$ that is close to $\mathrm{OPT}(I)$ and that is a structured solution with a lower bound.

Theorem 4.18. *The cardinality of $\tilde{I}^{(k)}$ is in $\mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$, i.e. $\tilde{I}$ has $\mathcal{O}(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})$ items. Algorithm 4.2 constructs $\tilde{I}$ in time $\mathcal{O}(\frac{1}{\varepsilon^2}\log^3\frac{1}{\varepsilon})$ and space $\mathcal{O}(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})$, which also includes the space to store $\tilde{I}$ and the backtracking information. The item $a_{\text{eff}-\text{c}}$ can be constructed in time $\mathcal{O}(1)$.*

Proof. The statement for $a_{\text{eff}-\text{c}}$ is trivial: the number of items $a_{\text{eff}-\text{c}}$ to glue together can be determined by division.

The number of items in $\tilde{I}^{(k)}$ and $\tilde{I}$ can be derived like the number of items in $I_{L,\text{red}}$ in Theorem 4.7. The running time of Algorithm 4.2 is obviously dominated by the second for-loop. It is in

$$\mathcal{O}\left(\kappa\cdot\left(2^{\kappa+1}(\kappa+1)\right)^2\right)=\mathcal{O}\left(\log\left(\frac{1}{\varepsilon}\right)\cdot\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)^2\right)=\mathcal{O}\left(\frac{1}{\varepsilon^2}\log^3\left(\frac{1}{\varepsilon}\right)\right)\ .$$

The space complexity is dominated by the space to save the $\tilde{a}_\gamma^{(k)}$ and the backtracking information, which is again asymptotically equal to the number of items in $\tilde{I}$. □

4.7 Finding an Approximate Structured Solution by Dynamic Programming

The previous section has presented three cases:

1. The instance I has one item of profit $2P_0$: return this item for an optimum solution, and $\text{OPT}(I)=2P_0$ (see Remark 4.8).
2. If this is not the case, and $\tilde{I}$ has one item of profit P_0 and size at most $\frac{c}{2}$, two copies of this item are an optimum solution to $\tilde{I}\cup\{a_{\text{eff}}\}$ (see Lemma 4.14). Undoing the gluing returns an optimum solution to I with $\text{OPT}(I)=2P_0$.
3. Otherwise, there is an approximate solution to $\tilde{I}\cup\{a_{\text{eff}-\text{c}}\}\cup\{a_{\text{eff}}\}$ where the large items are a structured solution with a lower bound (see Theorem 4.17).

The first two cases can be easily checked, which is the reason why it has not been necessary to construct the set $\tilde{I}^{(\kappa+1)}$. We will from now on assume that we are in the third case: regarding the large items, a solution uses at most one item from every $\tilde{I}^{(k)}$ for $k\in\{0,\ldots,\kappa\}$ as well as $a_{\text{eff}-\text{c}}$ at most once. At the same time, at least one item $\tilde{a}\in\tilde{I}^{(\kappa-2)}\cup\tilde{I}^{(\kappa-1)}\cup\tilde{I}^{(\kappa)}\cup\{a_{\text{eff}-\text{c}}\}$ is chosen. (See Definition 4.16.)

We use dynamic programming to find for all $0\le v\le c$ the corresponding set of large items $V\subseteq\tilde{I}\cup\{a_{\text{eff}-\text{c}}\}$ with $s(V)\le v$. For convenience, let $\tilde{I}^{(\kappa+1)}:=\{a_{\text{eff}-\text{c}}\}$. We introduce tuples (p,s,k) similar to Lawler [63]. For profit p with $0\le p\le 2P_0$ and size

$0 \leq s \leq c$, the tuple (p, s, k) states that there is an item set of size s whose total profit is p. Moreover, the set has only items in $\tilde{I}^{(k)} \cup \cdots \cup \tilde{I}^{(\kappa+1)}$ and respects the structure above.

The dynamic program is quite simple: start with the dummy tuple set $F^{(\kappa+2)} := \{(0, 0, \kappa+2)\}$. For $k = \kappa+1, \ldots, \kappa-2$, the tuples in $F^{(k)}$ are recursively constructed by

$$F^{(k)} := \left\{(p, s, k) \mid (p, s, k+1) \in F^{(k+1)}\right\} \cup \left\{(p + p(\tilde{a}), s + s(\tilde{a}), k) \mid (p, s, k+1) \in F^{(k+1)}, \tilde{a} \in \tilde{I}^{(k)}, s + s(\tilde{a}) \leq c\right\} .$$

Note that $(0, 0, k+1) \in F^{(k+1)}$, which guarantees that $F^{(k)}$ also contains the tuples $(p(\tilde{a}), s(\tilde{a}), k)$ for $\tilde{a} \in \tilde{I}^{(k)}$ if $k \in \{\kappa+1, \ldots, \kappa-2\}$. For $k = \kappa-3, \ldots, 0$, this tuple $(0, 0, k+1)$ is no longer considered to form the new tuples, which guarantees that tuples of the form $(p + p(\tilde{a}), s + s(\tilde{a}), k)$ for $\tilde{a} \in \tilde{I}^{(k)}$ have $p, s \neq 0$. The recursion becomes

$$F^{(k)} := \left\{(p, s, k) \mid (p, s, k+1) \in F^{(k+1)}\right\} \cup \left\{(p + p(\tilde{a}), s + s(\tilde{a}), k) \mid (p, s, k+1) \in F^{(k+1)} \setminus \{(0, 0, k+1)\}, \tilde{a} \in \tilde{I}^{(k)}, s + s(\tilde{a}) \leq c\right\} .$$

The actual item set corresponding to (p, s, k) can be reconstructed by saving backtracking information.

Definition 4.19. *A tuple (p_2, s_2, k) is dominated by (p_1, s_1, k) if $p_2 \leq p_1$ and $s_2 \geq s_1$.*

As in [63] and similar to Subsection 3.5.1, dominated tuples $(p, s, k+1)$ are now removed from $F^{(k+1)}$ before $F^{(k)}$ is constructed. This does not affect the outcome: dominated tuples only stand for sets of items with a profit not larger and a size not smaller than non-dominated tuples. A non-dominated tuple (p, s, k) is therefore optimal, i.e. the profit p can only be obtained with items of size at least s if items in $\tilde{I}^{(k)}, \ldots, \tilde{I}^{(\kappa+1)}$ are considered.

Lemma 4.20. *A tuple $(p, s, k) \in F^{(k)}$ stands for a structured solution with a lower bound (see Definition 4.16). Therefore, we have $p \geq 2^{\kappa-2}T$ if $p > 0$. For every $v \leq c$, there is a tuple $(p, s, 0) \in F^{(0)}$ with $p = \mathrm{OPT}_{\mathrm{St}}(\tilde{I} \cup \{a_{\mathrm{eff-c}}\}, v)$ and $s \leq v$.*

Proof. This lemma directly follows from the dynamic program: tuples use at most one item from every $\tilde{I}^{(k)}$. For $k \in \{\kappa-2, \ldots, \kappa+1\}$, a tuple with $p > 0$ represents an item set that uses at least one item in $\tilde{I}^{(k)}, \ldots, \tilde{I}^{(\kappa+1)}$, and such an item has a profit

of at least $2^{\kappa-2}T$. Tuples for $k \leq \kappa - 3$ with $p > 0$ are only derived from tuples that use at least one item in $\tilde{I}^{(\kappa-2)}, \ldots, \tilde{I}^{(\kappa+1)}$. If dominated tuples are not removed, the dynamic program obviously constructs tuples for all possible structured solutions with a lower bound, especially the optimum combinations for every $0 \leq v \leq c$. Removing dominated tuples does not affect the tuples that stand for the optimum item combinations so that the second property still holds. □

While the dynamic program above constructs the desired tuples, their number may increase dramatically until $F^{(0)}$ is obtained. We therefore use approximate dynamic programming for the tuples with profits in $[\frac{1}{4}P_0, 2P_0]$. This method is inspired by the dynamic programming used in [58] (see also [61, pp. 97–112]).

Definition 4.16 and Lemma 4.20 state that a tuple (p, s, k) with $p > 0$ satisfies $p \geq 2^{\kappa-2}T$. Apart from $(0, 0, k)$, all tuples have therefore profits in the interval $[2^{\kappa-2}T, 2P_0] \overset{(4.1)}{=} [\frac{1}{4}P_0, 2P_0] = [2^{\kappa-2}T, 2^{\kappa+1}T]$. We partition this interval into sub-intervals of length $2^{\kappa-2}K$. We get

$$\begin{aligned}[2^{\kappa-2}T, 2P_0] &= \bigcup_{\xi=0}^{\xi_0} \left[2^{\kappa-2}T + \xi \cdot 2^{\kappa-2}K,\ 2^{\kappa-2}T + (\xi+1)2^{\kappa-2}K\right) \cup \{2P_0\} \\ &=: \bigcup_{\xi=0}^{\xi_0} \tilde{L}_\xi^{(\kappa-2)} \cup \tilde{L}_{\xi_0+1}^{(\kappa-2)}\end{aligned}$$

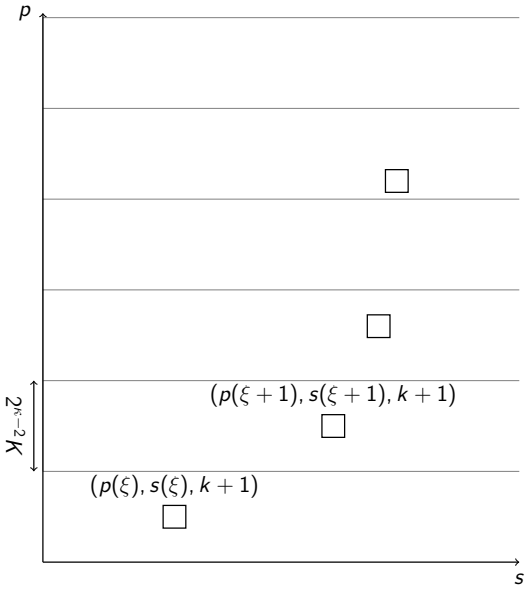
for $\xi_0 := 7(\kappa+1)2^{\kappa+1} - 1$. (A short calculation shows that $2^{\kappa-2}T + (\xi_0+1)2^{\kappa-2}K = 2P_0$.) The approximate dynamic program keeps for every $\xi \in \{0, \ldots, \xi_0 + 1\}$ only the tuple (p, s, k) with $p \in \tilde{L}_\xi^{(\kappa-2)}$ that has the smallest size s. The dominated tuples are removed when all tuples for k have been constructed. The modified dynamic program is presented in Algorithm 4.3 and shown in Figure 4.3. The sets of these non-dominated tuples are denoted by $D^{(k)}$. For convenience, $(p(\xi), s(\xi), k) \in D^{(k)}$ denotes the smallest tuple with a profit in $\tilde{L}_\xi^{(\kappa-2)}$. We again save the backtracking information during the execution of the algorithm.

Lemma 4.21. *Let $\tilde{D}^{(k)}$ be the set $D^{(k)}$ from Algorithm 4.3 before the dominated tuples are removed. A tuple $(p, s, k) \in \tilde{D}^{(k)}$ for $k = \kappa+1, \ldots, 0$ stands for a structured solution with a lower bound. Therefore, we have $p \geq 2^{\kappa-2}T$ if $p > 0$. This is also true for $(p, s, k) \in D^{(k)}$.*

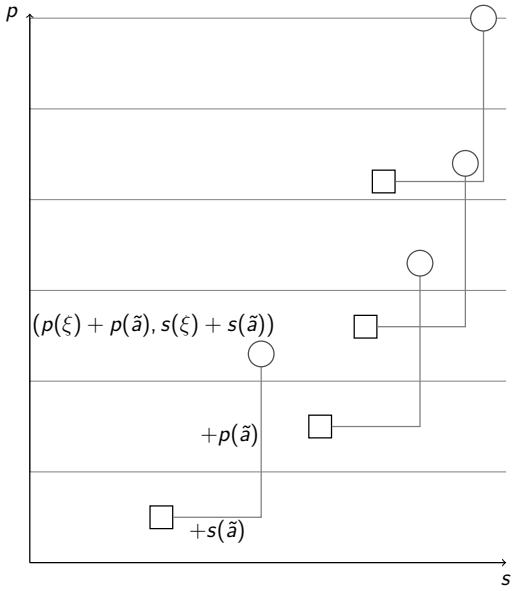
Proof. The proof is almost identical to the one of Lemma 4.20. In fact, the proof is not influenced by keeping only the tuple of smallest size in every profit interval $\tilde{L}_\xi^{(\kappa-2)}$.□

Algorithm 4.3: The approximate dynamic program

$D^{(\kappa+2)} := \{(0,0,\kappa+2)\}$;
Backtrack$(0,0,\kappa+2) := \emptyset$;
for $k = \kappa+1, \ldots, 0$ **do**
  $D^{(k)} := \emptyset$;
  for $(p(\xi), s(\xi), k+1) \in D^{(k+1)}$ **do**
    $D^{(k)} := D^{(k)} \cup \{(p(\xi), s(\xi), k)\}$;
    Backtrack$(p(\xi), s(\xi), k) :=$ Backtrack$(p(\xi), s(\xi), k+1)$;
  for $\tilde{a} \in \tilde{I}^{(k)}$ **do**
    for $(p, s, k+1) \in D^{(k+1)} \setminus \{(0,0,k+1)\}$ **do**
      `// Construction of new tuples`
      $(p', s', k) := (p + p(\tilde{a}), s + s(\tilde{a}), k)$;
      Determine ξ' for (p', s', k) such that $p' \in \tilde{L}^{(\kappa-2)}_{\xi'}$;
      if $s' < s(\xi')$ *or* $(p(\xi'), s(\xi'), k) = \emptyset$ **then**
        `// Only new tuples of smaller size are kept`
        $D^{(k)} := D^{(k)} \setminus \{(p(\xi'), s(\xi'), k)\}$;
        $(p(\xi'), s(\xi'), k) := (p', s', k)$;
        Backtrack$(p(\xi'), s(\xi'), k) := ((p, s, k+1), \tilde{a})$;
        $D^{(k)} := D^{(k)} \cup \{(p(\xi'), s(\xi'), k)\}$;
    if $k \geq \kappa - 2$ **then**
      `// Construction of (possible) tuples` $(p(\tilde{a}), s(\tilde{a}), k)$ `for` $k \geq \kappa - 2$
      Determine ξ' for $p(\tilde{a})$ such that $p(\tilde{a}) \in \tilde{L}^{(\kappa-2)}_{\xi'}$;
      if $s(\tilde{a}) < s(\xi')$ *or* $(p(\xi'), s(\xi'), k) = \emptyset$ **then**
        $D^{(k)} := D^{(k)} \setminus \{(p(\xi'), s(\xi'), k)\}$;
        $(p(\xi'), s(\xi'), k) := (p(\tilde{a}), s(\tilde{a}), k)$;
        Backtrack$(p(\xi'), s(\xi'), k) := (\tilde{a})$;
        $D^{(k)} := D^{(k)} \cup \{(p(\xi'), s(\xi'), k)\}$;
  Remove dominated tuples from $D^{(k)}$;

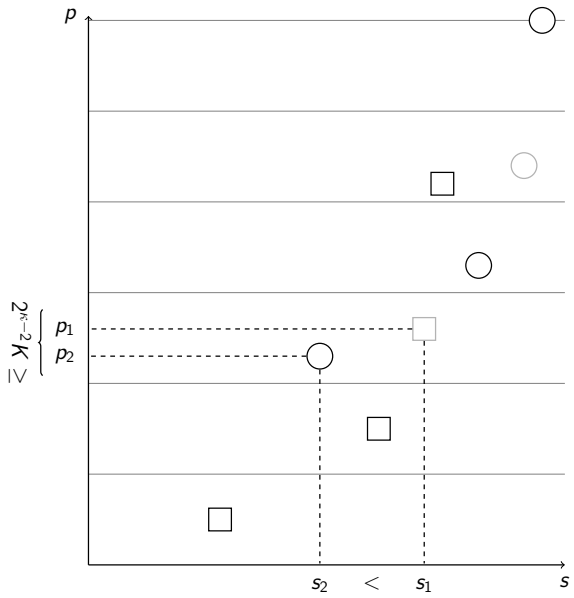


(a) The tuples in $D^{(k+1)}$

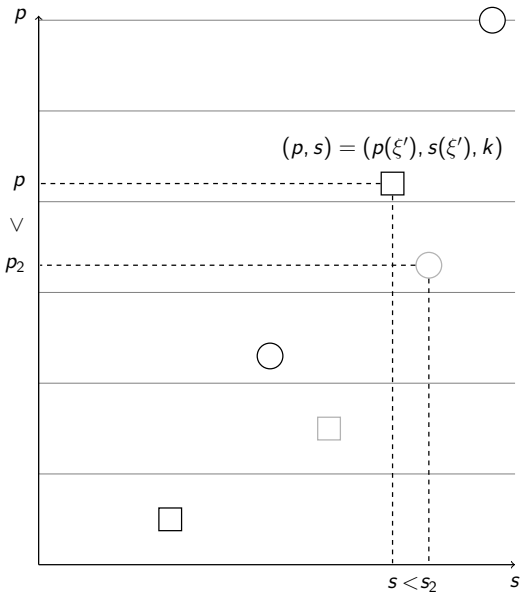


(b) The tuples in $D^{(k+1)}$ are kept. Additionally, new tuples are constructed with the items in $\tilde{I}^{(k)}$.

Figure 4.3: The principle of the approximate dynamic programming



(c) Only the tuple of smallest size is kept for every $\tilde{L}_{\xi}^{(\kappa-2)}$, which yields $\tilde{D}^{(k)}$,...



(d) ...and removing the dominated tuples yields $D^{(k)}$.

Figure 4.3: (Continued) The principle of the approximate dynamic programming

Theorem 4.22. *Let $k \in \{0, \ldots, \kappa+1\}$. For every (non-dominated) tuple $(\bar{p}, \bar{s}, k) \in F^{(k)}$, there is a tuple $(p, s, k) \in D^{(k)}$ such that*

$$p \geq \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa-k+1} \bar{p} \quad \text{and} \quad s \leq \bar{s} \ .$$

Proof. This statement is trivial for $(\bar{p}, \bar{s}, k) = (0, 0, k)$ because $(0, 0, k) \in D^{(k)}$ (this tuple is never removed in the construction of $F^{(k)}$ and $D^{(k)}$).

Suppose now that $(\bar{p}, \bar{s}, k) \neq (0, 0, k)$. The theorem is proved by induction for $k = \kappa+1, \ldots, 0$.

The statement is evident for $k = \kappa+1$. If $a_{\text{eff}-\text{c}}$ exists (i.e. enough copies of a_{eff} can be glued together without exceeding the capacity c), then $F^{(\kappa+1)} = D^{(\kappa+1)} = \{(0, 0, \kappa+1), (p(a_{\text{eff}-\text{c}}), s(a_{\text{eff}-\text{c}}), \kappa+1)\}$. If $a_{\text{eff}-\text{c}}$ does not exist, then we have $F^{(\kappa+1)} = D^{(\kappa+1)} = \{(0, 0, \kappa+1)\}$.

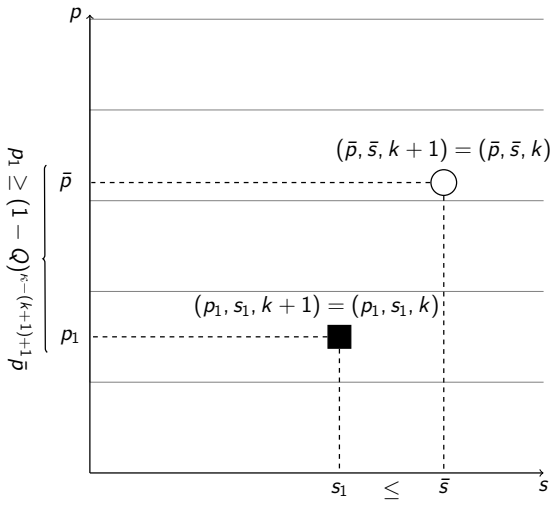
Suppose that the statement is true for $k+1, \ldots, \kappa+1$. As defined in Lemma 4.21, $\tilde{D}^{(k)}$ is the set $D^{(k)}$ before the dominated tuples are removed. Let $(\bar{p}, \bar{s}, k) \in F^{(k)}$.

There are two cases. In the first case, we have $(\bar{p}, \bar{s}, k+1) \in F^{(k+1)}$. By the induction hypothesis, there is a tuple $(p_1, s_1, k+1) \in D^{(k+1)}$ such that the inequalities $p_1 \geq \bar{p} \cdot (1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon})+1})^{\kappa-(k+1)+1}$ and $s_1 \leq \bar{s}$ hold (see Fig. 4.4(a)). Note that this implies $(p_1, s_1, k+1) \neq (0, 0, k+1)$ and therefore $p_1 \geq 2^{\kappa-2}T$ by Lemma 4.21. Let ξ_1 be the index such that $p_1 \in \tilde{L}_{\xi_1}^{(\kappa-2)}$. During the execution of Algorithm 4.3, $(p_1, s_1, k+1)$ yields the tuple (p_1, s_1, k), which may only be replaced in $\tilde{D}^{(k)}$ by a tuple of smaller size, but with a profit still in $\tilde{L}_{\xi_1}^{(\kappa-2)}$. Thus, there must be a tuple $(p_2, s_2, k) \in \tilde{D}^{(k)}$ with $s_2 \leq s_1$ and $p_2 \in \tilde{L}_{\xi_1}^{(\kappa-2)}$ (see Fig. 4.4(b)). Let now $(p, s, k) \in D^{(k)}$ be the tuple that dominates (p_2, s_2, k) (which can of course be (p_2, s_2, k) itself), i.e. $p \geq p_2$ and $s \leq s_2$ (see Fig. 4.4(c)). For the profit, we have

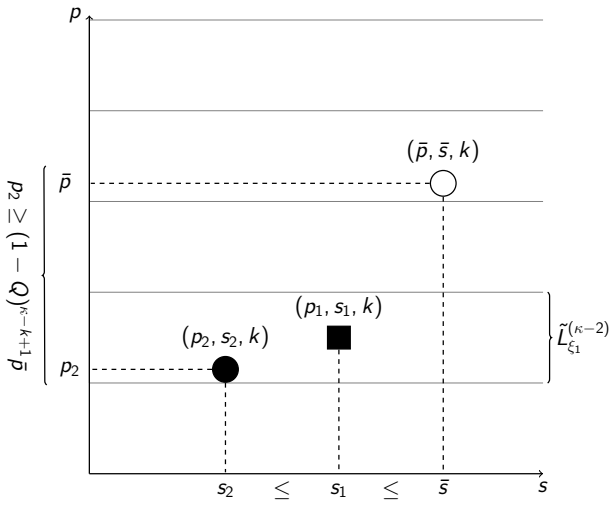
$$\begin{aligned} p \quad &\geq \quad p_2 \geq p_1 - 2^{\kappa-2}K \overset{p_1 \neq 0}{=} p_1 \cdot \left(1 - \frac{2^{\kappa-2}K}{p_1}\right) \overset{\text{Lem. 4.21}}{\geq} p_1 \cdot \left(1 - \frac{2^{\kappa-2}K}{2^{\kappa-2}T}\right) \\ &\overset{(4.1),(4.2)}{=} p_1 \cdot \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right) \geq \bar{p} \cdot \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa-k+1} . \end{aligned}$$

The lower bound on the profit is therefore true for (p, s, k). We have $s \leq s_2 \leq s_1 \leq \bar{s}$ for the bound on the size (see also Fig. 4.4(c)).

Consider now the second case where $(\bar{p}, \bar{s}, k) \in F^{(k)}$, but $(\bar{p}, \bar{s}, k+1) \notin F^{(k+1)}$. Therefore, $(\bar{p}, \bar{s}, k)$ is a new (non-dominated) tuple with $(\bar{p}, \bar{s}, k) = (\tilde{p} + p(\tilde{a}), \tilde{s} + s(\tilde{a}), k)$ for the right item $\tilde{a} \in \tilde{I}^{(k)}$ and tuple $(\tilde{p}, \tilde{s}, k+1) \in F^{(k+1)}$. By the induction hypothesis,

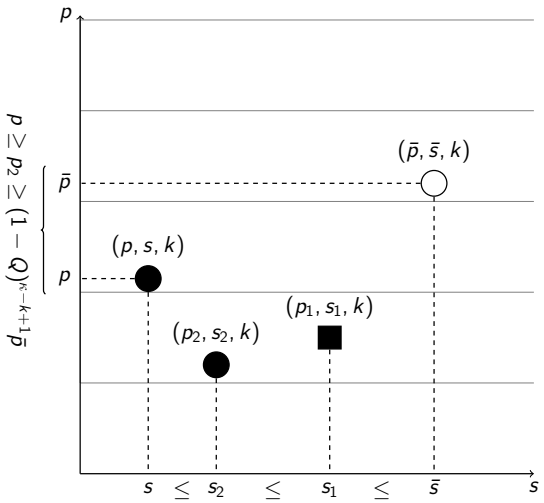


(a) Since $(\bar{p}, \bar{s}, k+1) \in F^{(k+1)}$, there must be a corresponding tuple $(p_1, s_1, k+1) \in D^{(k+1)}$ by the induction hypothesis whose profit can be bounded from below.



(b) By construction, there must be a tuple $(p_2, s_2, k) \in \tilde{D}^{(k)}$ with a profit in the same interval $\tilde{L}_{\xi_1}^{(\kappa-2)}$ as (p_1, s_1, k). This makes it possible to bound p_2 from below.

Figure 4.4: The first case of the proof for Theorem 4.22: we have $(\bar{p}, \bar{s}, k) \in F^{(k)}$ and also $(\bar{p}, \bar{s}, k+1) \in F^{(k+1)}$. We set $Q := (1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon})+1})$.



(c) There may be a tuple $(p, s, k) \in D^{(k)}$ that dominates (p_2, s_2, k). Since $p \geq p_2$, the bound still holds.

Figure 4.4: (Continued) The first case of the proof for Theorem 4.22: we have $(\bar{p}, \bar{s}, k) \in F^{(k)}$ and also $(\bar{p}, \bar{s}, k+1) \in F^{(k+1)}$. We set $Q := (1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon})+1})$.

there must be a tuple $(p_1, s_1, k+1) \in D^{(k+1)}$ such that $p_1 \geq \tilde{p} \cdot (1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon})+1})^{\kappa-(k+1)+1}$ and $s_1 \leq \tilde{s}$ (see Fig. 4.5(a)). Thus, the following inequality holds:

$$\begin{aligned} p_1 + p(\tilde{a}) &\geq p(\tilde{a}) + \tilde{p} \cdot \left(1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon})+1}\right)^{\kappa-(k+1)+1} \\ &\geq (p(\tilde{a}) + \tilde{p}) \cdot \left(1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon})+1}\right)^{\kappa-(k+1)+1} \\ &= \bar{p} \cdot \left(1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon})+1}\right)^{\kappa-(k+1)+1} . \end{aligned}$$

There are two possibilities: either $k \geq \kappa - 2$, i.e. $p(\tilde{a}) \geq 2^{\kappa-2}T$ holds, and $p_1 + p(\tilde{a}) \geq 2^{\kappa-2}T$ directly follows. Otherwise, we have $k \leq \kappa - 3$. Then, the identity $(\bar{p}, \bar{s}, k) = (\tilde{p} + p(\tilde{a}), \tilde{s} + s(\tilde{a}), k) \neq (0, 0, k)$ implies that $(\tilde{p}, \tilde{s}, k+1) \neq (0, 0, k+1)$ holds because the tuple $(0, 0, k+1)$ is not used to form any new tuple in $\tilde{D}^{(k)}$ and therefore in $D^{(k)}$. Because of $p_1 \geq \tilde{p} \cdot (1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon})+1})^{\kappa-(k+1)+1}$, this again implies that $p_1 \neq 0$ and therefore $p_1 + p(\tilde{a}) \geq p_1 \geq 2^{\kappa-2}T$ as seen in Lemma 4.21.

Thus, there is an index ξ_1 such that $p_1 + p(\tilde{a}) \in \tilde{L}_{\xi_1}^{(\kappa-2)}$. Similar to above, the tuple $(p_1 + p(\tilde{a}), s_1 + s(\tilde{a}), k)$ is formed during the construction of $\tilde{D}^{(k)}$ (see Fig. 4.5(b)). It

may only be replaced by a tuple of smaller size. Hence, there must be $(p_2, s_2, k) \in \tilde{D}^{(k)}$ with $p_2 \in \tilde{L}_{\xi_1}^{(\kappa-2)}$. Let $(p, s, k) \in D^{(k)}$ be the tuple that dominates (p_2, s_2, k) (see Fig. 4.5(c)). We get

$$\begin{aligned} p \quad &\geq \quad p_2 \geq p_1 + p(\tilde{a}) - 2^{\kappa-2}K \overset{p_1+p(\tilde{a})\neq 0}{=} (p_1 + p(\tilde{a})) \cdot \left(1 - \frac{2^{\kappa-2}K}{p_1 + p(\tilde{a})}\right) \\ &\overset{p_1+p(\tilde{a})\geq 2^{\kappa-2}T}{\geq} (p_1 + p(\tilde{a})) \cdot \left(1 - \frac{2^{\kappa-2}K}{2^{\kappa-2}T}\right) \overset{(4.1),(4.2)}{=} (p_1 + p(\tilde{a})) \cdot \left(1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right) \\ &\geq \quad \bar{p} \cdot \left(1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa-k+1} . \end{aligned}$$

We have similar to above $s \leq s_2 \leq s_1 + s(\tilde{a}) \leq \tilde{s} + s(\tilde{a}) = \bar{s}$ for the bound on the size (see also Fig. 4.5(c)). □

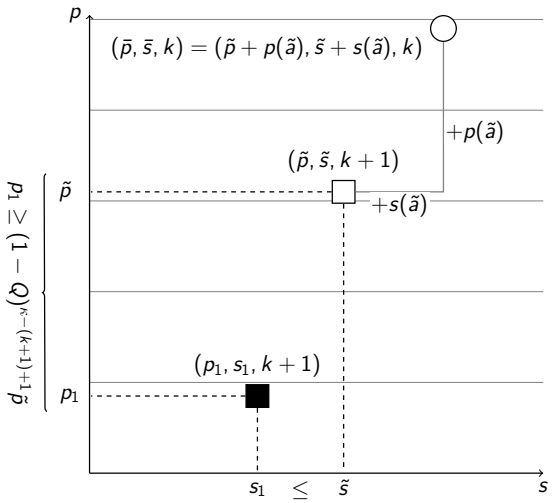
Remark 4.23. As can be seen, the proof of Theorem 4.22 is only possible because it is guaranteed that p_1 or $p_1 + p(\tilde{a})$ is at least $2^{\kappa-2}T$. In fact, this is achieved by the construction of the glued item set $\tilde{I}$ with its structured solutions (Definition 4.11 and Theorem 4.12). Hence, we can prove Lemma 4.14, and with the introduction of $a_{\text{eff}-\text{c}}$, we have structured solutions with a lower bound (Definition 4.16 and Theorem 4.17). This shows that $p_1 \geq 2^{\kappa-2}T$ or $p_1 + p(\tilde{a}) \geq 2^{\kappa-2}T$ (see also Lemma 4.20 and 4.21). Without the structure, a dynamic program like Algorithm 4.3 would also have to generate tuples (p, s, k) with $p < 2^{\kappa-2}T$ for $k \leq \kappa - 3$. Hence, we would need for the same approximation ratio profit sub-intervals like $\tilde{L}_{\xi}^{(\kappa-2)}$ with a smaller length than $2^{\kappa-2}K$, and we would have to save more tuples. Both would increase the asymptotic running time and space complexity as can be seen in the proof of Theorem 4.25.

Corollary 4.24. *For every $v \leq c$, there is a tuple $(p, s, 0) \in D^{(0)}$ such that $s \leq v$ and*

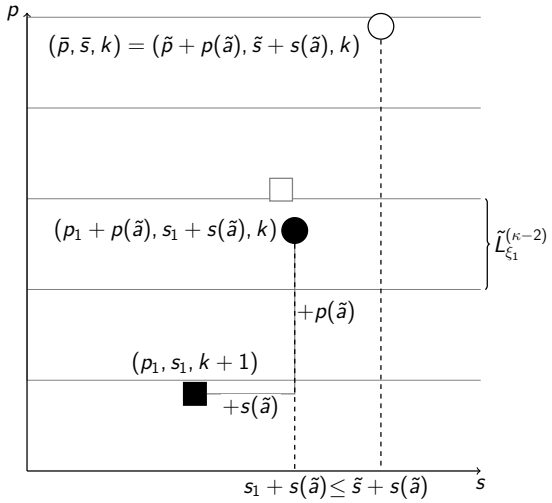
$$p \geq \left(1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa+1} \text{OPT}_{\text{St}}\left(\tilde{I} \cup \{a_{\text{eff}-\text{c}}\}, v\right) .$$

Proof. Lemma 4.20 states that there is a $(\bar{p}, \bar{s}, 0) \in F^{(0)}$ with $\bar{p} = \text{OPT}_{\text{St}}(\tilde{I} \cup \{a_{\text{eff}-\text{c}}\}, v)$ and $\bar{s} \leq v$. Theorem 4.22 implies that there is a tuple $(p, s, 0) \in D^{(0)}$ with the desired property. □

Theorem 4.25. *Algorithm 4.3 constructs all tuple sets $D^{(k)}$ for $k = \kappa + 1, \ldots, 0$ in time $\mathcal{O}(\frac{1}{\varepsilon^2}\log^3\frac{1}{\varepsilon})$. The space needed for the algorithm and to save the $D^{(k)}$ as well as the backtracking information is in $\mathcal{O}(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})$.*

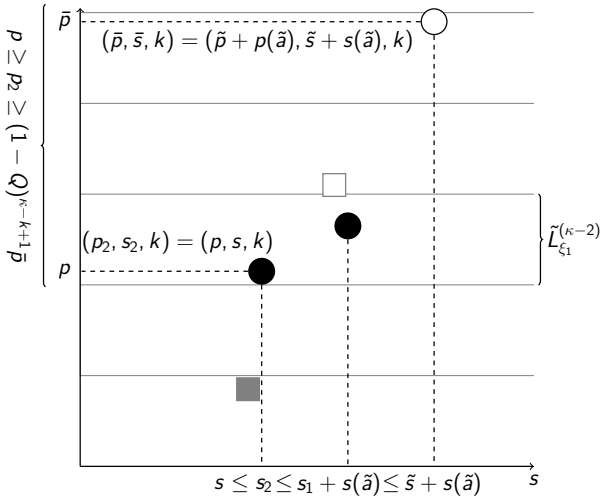


(a) Since $(\bar{p}, \bar{s}, k+1) \notin F^{(k+1)}$, there must be an item $\tilde{a}$ s.t. $(\bar{p}, \bar{s}, k) = (\tilde{p} + p(\tilde{a}), \tilde{s} + s(\tilde{a}), k)$ for a tuple $(\tilde{p}, \tilde{s}, k+1) \in F^{(k+1)}$. By the induction hypothesis, there must be a tuple $(p_1, s_1, k+1) \in D^{(k+1)}$ corresponding to $(\tilde{p}, \tilde{s}, k+1)$ whose profit can be bounded from below.



(b) The tuple $(p_1 + p(\tilde{a}), s_1 + s(\tilde{a}), k)$ is constructed during the execution of the dynamic program.

Figure 4.5: The second case of the proof for Theorem 4.22: we have $(\bar{p}, \bar{s}, k) \in F^{(k)}$, but $(\bar{p}, \bar{s}, k+1) \notin F^{(k+1)}$. We set $Q := (1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon})+1})$.



(c) As in the first case, there must be a tuple $(p, s, k) \in D^{(k)}$ whose profit can be bounded as desired. Here, (p_2, s_2, k) is not dominated, i.e. $(p, s, k) = (p_2, s_2, k)$.

Figure 4.5: (Continued) The second case of the proof for Theorem 4.22: we have $(\bar{p}, \bar{s}, k) \in F^{(k)}$, but $(\bar{p}, \bar{s}, k+1) \notin F^{(k+1)}$. We set $Q := (1 - \frac{\varepsilon}{4}\frac{1}{\log_2(\frac{2}{\varepsilon})+1})$.

Proof. Let us first bound the space complexity. The profit interval $[\frac{1}{4}P_0, 2P_0]$ is partitioned into $\mathcal{O}(\xi_0)$ intervals $\tilde{L}^{(\kappa-2)}_{\xi}$. The set $D^{(k)}$ saves at most one tuple with the corresponding backtracking information for every $\tilde{L}^{(\kappa-2)}_{\xi}$ or the information that a tuple does not exist. Thus, the space needed for all $D^{(k)}$ and the corresponding backtracking data is in $\mathcal{O}(\kappa \cdot \xi_0) = \mathcal{O}(\kappa \cdot (\kappa 2^\kappa)) = \mathcal{O}(\log(\frac{1}{\varepsilon}) \cdot (\log(\frac{1}{\varepsilon})\frac{1}{\varepsilon})) = \mathcal{O}(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})$. All other information of the algorithm is only temporarily saved and needs $\mathcal{O}(1)$.

The loops dominate the running time. Apart from removing the dominated tuples, they need in total

$$\begin{aligned}\mathcal{O}\left(\kappa \cdot \left(\xi_0 + \xi_0 \cdot |\tilde{I}^{(k)}| + |\tilde{I}^{(k)}|\right)\right) &\overset{\text{Thm. 4.18}}{=} \mathcal{O}\left(\log\left(\frac{1}{\varepsilon}\right)\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\varepsilon}\right) \cdot \frac{1}{\varepsilon}\log\left(\frac{1}{\varepsilon}\right)\right)\right) \\ &= \mathcal{O}\left(\frac{1}{\varepsilon^2}\log^3\frac{1}{\varepsilon}\right) \quad .\end{aligned}$$

As seen in Lemma 3.7 and stated in [63], non-dominated tuples (p, s, k) can be removed in linear time in the number of tuples if the tuples are different and sorted by profit. This is the case because every tuple in $D^{(k)}$ is stored in an array sorted according to the corresponding ξ. The total time to remove the dominated tuples from all $D^{(k)}$

is therefore in $\mathcal{O}(\kappa \cdot \xi_0) = \mathcal{O}(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})$, which is dominated by the overall running time. □

4.8 The Complete Algorithm

We can now put together the entire approximation algorithm.

Algorithm 4.4: The complete algorithm

Input: Item set I
Output: Profit P, solution set J
Determine P_0 and define T, K;
Partition the items into I_L and I_S and find a_{eff};
if *Item a with $p(a) = 2P_0$ found during the partitioning* **then**
  return $2P_0$, $\{a\}$;
Reduce I_L to $I_{L,\text{red}}$ with Algorithm 4.1;
Construct $\tilde{I}$ with Algorithm 4.2 and the item $a_{\text{eff}-c}$;
if $p(\tilde{a}_0^{(\kappa)}) = P_0$ *and* $s(\tilde{a}_0^{(\kappa)}) \leq \frac{c}{2}$ **then**
  Recursively undo the gluing of $\tilde{a}_0^{(\kappa)}$ to get the item set J'. Let J be the set consisting of two copies of every item in J';
  return $2P_0$, J ;
Construct with Algorithm 4.3 the tuple sets $D^{(\kappa+1)}, \ldots, D^{(0)}$;
Find $(p, s, 0) \in D^{(0)}$ such that
$P := p + \text{OPT}(\{a_{\text{eff}}\}, c - s) = \max_{(p',s',0)\in D^{(0)}} p' + \text{OPT}(\{a_{\text{eff}}\}, c - s')$;
Backtrack the tuple $(p, s, 0)$ to find the large items $J' \subset \tilde{I} \cup \{a_{\text{eff}-c}\}$ of the corresponding structured solution with a lower bound;
Recursively undo the gluing of all $\tilde{a} \in J'$ and add these items to the solution set J;
Add the items of $\text{OPT}(\{a_{\text{eff}}\}, c - s)$ to J;
return P, J ;

Theorem 4.26. *Algorithm 4.4 finds a solution of value at least* $(1 - \varepsilon)\text{OPT}(I)$.

Proof. The algorithm returns a feasible solution: $(p, s, 0)$ represents an item set of size s. If items $\tilde{a} \in \tilde{I}$ derived from gluing are part of the solution, their ungluing does not change the total size nor the total profit (see Remark 4.10).

We prove the solution quality. First, the algorithm considers the two special cases listed at the beginning of Section 4.7. Each of them returns a solution of profit $2P_0$ so that $\text{OPT}(I) = 2P_0$ (see Theorem 4.3 and Lemma 4.13). If the special cases do not

yield a solution, we are in the third case. Let v be the volume from Theorem 4.17. Corollary 4.24 guarantees the existence of one $(p, s, 0) \in D^{(0)}$ with $s \leq v$ such that

$$p \geq \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa+1} \mathrm{OPT}_{\mathrm{St}}\left(\tilde{I} \cup \{a_{\mathrm{eff-c}}\}, v\right) \ .$$

Moreover, we have $\mathrm{OPT}(\{a_{\mathrm{eff}}\}, c - s) \geq \mathrm{OPT}(\{a_{\mathrm{eff}}\}, c - v)$ because $c - s \geq c - v$. Thus, the following inequality holds for this $(p, s, 0)$:

$$\begin{aligned}
& p + \mathrm{OPT}\left(\{a_{\mathrm{eff}}\}, c - s\right) \\
& \quad \geq \quad \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa+1} \mathrm{OPT}_{\mathrm{St}}\left(\tilde{I} \cup \{a_{\mathrm{eff-c}}\}, v\right) + \mathrm{OPT}\left(\{a_{\mathrm{eff}}\}, c - v\right) \\
& \quad \geq \quad \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa+1} \left(\mathrm{OPT}_{\mathrm{St}}\left(\tilde{I} \cup \{a_{\mathrm{eff-c}}\}, v\right) + \mathrm{OPT}\left(\{a_{\mathrm{eff}}\}, c - v\right)\right) \\
& \overset{\text{Thm. 4.17}}{\geq} \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{2\kappa+2} \mathrm{OPT}(I) - \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{\kappa+1} T \\
& \quad \geq \quad \left(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon}) + 1}\right)^{2\kappa+2} \mathrm{OPT}(I) - T \\
& \overset{(4.1)}{\geq} \left(1 - \frac{\varepsilon}{4} \frac{2\kappa + 2}{\log_2(\frac{2}{\varepsilon}) + 1}\right) \mathrm{OPT}(I) - \frac{1}{2} \varepsilon P_0 \\
& \quad \geq \quad \left(1 - \frac{\varepsilon}{4} \cdot \frac{2 \cdot \left(\log_2(\frac{2}{\varepsilon}) + 1\right)}{\log_2(\frac{2}{\varepsilon}) + 1}\right) \mathrm{OPT}(I) - \frac{1}{2} \varepsilon \, \mathrm{OPT}(I) \\
& \quad = \quad (1 - \varepsilon) \, \mathrm{OPT}(I) \ .
\end{aligned}$$

Taking the maximum over all $(p, s, 0) \in D^{(0)}$ therefore yields the desired solution. Note that we have used $(1 - \delta)^k \geq (1 - k \cdot \delta)$ for $\delta < 1$. □

Remark 4.27. The total bound on the approximation ratio is mainly due to the exponent $2\kappa + 2$, i.e. that we make the multiplicative error of $(1 - \frac{\varepsilon}{4} \frac{1}{\log_2(\frac{2}{\varepsilon})+1})$ only $2\kappa + 2$ times. Such an error occurs when I is replaced by $I_{L,\mathrm{red}}$ at the beginning (Lemma 4.6), in each of the κ iterations in which $\tilde{I}$ is constructed (Theorem 4.12), and in $\kappa + 1$ of the $\kappa + 2$ iterations of the dynamic program (Theorem 4.22 and Corollary 4.24). The error of the dynamic program can be bounded because the structured solutions with a lower bound have at least one item of profit at least $2^{\kappa-2}T$ (see the second property of Definition 4.16 and Remark 4.23).

Theorem 4.28. *The algorithm has a running time in* $\mathcal{O}(n + \frac{1}{\varepsilon^2} \log^3 \frac{1}{\varepsilon})$ *and needs space in* $\mathcal{O}(n + \frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})$.

Proof. Determining P_0, constructing I_L and I_S as well as finding a_{eff} can all be done in time and space $\mathcal{O}(n)$ as stated in Theorem 4.3 and 4.4. The definition of T and K in time and space $\mathcal{O}(1)$ is obvious. It is also clear that an item $p(a) = 2P_0$ can directly be found during the construction of I_L such that the first if-condition does not influence the asymptotic running time.

Algorithm 4.1 returns the set $I_{L,\text{red}}$ in time $\mathcal{O}(n + \frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})$ and space $\mathcal{O}(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})$ (see Theorem 4.7).

Algorithm 4.2 constructs the $\tilde{I}^{(k)}$ and $\tilde{I}$ in time $\mathcal{O}(\frac{1}{\varepsilon^2}\log^3\frac{1}{\varepsilon})$ and space $\mathcal{O}(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})$ as explained in Theorem 4.18, which clearly dominates the construction of $a_{\text{eff}-\text{c}}$ in $\mathcal{O}(1)$.

The second if-condition can be checked in $\mathcal{O}(1)$. The running time for undoing the gluing will be determined at the end of the proof.

Algorithm 4.3 constructs the sets $D^{(k)}$ in time $\mathcal{O}(\frac{1}{\varepsilon^2}\log^3\frac{1}{\varepsilon})$ and space $\mathcal{O}(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})$ (see Theorem 4.25). For one tuple $(p', s', 0)$, the corresponding OPT $(\{a_{\text{eff}}\}, c - s')$ can be found in $\mathcal{O}(1)$ by computing $\lfloor\frac{c-s'}{s(a_{\text{eff}})}\rfloor \cdot p(a_{\text{eff}})$. Thus, finding the best tuple $(p, s, 0)$ can be done in $\mathcal{O}(|D^{(0)}|) = \mathcal{O}(\xi_0) = \mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$. Since only the currently best tuple $(p, s, 0)$ has to be saved, the space needed is in $\mathcal{O}(1)$.

The backtracking for the tuple $(p, s, 0)$ needs time in $\mathcal{O}(\kappa) = \mathcal{O}(\log\frac{1}{\varepsilon})$: the backtracking information Backtrack(p', s', k) for $k = 0, \ldots, \kappa + 1$ states whether the tuple (p', s', k) was formed by adding an item $\tilde{a} \in \tilde{I}^{(k)}$ and with which tuple $(p'', s'', k + 1)$ to continue. Hence, the item set J' also has at most $\mathcal{O}(\log\frac{1}{\varepsilon})$ items in $\tilde{I} \cup \{a_{\text{eff}-\text{c}}\}$, which bounds the storage space needed.

To conclude, the time and space for the ungluing still have to be bounded. Consider one item $\tilde{a} \in \tilde{I}$. The backtracking information Backtrack$(\tilde{a})$ returns two items $(\bar{a}_1, \bar{a}_2)$ (with $\bar{a}_1, \bar{a}_2 \in I_{L,\text{red}} \cup \tilde{I}$) on which the backtracking can be recursively applied. The recursive ungluing of the items can be represented as a binary tree where the root is the original item $\tilde{a}$ and the (two) children of each node are the items $(\bar{a}', \bar{a}'')$ returned by the backtracking information. The leaves of the tree are the original items in $I_{L,\text{red}}$. This binary tree obviously has a height in $\mathcal{O}(\kappa)$ because the children $(\bar{a}', \bar{a}'')$ for one $\bar{a} \in \tilde{I}^{(k)}$ are in $\tilde{I}^{(k-1)} \cup I_{L,\text{red}}$. A binary tree of height $\mathcal{O}(\kappa) = \mathcal{O}(\log\frac{1}{\varepsilon})$ has at most $\mathcal{O}(\frac{1}{\varepsilon})$ nodes. The backtracking and ungluing of $\tilde{a}$ can therefore be done in time and space in $\mathcal{O}(\frac{1}{\varepsilon})$, which also includes saving the items $\bar{a} \in I_{L,\text{red}}$ of which $\tilde{a}$ is composed. Since J' has $\mathcal{O}(\log\frac{1}{\varepsilon})$ items, the original items $I_{L,\text{red}}$ of the approximate solution can be found in time and space $\mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$. This also dominates the time to undo the gluing of $\tilde{a}_0^{(\kappa)}$ should the body of the second if-condition be executed.

Similar to above, the number of items a_{eff} for OPT $(\{a_{\text{eff}}\}, c - s)$ can be found in $\mathcal{O}(1)$. To sum up, Algorithm 4.4 has the stated running time and space complexity.□

4.9 Column Generation for Strip Packing

We apply the new FPTAS for UKP to Strip Packing (SP).

Known Results Well-known approaches for Strip Packing are shelf-based algorithms like Next Fit Decreasing Height (NFDH) and First Fit Decreasing Height (FFDH). The overview by Coffman et al. [12] (see also [48]) shows that NFDH finds a solution with $NFDH(I) \leq 2 \cdot \mathrm{OPT}(I) + 1$, and FFDH satisfies $FFDH(I) \leq 1.7 \cdot \mathrm{OPT}(I) + 1$. Both algorithms have an absolute approximation ratio of 3 and 2.7, respectively [9, 12]. The algorithm by Sleator [83] has an approximation guarantee of $2 \cdot \mathrm{OPT}(I) + \frac{1}{2} \max_j h(a_j)$ and therefore an absolute ratio of 2.5 and an asymptotic ratio of 2. Later, Schiermeyer [76] and Steinberg [84] both independently found an algorithm with an absolute approximation ratio of 2. Finally, the algorithm by Harren et al. [34] has the currently best absolute approximation ratio of $\frac{5}{3} + \varepsilon$ for $\varepsilon > 0$. On the other hand, the lower bound on the absolute approximation ratio is $\frac{3}{2}$ for every polynomial-time algorithm unless P = NP.

Jansen and Solis-Oba [53] presented an APTAS with an approximation guarantee of $(1+\varepsilon)\mathrm{OPT}(I) + 1$ for $\varepsilon > 0$. The first AFPTAS was found even earlier by Kenyon and Rémila [62] with the (larger) additive term $f(\frac{1}{\varepsilon}) \in \mathcal{O}(\frac{1}{\varepsilon^2})$. Bougeret et al. [9] and Sviridenko [86] independently improved the additive term with their AFPTAS to $f(\frac{1}{\varepsilon}) \in \mathcal{O}(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$. The running time for the algorithm in [9] can be bounded by $\mathcal{O}(\frac{1}{\varepsilon^6} \log(\frac{1}{\varepsilon}) + n \log n)$, which is the currently fastest known AFPTAS.

Our Result Similar to Theorem 2.5, the Strip Packing algorithm in [9] (see also [41]) has in fact a running time in $\mathcal{O}(d(\frac{1}{\varepsilon^2} + \ln d) \max\{UKP(d, \frac{\bar{\varepsilon}}{6}), d \ln \ln(\frac{d}{\varepsilon})\} + n \log n)$ where $d \in \mathcal{O}(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ and $\bar{\varepsilon} \in \Theta(\varepsilon)$. The new FPTAS for UKP yields the following improved AFPTAS:

Corollary 4.29. *There is an AFPTAS $(A_\varepsilon)_{\varepsilon>0}$ for Strip Packing that finds a packing for I of total height $A_\varepsilon(I) \leq (1+\varepsilon)\mathrm{OPT}(I) + \mathcal{O}(\frac{1}{\varepsilon} \log(\frac{1}{\varepsilon}))$. Its running time is in*

$$\mathcal{O}\left(\frac{1}{\varepsilon^5} \log^4 \frac{1}{\varepsilon} + n \log n\right) \quad .$$

5 A Faster FPTAS for the Unbounded Knapsack Problem with Inversely Proportional Profits

As stated in the introduction, this chapter combines the results in Chapter 3 and 4 to derive a faster FPTAS for the Unbounded Knapsack Problem with Inversely Proportional Profits (UKPIP). This allows us to again improve the running time of the AFPTAS for the Variable-sized Bin Packing Problem (VBP).

Note that the chapter is almost self-contained. Hence, many of the definitions, proofs and results in this chapter overlap with the ones in the previous two chapters. Similar to Chapter 3, we introduce the threshold $T_b > 0$ and a modified constant $K_b > 0$ for every set of knapsacks C_b, which will be determined at the end, and not be fixed at the beginning like in Chapter 4. This will also explain how the corresponding values for T and K in Chapter 4 were derived.

5.1 Our Result

We first state the improved FPTAS.

Theorem 5.1. *Let $\varepsilon > 0$. Let $c_{\min}$ be the smallest knapsack size of a UKPIP instance with M knapsacks and n item types. There is an FPTAS that solves this problem in time*

$$\mathcal{O}\Bigg(n \log M + M \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon} + \min\left\{\left\lfloor \log \frac{1}{c_{\min}} \right\rfloor + 1, M\right\} \frac{1}{\varepsilon^2} \log^3 \frac{1}{\varepsilon} + \min\left\{\left\lfloor \log \frac{1}{c_{\min}} \right\rfloor + 1, M\right\} \cdot n\Bigg)$$

and space $\mathcal{O}(M + n + \frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})$.

This new algorithm together with Theorem 2.4 proves

Theorem 2.1. *There is an AFPTAS $(A_\varepsilon)_{\varepsilon>0}$ for Variable-sized Bin Packing that finds for $\varepsilon \in (0, \frac{1}{2}]$ a packing of an instance (I, C) in $A_\varepsilon(I) \leq (1+\varepsilon)\mathrm{OPT}(I, C) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$ bins. Its running time is in*

$$\mathcal{O}\left(\frac{1}{\varepsilon^5} \log^5 \frac{1}{\varepsilon} + M + \log\left(\frac{1}{\varepsilon}\right) n\right) \quad .$$

For UKPIP, a preprocessing of the knapsack sizes allows to get a running time that does not depend on $c_{\min}$ and is only linear in M.

Corollary 5.2. *Let $\varepsilon > 0$. There is an FPTAS for UKPIP with a running time in*

$$\mathcal{O}\left(\frac{1}{\varepsilon^2}\log^4\frac{1}{\varepsilon} + M + \log\left(\frac{1}{\varepsilon}\right)n\right)$$

and a space complexity in $\mathcal{O}(M + n + \frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})$.

5.2 Overview

Like in Chapter 3, the optimum for knapsack size c_l with the basic profits p_j is denoted by an additional bar over the corresponding expression. Hence, the value $\overline{\mathrm{OPT}}(I, v)$ stands for the optimum value obtained with the basic profits p_j if a capacity $v \leq 1$ is considered. In a slight deviation to Chapter 3, we directly write $\overline{\mathrm{OPT}}(I, c_l)$ instead of $\overline{\mathrm{OPT}}_{c_l}(I)$.

As stated in Section 3.4, it is sufficient to find an implementation of `MaxSolution`, i.e. to determine an approximation to every $\overline{\mathrm{OPT}}(I, c_l)$.

Section 5.4 presents how the greedy approach of Subsection 3.5.2 is implemented for UKPIP. The greedy algorithm finds approximations $\bar{P}_{c_l} \geq \frac{1}{2}\overline{\mathrm{OPT}}(I, c_l)$ for every knapsack c_l. Because of the scaling property of Section 3.4, we (again) get the approximations $P_{c_l} = \frac{\bar{P}_{c_l}}{c_l} \geq \frac{1}{2}\mathrm{OPT}_{c_l}(I)$ and $P_0 = \max_{l\in\{1,\dots,M\}} P_{c_l} \geq \frac{1}{2}\mathrm{OPT}(I)$. Moreover, the special item sets S_l are again used, which are the items of size $c_{l-1} < s(a) \leq c_l$, and the derived sets $\bar{S}_l = \bigcup_{l'=1}^{l} S_{l'}$, which are the items that fit into c_l, i.e. $s(a) \leq c_l$. Finally, knapsack sizes with $P_{c_l} < \frac{1}{2}P_0$ are discarded.

In Subsection 5.5.1, we divide C into subsets $C_b = \{c_l \in C \mid c_l \in (2^{-(b+1)}, 2^{-b}]\}$ of similar size. The knapsacks $c_{\min}^{(b)}$ and $c_{\max}^{(b)}$ are the smallest and largest knapsack in C_b, respectively. We assume moreover without loss of generality that $\varepsilon = \frac{1}{2^{\kappa-1}} \leq \frac{1}{4}$. The threshold T_b and constant K_b for C_b are introduced, but not explicitly defined yet. Subsection 5.5.2 presents the division of the items into the set of large items $I_L^{(b)}$ (with $p_j \geq T_b$ and $s_j \leq c_{\max}^{(b)}$) and the set of small items $I_S^{(l)}$ for $c_l \in C_b$ (with $p_j < T_b$ and $s_j \leq c_l$). It is sufficient to keep for every $c_l \in C_b$ only the most efficient small item $a_{\mathrm{eff}}^{(l)}$. These items are collected in the set $I_{S,\mathrm{eff}}^{(b)}$. The basic idea of the FPTAS is (again) to find for each c_l the best combination of large and small items. In fact, $\overline{\mathrm{OPT}}(I, c_l) = \max_{0\leq v\leq c_l} \overline{\mathrm{OPT}}(I_L^{(b)}, v) + \overline{\mathrm{OPT}}\left(\{a \mid p(a) < T_b\}, c_l - v\right)$ holds. The solution values $\overline{\mathrm{OPT}}(I_L^{(b)}, v)$ to the large items are approximated by dynamic programming and the small items (in fact, copies of $a_{\mathrm{eff}}^{(l)}$) are added greedily to the volume $c_l - v$. As in Chapter 3, redundancy is avoided by determining for all knapsacks $c_l \in C_b$ at once the solutions of the dynamic program.

Fix one C_b. Subsection 5.5.3 introduces $\kappa^{(b)}_{\min}$ and $\kappa^{(b)}_{\max}$, the exponents such that $\bar{P}_{c^{(b)}_{\min}} \in [2^{\kappa^{(b)}_{\min}} T_b, 2^{\kappa^{(b)}_{\min}+1} T_b)$ and $\bar{P}_{c^{(b)}_{\max}} \in [2^{\kappa^{(b)}_{\max}} T_b, 2^{\kappa^{(b)}_{\max}+1} T_b)$. It is shown that a subset $I^{(b)}_{L,\text{red}} \subseteq I^{(b)}_L$ is sufficient to find an approximate solution (similar to Section 4.5), which reduces the overall time and space complexity. Like in Section 4.5, the interval of large item profits $[T_b, 2\bar{P}_{c^{(b)}_{\max}}]$ is first partitioned into $L^{(k,b)} = [2^k T_b, 2^{k+1} T_b)$ for $k \in \{0, \dots, \kappa^{(b)}_{\max} + 1\}$ and each $L^{(k,b)}$ into $\gamma_0 \in \mathbb{N}$ sub-intervals $L^{(k,b)}_\gamma$ of length $2^k K_b$. For $I^{(b)}_{L,\text{red}}$, only the smallest item $a^{(k,b)}_\gamma$ is kept of every $L^{(k,b)}_\gamma$.

Section 5.6 creates like in Section 4.6 a set of items with a special solution structure. The items in $I^{(b)}_{L,\text{red}}$ with profits in $L^{(k,b)}$ are denoted by $I^{(k,b)}$ for $k \in \{0, \dots, \kappa^{(b)}_{\max} + 1\}$. Starting with $\tilde{I}^{(0,b)} = I^{(0,b)}$, the items in $\tilde{I}^{(k,b)}$ are iteratively glued together to larger items $\tilde{\tilde{I}}^{(k+1,b)}$. The next set $\tilde{I}^{(k+1,b)}$ is then obtained by keeping from $\tilde{\tilde{I}}^{(k+1,b)} \cup I^{(k+1,b)}$ only the smallest item $\tilde{a}^{(k+1,b)}_\gamma$ of each profit sub-interval $L^{(k+1,b)}_\gamma$. The sets $\tilde{I}^{(k,b)}$ form the new set $\tilde{I}^{(b)}$, which has approximate *structured* solutions for $k = \kappa^{(b)}_{\max} + 1$: at most one large item of every $\tilde{I}^{(k,b)}$ is used. The value $\overline{\text{OPT}}_{\leq \kappa^{(b)}_{\max}+1}(\tilde{I}^{(b)}, v)$ denotes the optimum profit of these structured solutions for the knapsack volume $v \leq c^{(b)}_{\max}$. (For the analysis, $\overline{\text{OPT}}_{\leq k_0}(\cdot, v)$ is introduced for structured solutions for $k = k_0$ that use at most one item in $\tilde{I}^{(k,b)}$ for every $k \in \{0, \dots, k_0\}$, where $k_0 \leq \kappa^{(b)}_{\max} + 1$.) For each $a^{(l)}_{\text{eff}}$, a large item $a^{(l)}_{\text{eff}-\text{c}}$ that consists of several copies of $a^{(l)}_{\text{eff}}$ is introduced. A subset $I^{(b)}_{S,\text{red}-\text{c}}$ of the items $a^{(l)}_{\text{eff}-\text{c}}$ is kept. It is proved that there are approximate *structured solutions* to $\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{red}-\text{c}}$ *with a lower bound* (on the profit). They use at most one item of every set $\tilde{I}^{(0,b)}, \dots, \tilde{I}^{(\kappa^{(b)}_{\max}+1,b)}, I^{(b)}_{S,\text{red}-\text{c}}$. Additionally, these solutions have at least one item of profit at least $2^{\kappa^{(b)}_{\min}-2} T_b$. The value $\overline{\text{OPT}}_{\text{St}}(\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{red}-\text{c}}, v)$ is the optimum profit for the large items with this structure for the knapsack volume $v \leq c^{(b)}_{\max}$.

Section 5.7 introduces approximate dynamic programming to find the large items in a solution with the structure above. It is slightly modified compared to Section 4.7: instead of dividing the interval of possible profits into sub-intervals of equal length, the division into the sub-intervals $L^{(\bar{k},b)}_\gamma$ of variable length $2^{\bar{k}} K_b$ is re-used, and the dynamic program keeps in each iteration k the tuple (p, s, k) of profit $p \in L^{(\bar{k},b)}_\gamma$ that has the smallest size s. The fact that the tuples have profits of at least $2^{\kappa^{(b)}_{\min}-2} T_b$ and that at most one item of every set $\tilde{I}^{(0,b)}, \dots, \tilde{I}^{(\kappa^{(b)}_{\max}+1,b)}, I^{(b)}_{S,\text{red}-\text{c}}$ is used allows to reduce the time and space complexity, similar to Chapter 4. Interestingly, the modified approximate dynamic program does not need that $p \geq 2^{\kappa^{(b)}_{\min}-2} T_b$ for the approximation ratio (see Remark 5.37).

Finally, Section 5.8 puts the entire algorithm together. For each $c_l \in C_b$, it returns $\max_{(p,s,0)} p + \overline{\text{OPT}}(\{a^{(l)}_{\text{eff}-\text{c}}\}, c_l - s)$, and Subsection 5.8.1 shows how T_b and K_b have

to be set such that this is at least $(1-\varepsilon)\overline{\mathrm{OPT}}\,(I, c_l)$ for each $c_l \in C_b$. The algorithm is therefore an implementation of `MaxSolution`. The overall time and space complexity are bounded in Subsection 5.8.2.

This chapter is concluded with Section 5.9. It compares the constants T_b and K_b to the constants T and K of Chapter 4, and the representation of the solution as a multi-set is briefly discussed.

5.3 Notation and Remarks

The notation follows the one presented in Section 3.3. The only slight deviation is that $\overline{\mathrm{OPT}}_{c_l}(I)$ is written as the (equivalent) $\overline{\mathrm{OPT}}\,(I, c_l)$. Remember that $\overline{\mathrm{OPT}}\,(I, v) = \max\{\sum_a p(a)x_a \mid \sum_a s(a)x_a \leq v; x_a \in \mathbb{N} \text{ for } a \in I\}$.

We (still) assume that basic arithmetic operations as well as computing the logarithm can be performed in $\mathcal{O}(1)$.

Finally, we have a remark about the use of "item" and "item copy" when we consider a solution to a UKPIP instance, similar to Chapter 4.

Remark 5.3. Let $I, \tilde{I}$ be two sets of knapsack items with $\tilde{I} \subseteq I$. In 0-1 KPIP, a sentence like "the solution to I uses at most one item in $\tilde{I}$" is obvious: if the solution uses one item in $\tilde{I}$, all other items of the solution are in $I \setminus \tilde{I}$.

Consider now UKPIP. When we talk about solutions, we would formally have to distinguish between an item $a' \in I$ in the instance and the item copies of a' that a solution $V = \{x_a : a \mid a \in I, x_a \in \mathbb{N}\}$ uses. Like in Chapter 4, we however use the expressions "item" and "item copy" interchangeably when talking about solutions. As an example, let us consider the sentence "the solution to I uses at most one item in $\tilde{I}$." It means that the solution contains item copies of items in I, but at most one item copy whose corresponding item is in $\tilde{I}$. To be more precise, the multiset V uses only one item $a \in \tilde{I}$ with a multiplicity $x_a > 0$. We have $x_a \leq 1$, but $x_{a''} = 0$ for all other $a'' \in \tilde{I}$, i.e. $\sum_{a' \in \tilde{I}} x_{a'} \leq 1$. Similarly, "the solution V uses at most $n' \in \mathbb{N}$ items in $\tilde{I}$" means that there are only n' item copies whose corresponding item(s) are in $\tilde{I}$: we have $\sum_{a' \in \tilde{I}} x_{a'} \leq n'$.

The interchangeable use of "item" and "item copy" allows for shorter sentences. Moreover, it is based upon 0-1 KPIP where "item" and "item copy" are in fact identical.

Basic Principle of the Algorithm It has already been shown in Section 3.4 that it is sufficient to find an implementation of `MaxSolution`: we only have to determine an approximate solution $\bar{P}'_{c_l} \geq (1-\varepsilon)\overline{\mathrm{OPT}}(I, c_l)$ for every $l \in \{1, \ldots, M\}$ and return

max $\frac{1}{c_l}\bar{P}'_{c_l}$. The remaining chapter therefore demonstrates how the $\bar{P}'_{c_l}$ can be found efficiently.

5.4 A First Approximation

We present a first approximation for the $\overline{\text{OPT}}(I, c_l)$. Like in Subsection 3.5.2, let

$$S_1 := \{a_i \mid s(a_i) \leq c_1\} \text{ and } S_l := \{a_i \mid c_{l-1} < s(a_i) \leq c_l\} \text{ for } l \in \{2, \ldots, M\} \ .$$

Moreover, let $\bar{S}_l := \bigcup_{l'=1}^{l} S_{l'} = \{a \mid s(a) \leq c_l\}$, which is the set of the items that have to be considered for a solution of knapsack size c_l.

The greedy algorithm becomes easier because of the unboundedness (see Section 4.4): take the most efficient item $a^{(l)}_{\text{meff}} := \arg\max_{a \in \bar{S}_l} \frac{p(a)}{s(a)}$ (if ties occur, we can e.g. consider the item of smaller index to have a smaller efficiency). Fill the knapsack c_l with as many copies of $a^{(l)}_{\text{meff}}$ as possible, i.e. take $\lfloor \frac{c_l}{s(a^{(l)}_{\text{meff}})} \rfloor$ copies of $a^{(l)}_{\text{meff}}$. Let

$$\bar{P}_{c_l} := p(a^{(l)}_{\text{meff}}) \cdot \left\lfloor \frac{c_l}{s(a^{(l)}_{\text{meff}})} \right\rfloor$$

be this value. Moreover, let $P_{c_l} := \frac{\bar{P}_{c_l}}{c_l}$ and let $P_0 := \max_{l \in \{1,\ldots,M\}} P_{c_l}$ be the global maximum. Then the following holds:

Theorem 5.4. *We have $\bar{P}_{c_l} \geq \frac{1}{2}\overline{\text{OPT}}\,(I, c_l)$ and $P_{c_l} \geq \frac{1}{2}\text{OPT}_{c_l}(I)$. Finally, $P_0 \geq \frac{1}{2}\text{OPT}(I)$ holds.*

Proof. Suppose first that the copies of $a^{(l)}_{\text{meff}}$ can completely fill the knapsack c_l. Then $p(a^{(l)}_{\text{meff}}) \cdot \lfloor \frac{c_l}{s(a^{(l)}_{\text{meff}})} \rfloor = \overline{\text{OPT}}\,(I, c_l)$ holds. Otherwise, one additional item $a^{(l)}_{\text{meff}}$ exceeds the capacity c_l. Then we have $p(a^{(l)}_{\text{meff}}) \cdot \lfloor \frac{c_l}{s(a^{(l)}_{\text{meff}})} \rfloor + p(a^{(l)}_{\text{meff}}) \geq \overline{\text{OPT}}\,(I, c_l)$. If $p(a^{(l)}_{\text{meff}}) \leq \frac{1}{2}\overline{\text{OPT}}\,(I, c_l)$, then $p(a^{(l)}_{\text{meff}}) \cdot \lfloor \frac{c_l}{s(a^{(l)}_{\text{meff}})} \rfloor \geq \overline{\text{OPT}}\,(I, c_l) - p(a^{(l)}_{\text{meff}}) \geq \frac{1}{2}\overline{\text{OPT}}\,(I, c_l)$, and the theorem follows. Otherwise $p(a^{(l)}_{\text{meff}}) \cdot \lfloor \frac{c_l}{s(a^{(l)}_{\text{meff}})} \rfloor \geq p(a^{(l)}_{\text{meff}}) \geq \frac{1}{2}\overline{\text{OPT}}\,(I, c_l)$.

Thus, $\bar{P}_{c_l}$ is a $1 - \varepsilon = 1 - \frac{1}{2} = \frac{1}{2}$ approximation of $\overline{\text{OPT}}\,(I, c_l)$ so that $\bar{P}_{c_l}$ is a suitable lower bound on $\overline{\text{OPT}}\,(I, c_l)$ and $2\bar{P}_{c_l}$ an upper bound.

$P_{c_l} = \frac{\bar{P}_{c_l}}{c_l}$ is a $\frac{1}{2}$ solution for c_l with profits $\frac{p_j}{c_l}$ according to Lemma 3.5. Lemma 3.4 shows that $P_0 = \max_{l \in \{1,\ldots,M\}} P_{c_l}$ is also a $\frac{1}{2}$ approximation of $\text{OPT}(I)$ with $\frac{1}{2}\text{OPT}(I) \leq P_0 \leq \text{OPT}(I)$. (As above, the proof is taken from [61, p. 232, 63]) □

Corollary 5.5. *We can discard knapsacks c_l with $P_{c_l} = \frac{\bar{P}_{c_l}}{c_l} < \frac{1}{2}P_0$.*

Proof. See Corollary 3.10 on page 47. □

Interestingly, the largest knapsack size will never be discarded.

Lemma 5.6. *The largest knapsack size $c_M = 1$ will always satisfy $\bar{P}_{c_M} = P_{c_M} \geq \frac{1}{2}P_0$.*

Proof. Suppose first that $P_0 = \frac{\bar{P}_{c_l}}{c_l}$ for a knapsack $c_l < \frac{1}{2}$. Since the items that have the total profit $\bar{P}_{c_l}$ fit into c_l, we can fill c_M with these items $\lfloor \frac{1}{c_l} \rfloor$ times. Thus, $\bar{P}_{c_M} \geq \lfloor \frac{1}{c_l} \rfloor \bar{P}_{c_l}$. Note that $c_l \cdot P_0 = \bar{P}_{c_l}$. We get

$$\bar{P}_{c_M} \geq \left\lfloor \frac{1}{c_l} \right\rfloor \bar{P}_{c_l} \geq \left(\frac{1}{c_l} - 1 \right) \bar{P}_{c_l} = \frac{\bar{P}_{c_l}}{c_l} - \bar{P}_{c_l} = P_0 - c_l \cdot P_0 = (1 - c_l)\, P_0 \overset{c_l < 1/2}{\geq} \frac{1}{2} P_0 \ .$$

Suppose now that $c_l \geq \frac{1}{2}$. Then

$$\bar{P}_{c_M} \geq \bar{P}_{c_l} = c_l \cdot P_0 \geq \frac{1}{2} P_0 \ .$$

In both cases, the lemma follows. □

We present now one way to find the items and sets.

Theorem 5.7. *Algorithm 5.1 constructs the sets S_l and finds the $a^{(l)}_{\text{meff}}$, $\bar{P}_{c_l}$ and P_0 in time $\mathcal{O}(M + n \log M)$ and space $\mathcal{O}(n + M)$. Moreover, knapsack sizes c_l with $P_{c_l} = \frac{\bar{P}_{c_l}}{c_l} < \frac{1}{2}P_0$ are discarded.*

Proof. The running time and the space needed are almost obvious. The idea to construct the S_l is taken from Lawler [63]: create M stacks, one for each S_l. Each item a is then added to the right stack by binary search. Note that the item a_{meff} is the currently known most efficient item for $\bar{S}_l = \cup_{l'=1}^{l} S_{l'}$ so that $a^{(l)}_{\text{meff}}$ is correctly defined. The re-combination of sets S_l and S_{l+1} when c_l is discarded can be done in $\mathcal{O}(1)$ if the right data structure is used (e.g. linked lists). Note that Lemma 5.6 is taken into account because the knapsack size c_M is never discarded. □

Remark 5.8. Like Algorithm 3.2, Algorithm 5.1 only creates $\mathcal{O}(n)$ sets S_l. This is sufficient: it can easily be deduced that $S_l = \emptyset$ if one set $S_{l'}$ for $l' < l$ is followed by $S_{l''}$ with $l'' > l$ such that we do not have to save the sets $S_l = \emptyset$.

5.5 Partitioning the Knapsack Sizes and Reducing the Items

Remark 5.9. For ease of notation, M denotes from now on the number of the remaining knapsack sizes, i.e. the knapsacks that are not discarded because of Corollary 5.5.

Algorithm 5.1: This algorithm constructs the sets S_l and finds the $a^{(l)}_{\text{meff}}$, $\bar{P}_{c_l}$ and P_0. Knapsack sizes c_l with $P_{c_l} < \frac{1}{2}P_0$ are discarded.

for $l = 1, \ldots, M$ **do**
  $a^{(l)}_{\text{meff}} := \emptyset$;
for *all* $a \in I$ **do**
  Determine by binary search l such that $c_{l-1} < s(a) \leq c_l$;
  if S_l *not defined* **then**
    Create $S_l = \emptyset$;
  $S_l := S_l \cup \{a\}$;
$a_{\text{meff}} := \emptyset$;
for $l = 1, \ldots, M$ **do**
  for $a \in S_l$ **do**
    if $a_{\text{meff}} = \emptyset$ *or* $\frac{p(a)}{s(a)} > \frac{p(a_{\text{meff}})}{s(a_{\text{meff}})}$ **then**
      $a_{\text{meff}} := a$;
  $a^{(l)}_{\text{meff}} := a_{\text{meff}}$;
$P_0 := 0$;
for $l = 1, \ldots, M$ **do**
  $\bar{P}_{c_l} := p(a^{(l)}_{\text{meff}}) \cdot \left\lfloor \frac{c_l}{s(a^{(l)}_{\text{meff}})} \right\rfloor$;
  if $\frac{\bar{P}_{c_l}}{c_l} > P_0$ **then**
    $P_0 := \frac{\bar{P}_{c_l}}{c_l}$;
for $l = 1, \ldots, M-1$ **do**
  if $\frac{\bar{P}_{c_l}}{c_l} < \frac{1}{2}P_0$ **then**
    $C := C \setminus \{c_l\}$;
    $S_{l+1} := S_l \cup S_{l+1}$;
    Discard S_l;

5.5.1 The Knapsacks

As in Subsection 3.5.3 and as explained in Remark 3.24, we partition the knapsacks into intervals

$$C_b := \left\{ c_l \in C \;\middle|\; c_l \in \left(\frac{1}{2^{b+1}}, \frac{1}{2^b} \right] \right\} \quad \text{for } b \in \left\{ 0, \dots, \left\lfloor \log_2 \frac{1}{c_{\min}} \right\rfloor \right\} \tag{5.1}$$

if there are more than $\lfloor \log_2 \frac{1}{c_{\min}} \rfloor + 1 = \lfloor \log_2 \frac{1}{c_1} \rfloor + 1$ knapsack sizes in C, otherwise we directly set $C_l = C_b = \{c_l\}$ for $l \in \{1, \dots, M\}$. Furthermore, let $c_{\min}^{(b)}$ be the smallest and $c_{\max}^{(b)}$ be the largest knapsack size in C_b, and $l_{\min}^{(b)}$ and $l_{\max}^{(b)}$ their indices.

Theorem 5.10. *Algorithm 5.2 constructs the item sets C_b in time and space $\mathcal{O}(M)$.*

Proof. The correctness is obvious, For the running time, we use that we can compute the right b such that $c_l \in (\frac{1}{2^{b+1}}, \frac{1}{2^b}]$ in $\mathcal{O}(1)$ because the logarithm can be computed in $\mathcal{O}(1)$. □

Algorithm 5.2: The algorithm creates the knapsack intervals C_b.

```
if |C| ≥ ⌊log_2 1/c_1⌋ + 1 then
    for c_l ∈ C do
        Determine b such that c_l ∈ (1/2^{b+1}, 1/2^b];
        if C_b has not been defined then
            Create the set C_b = ∅ and set c_min^(b) := ∅ and c_max^(b) := ∅;
        C_b = C_b ∪ {c_l};
        if c_min^(b) = ∅ or c_l < c_min^(b) then c_min^(b) := c_l;
        if c_max^(b) = ∅ or c_l > c_max^(b) then c_max^(b) := c_l;
else
    for l ∈ {1, ..., M} do
        Create C_l and set C_l := {c};
```

Assumption 5.1. We assume without loss of generality that $\varepsilon \leq \frac{1}{4}$ and $\varepsilon = \frac{1}{2^{\kappa-1}}$ for $\kappa \in \mathbb{N}$ (i.e. $\kappa \geq 3$). Otherwise, we replace ε by the corresponding $\frac{1}{2^{\kappa-1}}$ for which $\frac{1}{2^{\kappa-1}} \leq \varepsilon < \frac{1}{2^{\kappa-2}}$. Note that $\log_2(\frac{2}{\varepsilon}) = \kappa$ holds.

Similar to Lawler [63] and the other knapsack algorithms in this thesis, we introduce the threshold $T_b > 0$ and a constant $K_b > 0$ for every C_b. Contrary to Chapter 4, we do not explicitly define them yet, but we will determine the right values at the end. This will also explain how the corresponding values for T and K in Chapter 4 were derived.

5.5.2 Partitioning of the Items and the Basic Idea of the Algorithm

From now on, we focus on one knapsack interval C_b. First, we partition the items into large(-profit) and small(-profit) items and only keep the most efficient small item for every $c_l \in C_b$:

$$I_L^{(b)} := \left\{ a \in I \mid p(a) \geq T_b \text{ and } s(a) \leq c_{\max}^{(b)} \right\} \tag{5.2}$$
$$\text{and} \quad a_{\text{eff}}^{(l)} := \arg\max \left\{ \frac{p(a)}{s(a)} \;\middle|\; p(a) < T_b \text{ and } s(a) \leq c_l \right\} \quad .$$

The items $a_{\text{eff}}^{(l)}$ are collected in one set

$$I_{S,\text{eff}}^{(b)} := \bigcup_{l=l_{\min}^{(b)}}^{l_{\max}^{(b)}} \left\{ a_{\text{eff}}^{(l)} \right\} \quad . \tag{5.3}$$

For convenience, we formally also introduce the set of small items

$$I_S^{(l)} := \{ a \in I \mid p(a) < T_b \text{ and } s(a) \leq c_l \} \qquad \text{for every } c_l \in C_b \ , \tag{5.4}$$

which will however not be explicitly constructed, but only used in the analysis of the algorithm.

We can now present the basic concept of our approximation algorithm. Take one $c_l \in C_b$. It is not difficult to see that

$$\begin{aligned} \overline{\text{OPT}}\,(I, c_l) &= \max_{0 \leq v \leq c_l} \overline{\text{OPT}}\,(\{a \mid p(a) \geq T_b\}, v) + \overline{\text{OPT}}\,(\{a \mid p(a) < T_b\}, c_l - v) \\ &= \max_{0 \leq v \leq c_l} \overline{\text{OPT}}\,(\{a \mid p(a) \geq T_b \text{ and } s(a) \leq c_l\}, v) + \overline{\text{OPT}}\left(I_S^{(l)}, c_l - v\right) \quad . \end{aligned}$$

The principle of many FPTAS for KP where only one knapsack size c is given (e.g. Lawler's algorithm [63] or our algorithm in Chapter 4) is to approximate both summands above. Additionally, Chapter 3 used the observation

$$\overline{\text{OPT}}\,(\{a \mid p(a) \geq T_b \text{ and } s(a) \leq c_l\}, v) = \overline{\text{OPT}}\left(I_L^{(b)}, v\right) \text{ for } 0 \leq v \leq c_l \leq c_{\max}^{(b)}$$

so that $\overline{\text{OPT}}(I_L^{(b)}, v)$ was approximated for all $0 \leq v \leq c_{\max}^{(b)}$ and these approximations used for all $c_l \in C_b$ (see Subsection 3.5.3). This led to an improved running time for the KPIP algorithms in Chapter 3. The algorithm in this chapter also uses these ideas.

Theorem 5.11. *Algorithm 5.3 finds $I_L^{(b)}$ and the items $a_{eff}^{(l)}$ in time $\mathcal{O}(n + |C_b|)$ and space $\mathcal{O}(n + |C_b|)$.*

Proof. It is clear that the set $I_L^{(b)}$ is constructed. Moreover, a_{eff} is the currently most efficient small item with $s(a_{\text{eff}}) \leq c_l$. If $l \geq l_{\min}^{(b)}$, it is saved as the most efficient small item that fits into c_l. The running time is obviously in $\mathcal{O}(|C_b| + \sum_{l=1}^{l_{\max}^{(b)}} |S_l|) \subseteq \mathcal{O}(|C_b| + n)$, where we use Remark 5.8. The space complexity is clear. □

Algorithm 5.3: Construction of $I_L^{(b)}$ and determining the $a_{\text{eff}}^{(l)}$

```
I_L^(b) := ∅;
a_eff := ∅;
for c_l ∈ C_b do Set a_eff^(l) := ∅;
for S_l ≠ ∅ and l = 1, ..., l_max^(b) do
    for a ∈ S_l do
        if p(a) ≥ T_b then
            I_L^(b) := I_L^(b) ∪ {a};
        else if a_eff = ∅ or p(a)/s(a) > p(a_eff)/s(a_eff) then
            a_eff := a;
    if a_eff ≠ ∅ and l ≥ l_min^(b) then a_eff^(l) := a_eff;
```

5.5.3 Reduction of the Items

We now reduce the set of large items $I_L^{(b)}$ to $I_{L,\text{red}}^{(b)}$ as done in Section 4.5. First, we introduce a technical definition.

Definition 5.12. *For $c_l \in C_b$, let $\kappa_l \in \mathbb{N}$ be the exponent such that $\bar{P}_{c_l} \in [2^{\kappa_l} T_b, 2^{\kappa_l+1} T_b)$. Let $\kappa_{\min}^{(b)}$ and $\kappa_{\max}^{(b)}$ be the values for $c_{\min}^{(b)}$ and $c_{\max}^{(b)}$, respectively.*

We obviously use the following assumption:

Assumption 5.2. We have $T_b \leq \bar{P}_{c_l}$ for all $c_l \in C_b$.

The values $\kappa_{\min}^{(b)}$ and $\kappa_{\max}^{(b)}$ are related.

Lemma 5.13. *The inequalities $\kappa_{\min}^{(b)} \leq \kappa_{\max}^{(b)} \leq \kappa_{\min}^{(b)} + 2$ hold.*

Proof. The first inequality is obvious. For the second one, we use Corollary 5.5 (see also Theorem 5.7), the fact that $c_{\max}^{(b)} \leq 2c_{\min}^{(b)}$ because of (5.1), and the identity $P_{c_l} = \frac{\bar{P}_{c_l}}{c_l}$:

$$2\bar{P}_{c_{\max}^{(b)}} = 2c_{\max}^{(b)} P_{c_{\max}^{(b)}} \leq 2c_{\max}^{(b)} P_0 = 4c_{\max}^{(b)} \frac{P_0}{2} \overset{\text{Cor. 5.5}}{\leq} 4c_{\max}^{(b)} P_{c_{\min}^{(b)}} = 4\frac{c_{\max}^{(b)}}{c_{\min}^{(b)}} \bar{P}_{c_{\min}^{(b)}} \overset{(5.1)}{\leq} 8\bar{P}_{c_{\min}^{(b)}} .$$

Hence, $\bar{P}_{c^{(b)}_{\max}} \leq 4\bar{P}_{c^{(b)}_{\min}}$ holds from which the second inequality follows by the definition of the values κ_l in Definition 5.12. □

Let $a \in I_L^{(b)}$ so that $s(a) \leq c^{(b)}_{\max}$. Obviously, $p(a) \leq \overline{\mathrm{OPT}}(I, c^{(b)}_{\max}) \leq 2\bar{P}_{c^{(b)}_{\max}}$ holds (see Theorem 5.4). Hence, all large items $a \in I_L^{(b)}$ have a profit in the interval $[T_b, 2\bar{P}_{c^{(b)}_{\max}}] \subseteq [T_b, 2^{\kappa^{(b)}_{\max}+2}T_b)$. We partition this interval into

$$\begin{aligned} L^{(k,b)} &:= \left[2^k T_b, 2^{k+1}T_b\right) \text{ for } k \in \left\{0, \ldots, \kappa^{(b)}_{\max}\right\} \\ \text{and} \quad L^{(\kappa^{(b)}_{\max}+1,b)} &:= \left[2^{\kappa^{(b)}_{\max}+1}T_b, 2\bar{P}_{c^{(b)}_{\max}}\right] \ . \end{aligned} \tag{5.5}$$

Note that the $L^{(k,b)}$ are similar to the intervals in Subsection 3.5.4.

We further split the $L^{(k,b)}$ into disjoint sub-intervals, each of length $2^k K_b$:

$$L_\gamma^{(k,b)} := \left[2^k T_b + \gamma \cdot 2^k K_b,\ 2^k T + (\gamma+1)2^k K_b\right) \ \text{ for } \ \gamma \in \{0, \ldots, \gamma_0\} \ , \tag{5.6}$$

where $\gamma_0 = \gamma_0(T_b, K_b)$ will be determined later. Obviously, the partitioning of $L^{(\kappa^{(b)}_{\max}+1,b)}$ is slightly different:

$$\begin{aligned} L_\gamma^{(\kappa^{(b)}_{\max}+1,b)} &:= \left[2^{\kappa^{(b)}_{\max}+1}T_b + \gamma \cdot 2^{\kappa^{(b)}_{\max}+1}K_b,\ 2^{\kappa^{(b)}_{\max}+1}T + (\gamma+1)2^{\kappa^{(b)}_{\max}+1}K_b\right) \\ &\qquad \text{for } \ \gamma \in \{0, \ldots, \gamma_1 - 1\} \\ \text{and} \quad L_{\gamma_1}^{(\kappa^{(b)}_{\max}+1,b)} &:= \left[2^{\kappa^{(b)}_{\max}+1}T_b + \gamma_1 \cdot 2^{\kappa^{(b)}_{\max}+1}K_b, 2\bar{P}_{c^{(b)}_{\max}}\right] \end{aligned}$$

for the right $\gamma_1 \leq \gamma_0$.

We make the following assumption:

Assumption 5.3. We have $(\gamma_0 + 1)K_b = T_b$ for the right choice of $\gamma_0 \in \mathbb{N}_{\geq 1}$ and therefore $\frac{K_b}{T_b} < 1$.

Hence, we have

$$\bigcup_{k=0}^{\kappa^{(b)}_{\max}} \bigcup_{\gamma=0}^{\gamma_0} L_\gamma^{(k,b)} \cup \bigcup_{\gamma=0}^{\gamma_1} L_\gamma^{(\kappa^{(b)}_{\max}+1,b)} = \bigcup_{k=0}^{\kappa^{(b)}_{\max}+1} L^{(k,b)} = [T_b, 2\bar{P}_{c^{(b)}_{\max}}] \ .$$

The idea is (again) to keep only the smallest item a for every profit interval $L_\gamma^{(k,b)}$. We will see that these items are sufficient to determine an approximate solution. As mentioned in the preceding chapters, this is the reasoning used by Lawler [63], which we also employed for our UKPIP FPTAS (see Subsection 3.6.1). The difference is that the profit of an item in $L_\gamma^{(k,b)}$ was scaled to $q(a) = 2^k \lfloor \frac{p(a)}{2^k K_b} \rfloor$ (see Subsection 3.5.3) while the original profit $p(a)$ is kept here. Note that items in one $L_\gamma^{(k,b)}$ would indeed have the same scaled profit.

Definition 5.14. *For an item a with $p(a) \geq T_b$, let $k(a) \in \mathbb{N}$ be the interval such that $p(a) \in L^{(k(a),b)}$ and $\gamma(a) \in \mathbb{N}$ be the sub-interval such that $p(a) \in L^{(k(a),b)}_{\gamma(a)}$. Let $a^{(k,b)}_\gamma$ be the smallest item for the profit interval $L^{(k,b)}_\gamma$, i.e.*

$$a^{(k,b)}_\gamma := \arg\min \left\{ s(a) \mid a \in I^{(b)}_L \text{ and } p(a) \in L^{(k,b)}_\gamma \right\} \text{ for all } k \text{ and } \gamma \ .$$

Algorithm 5.4 shows the algorithm to determine the $a^{(k,b)}_\gamma$. They form the reduced set of large items

$$I^{(b)}_{L,\text{red}} := \bigcup_k \bigcup_\gamma \left\{ a^{(k,b)}_\gamma \right\} \ . \tag{5.7}$$

Algorithm 5.4: The algorithm to determine the $a^{(k,b)}_\gamma$.

for $k = 0, \ldots, \kappa^{(b)}_{\max}$ **do**
  for $\gamma = 0, \ldots, \gamma_0$ **do**
    $a^{(k,b)}_\gamma := \emptyset$;
for $\gamma = 0, \ldots, \gamma_1$ **do**
  $a^{(\kappa^{(b)}_{\max}+1,b)}_\gamma := \emptyset$;
for $a \in I^{(b)}_L$ **do**
  Determine $(k(a), \gamma(a))$;
  if $s(a^{(k(a),b)}_{\gamma(a)}) > s(a)$ *or* $a^{(k(a),b)}_{\gamma(a)} = \emptyset$ **then**
    $a^{(k(a),b)}_{\gamma(a)} := a$;
Output: $I^{(b)}_{L,\text{red}} := \bigcup_k \bigcup_\gamma \{a^{(k,b)}_\gamma\}$

Similar to Lawler [63], we now prove that the overall solution quality does not decrease too much.

Lemma 5.15. *Let $0 \leq v \leq c^{(b)}_{\max}$. Then*

$$\overline{\text{OPT}}\left(I^{(b)}_{S,\mathit{eff}}, c_l - v\right) \geq \overline{\text{OPT}}\left(\left\{a^{(l)}_{\mathit{eff}}\right\}, c_l - v\right) \geq \overline{\text{OPT}}\left(I^{(l)}_S, c_l - v\right) - T_b$$

and

$$\overline{\text{OPT}}\left(I^{(b)}_{L,\text{red}}, v\right) \geq \left(1 - \frac{K_b}{T_b}\right) \overline{\text{OPT}}\left(I^{(b)}_L, v\right) \ .$$

Proof. We prove the first chain of inequalities. The first inequality in it is clear because $a^{(l)}_{\text{eff}} \in I^{(b)}_{S,\text{eff}} = \bigcup_{l'=l^{(b)}_{\min}}^{l^{(b)}_{\max}} \{a^{(l')}_{\text{eff}}\}$. For the second inequality, there are two possibilities:

either copies of $a_{\text{eff}}^{(l)}$ can be taken such that the entire capacity $c_l - v$ is used. Then obviously $\overline{\text{OPT}}(\{a_{\text{eff}}^{(l)}\}, c_l - v) = \overline{\text{OPT}}(I_S^{(l)}, c_l - v)$ holds. Otherwise, we have similar to the proof of Theorem 5.4 that $\overline{\text{OPT}}(\{a_{\text{eff}}^{(l)}\}, c_l - v) + p(a_{\text{eff}}^{(l)}) = \lfloor \frac{c_l - v}{s(a_{\text{eff}}^{(l)})} \rfloor \cdot p(a_{\text{eff}}^{(l)}) + p(a_{\text{eff}}^{(l)}) \geq \overline{\text{OPT}}(I_S^{(l)}, c_l - v)$. Thus, $\overline{\text{OPT}}(\{a_{\text{eff}}^{(l)}\}, c_l - v) \geq \overline{\text{OPT}}(I_S^{(l)}, c_l - v) - p(a_{\text{eff}}^{(l)}) \geq \overline{\text{OPT}}(I_S^{(l)}, c_l - v) - T_b$.

For the second inequality, take an optimal solution $(x_a)_{a \in I_L^{(b)}}$ such that $\overline{\text{OPT}}(I_L^{(b)}, v) = \sum_{a \in I_L^{(b)}} p(a) x_a$. Replace now every item a by its counterpart $a_{\gamma(a)}^{(k(a),b)}$ in $I_{L,\text{red}}^{(b)}$. Obviously, the solution stays feasible, i.e. the volume v will not be exceeded, because an item may only be replaced by a smaller one. This solution has the total profit $\sum_{a \in I_L^{(b)}} p(a_{\gamma(a)}^{(k(a),b)}) x_a$. Moreover, we have

$$\begin{aligned} p(a_{\gamma(a)}^{(k(a),b)}) \quad &\geq \quad p(a) - 2^{k(a)} K_b = p(a) \cdot \left(1 - \frac{2^{k(a)} K_b}{p(a)}\right) \\ &\overset{p(a) \geq 2^{k(a)} T_b}{\geq} p(a) \cdot \left(1 - \frac{2^{k(a)} K_b}{2^{k(a)} T_b}\right) = p(a) \cdot \left(1 - \frac{K_b}{T_b}\right) \end{aligned} \tag{5.8}$$

by the definition of the $L_\gamma^{(k,b)}$. We get

$$\begin{aligned} \overline{\text{OPT}}\left(I_{L,\text{red}}^{(b)}, v\right) &\geq \sum_{a \in I_L^{(b)}} p\left(a_{\gamma(a)}^{(k(a),b)}\right) x_a \overset{(5.8)}{\geq} \sum_{a \in I_L^{(b)}} \left(1 - \frac{K_b}{T_b}\right) \cdot p(a) x_a \\ &= \left(1 - \frac{K_b}{T_b}\right) \overline{\text{OPT}}\left(I_L^{(b)}, v\right) \quad . \end{aligned}$$

(The reasoning is again taken in parts directly from or close to the one by Lawler in [63].) Note that we have used the expressions "items" and "item copies" interchangeably in this proof as described in Remark 5.3. □

Theorem 5.16. *$I_{L,\text{red}}^{(b)}$ has $\mathcal{O}(\gamma_0 \cdot \kappa_{\max}^{(b)})$ items. Algorithm 5.4 needs time in $\mathcal{O}(n + \gamma_0 \cdot \kappa_{\max}^{(b)})$ and space in $\mathcal{O}(\gamma_0 \cdot \kappa_{\max}^{(b)})$ for the construction and for saving $I_{L,\text{red}}^{(b)}$.*

Proof. Together with the $a_\gamma^{(\kappa_{\max}^{(b)}+1,b)}$, we have $\mathcal{O}((\gamma_0 + 1) \cdot (\kappa_{\max}^{(b)} + 1) + (\gamma_1 + 1)) = \mathcal{O}(\gamma_0 \cdot \kappa_{\max}^{(b)})$ items $a_\gamma^{(k,b)}$. The space needed is asymptotically bounded by the space required to save these items $a_\gamma^{(k,b)}$. Finally, the running time is obviously bounded by $\mathcal{O}(n + \gamma_0 \cdot \kappa_{\max}^{(b)})$: the values $k(a)$ and $\gamma(a)$ can be found in $\mathcal{O}(1)$ because we assume that the logarithm can be determined in $\mathcal{O}(1)$. □

5.6 A Simplified Solution Structure

This section closely follows the corresponding Section 4.6. We will transform $I_{L,\text{red}}^{(b)}$ into a new instance $\tilde{I}^{(b)}$ whose optimum $\overline{\text{OPT}}(\tilde{I}^{(b)}, v)$ is only slightly smaller than

$\overline{\mathrm{OPT}}(I^{(b)}_{L,\mathrm{red}}, v)$ and where the corresponding solution has a special structure. This new transformation will allow us later to faster construct the approximate solution. First, we define

$$I^{(k,b)} := \left\{ a \in I^{(b)}_{L,\mathrm{red}} \mid p(a) \in L^{(k,b)} \right\} = \left\{ a \in I^{(b)}_{L,\mathrm{red}} \mid p(a) \in \left[2^k T_b, 2^{k+1} T_b \right) \right\}$$
$$\text{for } k \in \left\{ 0, \dots, \kappa^{(b)}_{\max} \right\}$$

$$\text{and } I^{(\kappa^{(b)}_{\max}+1,b)} := \left\{ a \in I^{(b)}_{L,\mathrm{red}} \mid p(a) \in L^{(\kappa^{(b)}_{\max}+1,b)} \right\}$$
$$= \left\{ a \in I^{(b)}_{L,\mathrm{red}} \mid p(a) \in \left[2^{\kappa^{(b)}_{\max}+1} T_b, 2\bar{P}_{c^{(b)}_{\max}} \right] \right\} .$$

Note that the items are already partitioned into the sets $I^{(k,b)}$ because of the way the set $I^{(b)}_{L,\mathrm{red}}$ has been constructed. We (re-)define the gluing operation (see Definition 4.9):

Definition 5.17. *Let a_1, a_2 be two knapsack items with $s(a_1) + s(a_2) \leq c^{(b)}_{\max}$. The gluing operation $\oplus$ combines them into a new item $a_1 \oplus a_2$ with $p(a_1 \oplus a_2) = p(a_1) + p(a_2)$ and $s(a_1 \oplus a_2) = s(a_1) + s(a_2)$.*

Thus, the gluing operation is only defined on pairs of items whose combined size does not exceed $c^{(b)}_{\max}$.

The basic idea for the new instance $\tilde{I}^{(b)}$ is as follows: we first set $\tilde{I}^{(0,b)} := I^{(0,b)}$. Then, we construct $a_1 \oplus a_2$ for all $a_1, a_2 \in \tilde{I}^{(0,b)}$ (including the case $a_1 = a_2$), which yields the item set $\tilde{\tilde{I}}^{(1,b)} := \{a_1 \oplus a_2 \mid a_1, a_2 \in \tilde{I}^{(0,b)}\}$. Note that $p(a_1 \oplus a_2) \in [2T_b, 4T_b) = L^{(1,b)}$. For every profit interval $L^{(1,b)}_\gamma$, we keep only the item of smallest size in $I^{(1,b)} \cup \tilde{\tilde{I}}^{(1,b)}$, which yields the item set $\tilde{I}^{(1,b)}$. This procedure is iterated for $k = 1, \dots, \kappa^{(b)}_{\max}$: the set $\tilde{I}^{(k,b)}$ contains the items with profits in $[2^k T_b, 2^{k+1} T_b) = L^{(k,b)}$. Gluing like above yields the item set $\tilde{\tilde{I}}^{(k+1,b)}$ with profits in $[2^{k+1} T_b, 2^{k+2} T_b) = L^{(k+1,b)}$. By taking again the smallest item in $\tilde{\tilde{I}}^{(k+1,b)} \cup I^{(k+1,b)}$ for every $L^{(k+1,b)}_\gamma$, the set $\tilde{I}^{(k+1,b)}$ is derived. For every k and γ, the item in $\tilde{I}^{(k,b)}$ with a profit in $L^{(k,b)}_\gamma$ is denoted by $\tilde{a}^{(k,b)}_\gamma$.

Note that we may glue items together that already consist of glued items. For backtracking, we save for every $\tilde{a}^{(k,b)}_\gamma$ which two items in $\tilde{I}^{(k-1,b)}$ have formed it or whether $\tilde{a}^{(k,b)}_\gamma$ has already been an item in $I^{(k,b)}$. Algorithm 5.5 presents one way to construct the sets $\tilde{I}^{(k,b)}$.

We finish when $\tilde{I}^{(\kappa^{(b)}_{\max}+1,b)}$ has been constructed, i.e. the gluing for $k = \kappa^{(b)}_{\max} + 1$ and therefore the construction of $\tilde{I}^{(\kappa^{(b)}_{\max}+2,b)}$ is not done. Corollary 5.21 below presents the reason for it.

Remark 5.18. One item $\tilde{a}^{(k,b)}_\gamma$ is in fact the combination of several items in $I^{(b)}_{L,\mathrm{red}}$. The profit and size of $\tilde{a}^{(k,b)}_\gamma$ is equal to the total profit and size of these items. The $\tilde{a}^{(k,b)}_\gamma$

Algorithm 5.5: The construction of the item sets $\tilde{I}^{(k,b)}$.

for $k = 0, \ldots, \kappa_{\max}^{(b)}$ **do**
  for $\gamma = 0, \ldots, \gamma_0$ **do**
    $\tilde{a}_\gamma^{(k,b)} := a_\gamma^{(k,b)}$;
    Backtrack$(\tilde{a}_\gamma^{(k,b)}) := a_\gamma^{(k,b)}$;

for $\gamma = 0, \ldots, \gamma_1$ **do** `// Items with profit in` $L^{(\kappa_{\max}^{(b)}+1,b)}$
  $\tilde{a}_\gamma^{(\kappa_{\max}^{(b)}+1,b)} := a_\gamma^{(\kappa_{\max}^{(b)}+1,b)}$;
  Backtrack$(\tilde{a}_\gamma^{(\kappa_{\max}^{(b)}+1,b)}) := a_\gamma^{(\kappa_{\max}^{(b)}+1,b)}$;

$\tilde{I}^{(0,b)} := I^{(0,b)}$;

for $k = 0, \ldots, \kappa_{\max}^{(b)}$ **do**
  for *all* $\tilde{a}_1 \in \tilde{I}^{(k,b)}$ **do**
    for *all* $\tilde{a}_2 \in \tilde{I}^{(k,b)}$ **do**
      if $s(\tilde{a}_1) + s(\tilde{a}_2) \leq c_{\max}^{(b)}$ **then**
        $\tilde{a} := \tilde{a}_1 \oplus \tilde{a}_2$;
        if $s(\tilde{a}) < s(\tilde{a}_{\gamma(\tilde{a})}^{(k+1,b)})$ *or* $\tilde{a}_{\gamma(\tilde{a})}^{(k+1,b)} = \emptyset$ **then**
          $\tilde{a}_{\gamma(\tilde{a})}^{(k+1,b)} := \tilde{a}$;
          Backtrack$(\tilde{a}_{\gamma(\tilde{a})}^{(k+1,b)}) := (\tilde{a}_1, \tilde{a}_2)$;
  $\tilde{I}^{(k+1,b)} := \bigcup_\gamma \left\{ \tilde{a}_\gamma^{(k+1,b)} \right\}$;

represent feasible item combinations because an arbitrary number of item copies can be taken in UKPIP.

In Section 4.6, items were only glued together for $k = 0, \ldots, \kappa - 1$. Indeed, the gluing of the items in $\tilde{I}^{(\kappa)}$ was not necessary as explained at the beginning of Section 4.7.

The item set

$$\tilde{I}^{(b)} := \bigcup_{k=0}^{\kappa_{\max}^{(b)}+1} \tilde{I}^{(k,b)}$$

has for every $0 \leq v \leq c_{\max}^{(b)}$ a solution near the original optimum $\overline{\mathrm{OPT}}(I_{L,\mathrm{red}}^{(b)}, v)$ as shown below in Theorem 5.20. It is additionally proved in Lemma 5.23 that at most one item of every $\tilde{I}^{(k,b)}$ for $k \in \{0, \ldots, \kappa_{\max}^{(b)} + 1\}$ is needed. First, we introduce a (rather technical) definition.

Definition 5.19. *Let I' be a set of knapsack items with $p(a) \geq T_b$ for every $a \in I'$. For a knapsack volume $v \leq c_{\max}^{(b)}$ and $k_0 \in \{0, \ldots, \kappa_{\max}^{(b)} + 1\}$, a solution is* structured for $k = k_0$ *if it fits into v and uses for every $k \in \{0, \ldots, k_0\}$ at most one item copy with a profit in $L^{(k,b)}$. We denote by $\mathrm{OPT}_{\leq k_0}(I', v)$ the corresponding optimum profit.*

For instance, the solution for

$$\overline{\mathrm{OPT}}_{\leq k_0}\left(\tilde{I}^{(0,b)} \cup \ldots \cup \tilde{I}^{(k_0,b)} \cup \tilde{I}^{(k_0+1,b)} \cup I^{(k_0+2,b)} \cup \ldots \cup I^{(\kappa_{\max}^{(b)}+1,b)}, v\right)$$

fits into the volume v, and it uses only one item from every $\tilde{I}^{(k,b)}$ for $k \in \{0, \ldots, k_0\}$. It may however use an arbitrary number of item copies in e.g. $\tilde{I}^{(k_0+1,b)}$ or $I^{(k_0+2,b)}$.

Theorem 5.20. *For $v \leq c_{\max}^{(b)}$ and $k_0 \in \{0, \ldots, \kappa_{\max}^{(b)}\}$, we have*

$$\overline{\mathrm{OPT}}_{\leq k_0}\left(\bigcup_{k=0}^{k_0+1} \tilde{I}^{(k,b)} \cup \bigcup_{k=k_0+2}^{\kappa_{\max}^{(b)}+1} I^{(k,b)}, v\right) \geq \left(1 - \frac{K_b}{T_b}\right)^{k_0+1} \overline{\mathrm{OPT}}\left(I_{L,\mathrm{red}}^{(b)}, v\right) \quad .$$

Proof. The proof idea is quite simple: we iteratively replace the items in $I^{(k_0+1,b)}$ by their counterpart in $\tilde{I}^{(k_0+1,b)}$ and also replace every pair of items in $\tilde{I}^{(k_0,b)}$ by the counterpart in $\tilde{I}^{(k_0+1,b)}$. This directly follows the way to construct the item sets $\tilde{I}^{(k,b)}$ presented in Algorithm 5.5.

Formally, the statement is proved by induction over k_0. Let $k_0 = 0$. Take an optimum solution to $\tilde{I}^{(0,b)} \cup I^{(1,b)} \cup \ldots \cup I^{(\kappa_{\max}^{(b)}+1,b)} = I^{(0,b)} \cup I^{(1,b)} \cup \ldots \cup I^{(\kappa_{\max}^{(b)}+1,b)} = I_{L,\mathrm{red}}^{(b)}$. For ease of notation, we directly write each item as often as it appears in the solution, i.e. we directly consider the item copies. We have three sub-sequences:

- Let $\bar{a}_1, \ldots, \bar{a}_\eta$ ($\eta \in \mathbb{N}$) be the item copies from $\tilde{I}^{(0,b)} = I^{(0,b)}$ in the optimal solution for $\overline{\mathrm{OPT}}(I^{(b)}_{L,\mathrm{red}}, v)$. We assume that η is odd (the case where η is even is easier and handled below.)

- Let $\bar{a}_{\eta+1}, \ldots, \bar{a}_{\eta+\xi}$ ($\xi \in \mathbb{N}$) be the item copies from $I^{(1,b)}$ in the optimal solution for $\overline{\mathrm{OPT}}(I^{(b)}_{L,\mathrm{red}}, v)$.

- Let $\bar{a}'_1, \ldots, \bar{a}'_\lambda$ ($\lambda \in \mathbb{N}$) be the remaining item copies from $I^{(2,b)} \cup \ldots \cup I^{(\kappa^{(b)}_{\max}+1,b)}$ in the optimal solution for $\overline{\mathrm{OPT}}(I^{(b)}_{L,\mathrm{red}}, v)$. This set is denoted by Λ. As defined above, the total profit of these items is written as $p(\Lambda)$.

We have

$$\overline{\mathrm{OPT}}\left(\tilde{I}^{(0,b)} \cup I^{(1,b)} \cup \ldots \cup I^{(\kappa^{(b)}_{\max}+1,b)}, v\right) = \sum_{i=1}^{\eta} p(\bar{a}_i) + \sum_{j=\eta+1}^{\eta+\xi} p(\bar{a}_j) + p(\Lambda) \ . \tag{5.9}$$

In the first step, every pair of items $\bar{a}_{2i-1}$ and $\bar{a}_{2i}$ from $\tilde{I}^{(0,b)}$ for $i \in \{1, \ldots, \lfloor \frac{\eta}{2} \rfloor\}$ is replaced by $\bar{a}_{2i-1} \oplus \bar{a}_{2i} \in \tilde{\tilde{I}}^{(1,b)}$. In the second step, every item $\bar{a}_{2i-1} \oplus \bar{a}_{2i}$ is again replaced by the corresponding item $\tilde{a}^{(1,b)}_{\gamma(\bar{a}_{2i-1} \oplus \bar{a}_{2i})} =: \tilde{a}^{(1,b)}_{\rho(i)}$ in $\tilde{I}^{(1,b)}$ (for $i \in \{1, \ldots, \lfloor \frac{\eta}{2} \rfloor\}$). Only the item $\bar{a}_\eta$ remains unchanged. Moreover, $\bar{a}_j$ from $I^{(1,b)}$ is replaced by the corresponding $\tilde{a}^{(1,b)}_{\gamma(\bar{a}_j)} =: \tilde{a}^{(1,b)}_{\rho(j)}$ for $j \in \{\eta+1, \ldots, \eta+\xi\}$. Note that this new solution is indeed feasible because the replacing items $\tilde{a}^{(1,b)}_\gamma$ are at most as large as the original ones. Moreover, the corresponding items $\tilde{a}^{(1,b)}_{\rho(i)}$ and $\tilde{a}^{(1,b)}_{\rho(j)}$ must exist by the construction of $\tilde{I}^{(1,b)}$. Thus, we have a (feasible) solution that consists of the item $\bar{a}_\eta \in \tilde{I}^{(0,b)}$, the items $\tilde{a}^{(1,b)}_{\rho(i)}$ and $\tilde{a}^{(1,b)}_{\rho(j)}$ in $\tilde{I}^{(1,b)}$, and the remaining items $\bar{a}'_1, \ldots, \bar{a}'_\lambda$ in $I^{(2,b)}, \ldots, I^{(\kappa^{(b)}_{\max}+1,b)}$: this solution respects the structure of $\overline{\mathrm{OPT}}_{\leq 0}(\cdot, v)$. (If η is even, no item in $\tilde{I}^{(0,b)}$ is used.)

Let now $\bar{a}$ be an item $\bar{a}_{2i-1} \oplus \bar{a}_{2i}$ or $\bar{a}_j$. It can be proved like for Inequality (5.8) that

$$p\left(\tilde{a}^{(1,b)}_{\gamma(\bar{a})}\right) \geq \left(1 - \frac{K_b}{T_b}\right) p(\bar{a}) \ . \tag{5.10}$$

Thus, we have

$$\begin{aligned}
&\overline{\mathrm{OPT}}_{\leq 0}\left(\tilde{I}^{(0,b)} \cup \tilde{I}^{(1,b)} \cup I^{(2,b)} \cup \ldots \cup I^{(\kappa_{\max}^{(b)}+1,b)}, v\right) \\
&\quad \geq \; p(\bar{a}_\eta) + \sum_{i=1}^{\lfloor \frac{\eta}{2} \rfloor} p(\tilde{a}_{\rho(i)}^{(1,b)}) + \sum_{j=\eta+1}^{\eta+\xi} p(\tilde{a}_{\rho(j)}^{(1,b)}) + p(\Lambda) \\
&\quad \overset{(5.10)}{\geq} \; p(\bar{a}_\eta) + \left(1 - \frac{K_b}{T_b}\right) \sum_{i=1}^{\lfloor \frac{\eta}{2} \rfloor} p(\bar{a}_{2i-1} \oplus \bar{a}_{2i}) + \left(1 - \frac{K_b}{T_b}\right) \sum_{j=\eta+1}^{\eta+\xi} p(\bar{a}_j) + p(\Lambda) \\
&\quad \geq \; \left(1 - \frac{K_b}{T_b}\right) \left(\sum_{i=1}^{\eta} p(\bar{a}_i) + \sum_{j=\eta+1}^{\eta+\xi} p(\bar{a}_j) + p(\Lambda) \right) \\
&\quad \overset{(5.9)}{=} \; \left(1 - \frac{K_b}{T_b}\right) \overline{\mathrm{OPT}}\left(\tilde{I}^{(0,b)} \cup I^{(1,b)} \cup \ldots \cup I^{(\kappa_{\max}^{(b)}+1,b)}, v\right) \\
&\quad = \; \left(1 - \frac{K_b}{T_b}\right) \overline{\mathrm{OPT}}\left(I_{L,\mathrm{red}}^{(b)}, v\right) \quad .
\end{aligned}$$

The statement for $k_0 = 1, \ldots, \kappa_{\max}^{(b)}$ now follows by induction. The proof is almost identical to the case $k_0 = 0$ above, the only difference is that there are additionally the items in $\tilde{I}^{(0,b)}, \ldots, \tilde{I}^{(k_0-1,b)}$ that remain unchanged like the items $I^{(k_0+2,b)}, \ldots, I^{(\kappa_{\max}^{(b)}+1,b)}$. Only items in $\tilde{I}^{(k_0,b)}$ and $I^{(k_0+1,b)}$ are replaced.

Note that we have again used the expressions "items" and "item copies" interchangeably in this proof as described in Remark 5.3. □

Corollary 5.21. *By setting $k_0 = \kappa_{\max}^{(b)}$, Theorem 5.20 shows that $\tilde{I}^{(b)} = \bigcup_{k=0}^{\kappa_{\max}^{(b)}+1} \tilde{I}^{(k,b)}$ is sufficient for an approximate solution and that it is not necessary to construct $\tilde{I}^{(k_0,b)}$ for $k_0 \geq \kappa_{\max}^{(b)} + 2$.*

Lemma 5.22. *For $c_l \in C_b$, we have*

$$\begin{aligned}
\overline{\mathrm{OPT}}\left(\tilde{I}^{(b)} \cup \bigcup_{l'=l_{\min}^{(b)}}^{l} \left\{a_{\mathit{eff}}^{(l')}\right\}, c_l\right) &\leq \overline{\mathrm{OPT}}\left(\tilde{I}^{(b)} \cup I_{S,\mathit{eff}}^{(b)}, c_{\max}^{(b)}\right) \\
&\leq \overline{\mathrm{OPT}}\left(I_{L,\mathrm{red}}^{(b)} \cup I_S^{(l_{\max}^{(b)})}, c_{\max}^{(b)}\right) \\
&\leq \overline{\mathrm{OPT}}\left(I_L^{(b)} \cup I_S^{(l_{\max}^{(b)})}, c_{\max}^{(b)}\right) = \overline{\mathrm{OPT}}\left(I, c_{\max}^{(b)}\right) \\
&\leq 2\bar{P}_{c_{\max}^{(b)}} \quad .
\end{aligned}$$

Proof. The first inequality is obvious because $\bigcup_{l'=l_{\min}^{(b)}}^{l} \{a_{\mathrm{eff}}^{(l')}\} \subseteq I_{S,\mathrm{eff}}^{(b)}$ (see (5.3)) and $c_l \leq c_{\max}^{(b)}$. For the second inequality, note that $\tilde{I}^{(b)}$ consists of items in $I_{L,\mathrm{red}}^{(b)}$ or of items

that can be obtained by gluing several items in $I^{(b)}_{L,\text{red}}$ together. Every combination of items in $\tilde{I}^{(b)}$ can therefore be represented by items in $I^{(b)}_{L,\text{red}}$ (see also Remark 5.18). Moreover, we have $I^{(b)}_{S,\text{eff}} \subseteq I^{(l^{(b)}_{\max})}_{S}$ (see (5.3) and (5.4)). The second inequality follows.

Since $I^{(b)}_{L,\text{red}} \subseteq I^{(b)}_{L}$, the third inequality is obvious. The identity is also trivial because $I^{(b)}_{L} \cup I^{(l^{(b)}_{\max})}_{S}$ is just the set I reduced to the items that fit into $c^{(b)}_{\max}$ (see (5.2) and (5.4)). The last inequality directly follows from Theorem 5.4. □

Up to now, we have (only) reduced the original item set I to $\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{eff}}$.

Lemma 5.23. *Let $c_l \in C_b$. Consider the optimum structured solutions to $\tilde{I}^{(b)} \cup I^{(b)}_{S,eff}$ for knapsack c_l and $k_0 = \kappa^{(b)}_{\max}$ (see Definition 5.19). This means that at most one item is used from every $\tilde{I}^{(k,b)}$ for $k \in \{0, \dots, \kappa^{(b)}_{\max}\}$. (The items $a^{(l)}_{eff}$ have profits $p(a^{(l)}_{eff}) < T_b$ such that they do not have to satisfy any structural conditions.)*

- *Then, every solution uses at most one item in $\tilde{I}^{(\kappa^{(b)}_{\max}+1,b)}$. Hence, $\overline{\text{OPT}}_{\leq \kappa^{(b)}_{\max}}(\tilde{I}^{(b)}, v') = \overline{\text{OPT}}_{\leq \kappa^{(b)}_{\max}+1}(\tilde{I}^{(b)}, v')$ holds for all values $0 \leq v' \leq c_l$.*
- *Moreover, there is a value $0 \leq v \leq c_l$ such that*

$$\begin{aligned}
&\overline{\text{OPT}}_{\leq \kappa^{(b)}_{\max}+1}\left(\tilde{I}^{(b)}, v\right) + \overline{\text{OPT}}\left(I^{(b)}_{S,eff}, c_l - v\right) \\
&\quad = \overline{\text{OPT}}_{\leq \kappa^{(b)}_{\max}}\left(\tilde{I}^{(b)}, v\right) + \overline{\text{OPT}}\left(I^{(b)}_{S,eff}, c_l - v\right) \\
&\quad \geq \overline{\text{OPT}}_{\leq \kappa^{(b)}_{\max}}\left(\tilde{I}^{(b)}, v\right) + \overline{\text{OPT}}\left(\left\{a^{(l)}_{eff}\right\}, c_l - v\right) \\
&\quad \geq \left(1 - \frac{K_b}{T_b}\right)^{\kappa^{(b)}_{\max}+2} \overline{\text{OPT}}(I, c_l) - \left(1 - \frac{K_b}{T_b}\right)^{\kappa^{(b)}_{\max}+2} T_b \ .
\end{aligned}$$

Proof. Suppose that one structured solution to $\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{eff}}$ for $k_0 = \kappa^{(b)}_{\max}$ uses more than one item in $\tilde{I}^{(\kappa^{(b)}_{\max}+1,b)}$. Since items in $\tilde{I}^{(\kappa^{(b)}_{\max}+1,b)}$ have profits in $[2^{\kappa^{(b)}_{\max}+1}T_b, 2\bar{P}_{c^{(b)}_{\max}}]$, two items $\tilde{a}_1, \tilde{a}_2 \in \tilde{I}^{(\kappa^{(b)}_{\max}+1,b)}$ have a total profit

$$p(\tilde{a}_1) + p(\tilde{a}_2) \geq 2 \cdot 2^{\kappa^{(b)}_{\max}+1} T_b = 2^{\kappa^{(b)}_{\max}+2} T_b > 2 \cdot \bar{P}_{c^{(b)}_{\max}}$$

by the definition of $\kappa^{(b)}_{\max}$ (see Definition 5.12). However, this is a contradiction to the bound $\overline{\text{OPT}}(\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{eff}}, c_l) \leq 2\bar{P}_{c^{(b)}_{\max}}$ seen in Lemma 5.22.

Thus, $\overline{\text{OPT}}_{\leq \kappa^{(b)}_{\max}}(\tilde{I}^{(b)}, v') = \overline{\text{OPT}}_{\leq \kappa^{(b)}_{\max}+1}(\tilde{I}^{(b)}, v')$ holds for all $0 \leq v' \leq c$.

Let $v \leq c_l$ now be the volume the large items $I^{(b)}_{L}$ occupy in an optimum solution to I and knapsack size c_l. Then obviously $\overline{\text{OPT}}\,(I, c_l) = \overline{\text{OPT}}(I^{(b)}_{L}, v) + \text{OPT}(I^{(l)}_{S}, c_l - v)$

holds (see the basic idea of the algorithm in Subsection 5.5.2 as well as (5.2) and (5.4)). We get the following inequalities:

$$\begin{aligned}
\overline{\mathrm{OPT}}_{\leq \kappa_{\max}^{(b)}+1}\left(\tilde{I}^{(b)}, v\right) + \overline{\mathrm{OPT}}\left(I_{S,\mathrm{eff}}^{(b)}, c_l - v\right) &= \overline{\mathrm{OPT}}_{\leq \kappa_{\max}^{(b)}}\left(\tilde{I}^{(b)}, v\right) + \overline{\mathrm{OPT}}\left(I_{S,\mathrm{eff}}^{(b)}, c_l - v\right) \\
&\overset{(5.3)}{\geq} \overline{\mathrm{OPT}}_{\leq \kappa_{\max}^{(b)}}\left(\tilde{I}^{(b)}, v\right) + \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v\right) \qquad (5.11) \\
&\overset{\text{Thm. 5.20}}{\geq} \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+1} \overline{\mathrm{OPT}}\left(I_{L,\mathrm{red}}^{(b)}, v\right) + \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v\right) \\
&\overset{\text{Lem. 5.15}}{\geq} \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}\left(I_{L}^{(b)}, v\right) + \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v\right) \\
&\overset{\text{Ass. 5.3}}{\geq} \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \left(\overline{\mathrm{OPT}}\left(I_{L}^{(b)}, v\right) + \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v\right)\right) \\
&\overset{\text{Lem. 5.15}}{\geq} \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \left(\overline{\mathrm{OPT}}\left(I_{L}^{(b)}, v\right) + \overline{\mathrm{OPT}}\left(I_{S}^{(l)}, c_l - v\right) - T_b\right) \\
&= \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}\left(I, c_l\right) - \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} T_b \ . \qquad (5.12)
\end{aligned}$$

Note that the Inequality (5.11) is the second one in the inequality chain we want to prove. □

Let v be the volume defined as in the lemma. Assume that any solution with a value $\overline{\mathrm{OPT}}_{\leq\kappa}(\tilde{I}^{(b)}, v)$ does not use any items in $\tilde{I}^{(\kappa_{\min}^{(b)}-2,b)} \cup \ldots \cup \tilde{I}^{(\kappa_{\max}^{(b)}+1,b)}$. Because of the structure, we get

$$\begin{aligned}
\overline{\mathrm{OPT}}_{\leq \kappa_{\max}^{(b)}+1}\left(\tilde{I}^{(b)}, v\right) &\leq \sum_{k=0}^{\kappa_{\min}^{(b)}-3} \max\left\{p(a) \mid a \in \tilde{I}^{(k,b)}\right\} \leq \sum_{k=0}^{\kappa_{\min}^{(b)}-3} 2 \cdot 2^k T_b \\
&< 2^{\kappa_{\min}^{(b)}-1} T_b = \frac{1}{2} 2^{\kappa_{\min}^{(b)}} T_b \leq \frac{1}{2} \bar{P}_{c_{\min}^{(b)}} \leq \frac{1}{2} \bar{P}_{c_l} \qquad (5.13)
\end{aligned}$$

by the value of $\kappa_{\min}^{(b)}$ (see Definition 5.12). Note that Assumption 5.2 only guarantees that $\kappa_{\min}^{(b)} \geq 0$. Hence, $\tilde{I}^{(\kappa_{\min}^{(b)}-2,b)}$ and $\tilde{I}^{(\kappa_{\min}^{(b)}-1,b)}$ may indeed not exist a priori. Still, this does not change the correctness of the bound above.

Let us now assume the following lower bound:

Assumption 5.4. The constants T_b and K_b are chosen in such a way that we have for all $c_l \in C_b$ the lower bound

$$\left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}(I, c_l) - \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} T_b \geq \frac{3}{4} \bar{P}_{c_l} \ .$$

We get the following lemma:

Lemma 5.24. *Let $c_l \in C_b$, and let v be defined as in Lemma 5.23. If Assumption 5.4 holds and a solution for $\overline{\mathrm{OPT}}_{\leq \kappa^{(b)}_{\max}+1}(\tilde{I}^{(b)}, v) + \overline{\mathrm{OPT}}(I^{(b)}_{S,\mathit{eff}}, c_l - v)$ does not use any items in $\tilde{I}^{(\kappa^{(b)}_{\min}-2,b)} \cup \ldots \cup \tilde{I}^{(\kappa^{(b)}_{\max}+1,b)}$, then*

$$\overline{\mathrm{OPT}}\left(I^{(l)}_S, c_l - v\right) \geq \overline{\mathrm{OPT}}\left(\left\{a^{(l)}_{\mathit{eff}}\right\}, c_l - v\right) \geq \frac{1}{4}\bar{P}_{c_l} \ .$$

Proof. Assumption 5.4 gives a lower bound on the expressions in Lemma 5.23. On the other hand, we consider the case where the upper bound (5.13) holds. Thus, the lemma follows. □

Definition 5.25. *For every $c_l \in C_b$, take $\left\lceil \frac{\bar{P}_{c_l}/4}{p(a^{(l)}_{\mathit{eff}})} \right\rceil$ items $a^{(l)}_{\mathit{eff}}$. If their total size is at most c_l, they are glued together to $a^{(l)}_{\mathrm{eff-c}}$.*

We now strengthen Assumption 5.2.

Assumption 5.5. We have $T_b \leq \frac{1}{4}\bar{P}_{c_l}$ for all $c_l \in C_b$.

Lemma 5.26. *We have $\frac{1}{4}\bar{P}_{c^{(b)}_{\min}} \geq 2^{\kappa^{(b)}_{\min}-2}T_b$ and $\kappa^{(b)}_{\min} \geq 2$. The sets $\tilde{I}^{(\kappa^{(b)}_{\min}-2,b)}, \ldots, \tilde{I}^{(\kappa^{(b)}_{\max}+1,b)}$ are therefore well-defined and contain large items. The items $a^{(l)}_{\mathrm{eff-c}}$ have profits $p(a^{(l)}_{\mathrm{eff-c}}) \geq \frac{1}{4}\bar{P}_{c_l} \geq 2^{\kappa^{(b)}_{\min}-2}T_b \geq T_b$ and are therefore also large.*

Proof. We assume $\frac{1}{4}\bar{P}_{c^{(b)}_{\min}} \geq T_b$. By Definition 5.12, we have $2^{\kappa^{(b)}_{\min}+1}T_b > \bar{P}_{c^{(b)}_{\min}} \geq 2^{\kappa^{(b)}_{\min}}T_b$, from which the statements follow. □

There may of course be several items $a^{(l)}_{\mathrm{eff-c}}$ with profits in the same sub-interval $L^{(k,b)}_\gamma$. It is not necessary to keep all of them.

Definition 5.27. *Similar to the definition of $I^{(b)}_{L,\mathrm{red}}$ in (5.7), we keep for all k and γ only the item $a^{(l)}_{\mathrm{eff-c}}$ with a profit in $L^{(k,b)}_\gamma$ that has the smallest size. Hence, we set*

$$a^{(k,b)}_{\gamma,\mathrm{e-c}} := \arg\min\left\{s(a) \mid a = a^{(l)}_{\mathrm{eff-c}} \text{ for } c_l \in C_b \text{ and } p(a) \in L^{(k,b)}_\gamma\right\}$$

for every k and γ, and we define

$$I^{(b)}_{S,\mathrm{red-c}} := \bigcup_k \bigcup_\gamma \left\{a^{(k,b)}_{\gamma,\mathrm{e-c}}\right\} \ . \tag{5.14}$$

Having defined these auxiliary items $a^{(k,b)}_{\gamma,\mathrm{e-c}}$, we can now further refine the structure introduced in Definition 5.19.

Definition 5.28. *Take a knapsack volume $v \leq c^{(b)}_{\max}$. Consider the following solutions to $\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{red}-\text{c}}$ of size at most v:*

- *They are structured for $k = \kappa^{(b)}_{\max} + 1$, i.e. they use for every $k \in \{0, \dots, \kappa^{(b)}_{\max} + 1\}$ at most one item in $\tilde{I}^{(k,b)}$.*
- *They additionally use at most one item in $I^{(b)}_{S,\text{red}-\text{c}}$ and at least one item $\tilde{a} \in \tilde{I}^{(\kappa^{(b)}_{\min}-2,b)} \cup \dots \cup \tilde{I}^{(\kappa^{(b)}_{\max}+1,b)} \cup I^{(b)}_{S,\text{red}-\text{c}}$.*

Hence, these solutions have a profit of at least $p(\tilde{a}) \geq 2^{\kappa^{(b)}_{\min}-2} T_b$. These special solutions are called structured solutions with a lower bound *(on the profit).*

The value $\text{OPT}_{\text{St}}(\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{red}-\text{c}}, v)$ denotes the optimal profit for such solutions of total size at most v. We set $\text{OPT}_{\text{St}}(\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{red}-\text{c}}, v) = 0$ if v is too small so that such a solution does not exist.

Theorem 5.29. *For every $c_l \in C_b$, there is a value $0 \leq v \leq c_l$ such that*

$$\begin{aligned}&\overline{\text{OPT}}_{\text{St}}\left(\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{red}-\text{c}}, v\right) + \overline{\text{OPT}}\left(\left\{a^{(l)}_{\mathit{eff}}\right\}, c_l - v\right)\\ &\qquad \geq \left(1 - \frac{K_b}{T_b}\right)^{\kappa^{(b)}_{\max}+2} \overline{\text{OPT}}\left(I, c_l\right) - \left(1 - \frac{K_b}{T_b}\right)^{\kappa^{(b)}_{\max}+2} T_b \ .\end{aligned}$$

Proof. Like in the proof of Lemma 5.23, let v' be the volume the large items $I^{(b)}_L$ occupy in an optimum solution to I, i.e. $\overline{\text{OPT}}(I^{(b)}_L, v') + \overline{\text{OPT}}(I^{(l)}_S, c_l - v') = \overline{\text{OPT}}\left(I, c_l\right)$.

Consider an optimum structured solution for $\overline{\text{OPT}}_{\leq \kappa^{(b)}_{\max}+1}(\tilde{I}^{(b)}, v')$ and suppose that it does not use any items in $\tilde{I}^{(\kappa^{(b)}_{\min}-2,b)} \cup \dots \cup \tilde{I}^{(\kappa^{(b)}_{\max}+1,b)}$. By Lemma 5.24, we have $\frac{1}{4}\bar{P}_{c_l} \leq \overline{\text{OPT}}(\{a^{(l)}_{\text{eff}}\}, c_l - v') \leq \overline{\text{OPT}}(I^{(l)}_S, c_l - v')$. If we fill $c_l - v'$ with item copies of $a^{(l)}_{\text{eff}}$, we can therefore replace a part of them with one item $a^{(l)}_{\text{eff}-\text{c}}$ and this item again by the corresponding $a^{(k,b)}_{\gamma,\text{e}-\text{c}}$ (see Definition 5.27). (For ease of notation, we write $k(a^{(l)}_{\text{eff}-\text{c}}) = k$ and $\gamma(a^{(l)}_{\text{eff}-\text{c}}) = \gamma$.) Note that $c_l - v' \geq s(a^{(l)}_{\text{eff}-\text{c}}) \geq s(a^{(k,b)}_{\gamma,\text{e}-\text{c}})$. This implies together with (5.14) that we have for any volume $0 \leq v'' \leq c_l - s(a^{(l)}_{\text{eff}-\text{c}})$

$$\overline{\text{OPT}}_{\text{St}}\left(\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{red}-\text{c}}, v'' + s\left(a^{(l)}_{\text{eff}-\text{c}}\right)\right) \geq \overline{\text{OPT}}_{\leq \kappa^{(b)}_{\max}+1}\left(\tilde{I}^{(b)}, v''\right) + p\left(a^{(k,b)}_{\gamma,\text{e}-\text{c}}\right) \ . \quad (5.15)$$

By definition, we have $p(a^{(k,b)}_{\gamma,\text{e}-\text{c}}) \geq p(a^{(l)}_{\text{eff}-\text{c}}) - 2^k K_b$. Set $v := v' + s(a^{(l)}_{\text{eff}-\text{c}})$. Hence, we get

$$\begin{aligned}&\overline{\text{OPT}}_{\text{St}}\left(\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{red}-\text{c}}, v\right) + \overline{\text{OPT}}\left(\left\{a^{(l)}_{\text{eff}}\right\}, c_l - v\right)\\ &\quad = \quad \overline{\text{OPT}}_{\text{St}}\left(\tilde{I}^{(b)} \cup I^{(b)}_{S,\text{red}-\text{c}}, v' + s\left(a^{(l)}_{\text{eff}-\text{c}}\right)\right)\end{aligned}$$

$$
\begin{aligned}
&\quad + \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v' - s\left(a_{\mathrm{eff-c}}^{(l)}\right)\right) \\
&\overset{(5.15)}{\geq} \quad \overline{\mathrm{OPT}}_{\leq \kappa_{\max}^{(b)}+1}\left(\tilde{I}^{(b)}, v'\right) + p\left(a_{\gamma,\mathrm{e-c}}^{(k,b)}\right) + \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v' - s\left(a_{\mathrm{eff-c}}^{(l)}\right)\right) \\
&\overset{\text{Lem. 5.23}}{=} \quad \overline{\mathrm{OPT}}_{\leq \kappa_{\max}^{(b)}}\left(\tilde{I}^{(b)}, v'\right) + p\left(a_{\gamma,\mathrm{e-c}}^{(k,b)}\right) + \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v' - s\left(a_{\mathrm{eff-c}}^{(l)}\right)\right) \\
&\overset{\text{Thm. 5.20}}{\geq} \quad \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+1} \overline{\mathrm{OPT}}\left(I_{L,\mathrm{red}}^{(b)}, v'\right) + p\left(a_{\mathrm{eff-c}}^{(l)}\right) - 2^k K_b \\
&\quad + \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v' - s\left(a_{\mathrm{eff-c}}^{(l)}\right)\right) \\
&\overset{\text{Lem. 5.15}}{\geq} \quad \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}\left(I_L^{(b)}, v'\right) + p\left(a_{\mathrm{eff-c}}^{(l)}\right) - 2^k K_b \\
&\quad + \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v' - s\left(a_{\mathrm{eff-c}}^{(l)}\right)\right) \\
&\overset{p(a_{\mathrm{eff-c}}^{(l)}) \geq 2^k T_b}{\geq} \quad \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}\left(I_L^{(b)}, v'\right) + p\left(a_{\mathrm{eff-c}}^{(l)}\right)\left(1 - \frac{2^k K_b}{2^k T_b}\right) \\
&\quad + \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v' - s\left(a_{\mathrm{eff-c}}^{(l)}\right)\right) \\
&\overset{\text{Ass. 5.3}}{\geq} \quad \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}\left(I_L^{(b)}, v'\right) + \left(1 - \frac{K_b}{T_b}\right) p\left(a_{\mathrm{eff-c}}^{(l)}\right) \\
&\quad + \left(1 - \frac{K_b}{T_b}\right) \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v' - s\left(a_{\mathrm{eff-c}}^{(l)}\right)\right) \\
&= \quad \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}\left(I_L^{(b)}, v'\right) + \left(1 - \frac{K_b}{T_b}\right) \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v'\right) \\
&\overset{\text{Lem. 5.15}}{\geq} \quad \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}\left(I_L^{(b)}, v'\right) + \left(1 - \frac{K_b}{T_b}\right)\left(\overline{\mathrm{OPT}}\left(I_S^{(l)}, c_l - v'\right) - T_b\right) \\
&\geq \quad \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \left(\overline{\mathrm{OPT}}\left(I_L^{(b)}, v'\right) + \overline{\mathrm{OPT}}\left(I_S^{(l)}, c_l - v'\right) - T_b\right) \\
&= \quad \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}\left(I, c_l\right) - \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} T_b \ .
\end{aligned}
$$

Note that $\overline{\mathrm{OPT}}(\{a_{\mathrm{eff}}^{(l)}\}, c_l - v)$ is well-defined—and therefore the entire chain of inequalities feasible—because $c_l - v = c_l - v' - s(a_{\mathrm{eff-c}}^{(l)}) \geq 0$.

The case where one solution $\overline{\mathrm{OPT}}_{\leq \kappa_{\max}^{(b)}+1}(\tilde{I}^{(b)}, v')$ uses at least one item in $\tilde{I}^{(\kappa_{\min}^{(b)}-2,b)} \cup \ldots \cup \tilde{I}^{(\kappa_{\max}^{(b)}+1,b)}$ is proven in a similar way, we directly set $v := v'$. (The entire proof is similar to the proof of Lemma 5.23.) □

So far, we have not constructed an actual solution. Instead, we have shown in Theorem 5.29 that there is for every $c_l \in C_b$ a solution to $\tilde{I}^{(b)} \cup I_{S,\mathrm{red-c}}^{(b)} \cup \{a_{\mathrm{eff-c}}^{(l)}\}$ that is close to $\overline{\mathrm{OPT}}_{c_l}(I)$ and that is a structured solution with a lower bound.

Theorem 5.30. *The cardinality of $\tilde{I}^{(k,b)}$ is in $\mathcal{O}(\gamma_0)$, i.e. $\tilde{I}^{(b)}$ has $\mathcal{O}(\gamma_0 \cdot \kappa_{\max}^{(b)})$ items. The cardinality of $I_{S,\text{red}-\text{c}}^{(b)}$ is in $\mathcal{O}(\min\{|C_b|, \gamma_0\}) \subseteq \mathcal{O}(\gamma_0)$.*

Algorithm 5.5 constructs $\tilde{I}^{(b)}$ in time $\mathcal{O}(\gamma_0^2 \cdot \kappa_{\max}^{(b)})$ and space $\mathcal{O}(\gamma_0 \cdot \kappa_{\max}^{(b)})$, which also includes the space to store $\tilde{I}^{(b)}$ and the backtracking information. One item $a_{\text{eff}-\text{c}}^{(l)}$ can be constructed in time $\mathcal{O}(1)$. By combining the construction of all $a_{\text{eff}-\text{c}}^{(l)}$ with an algorithm similar to Algorithm 5.4, the set $I_{S,\text{red}-\text{c}}^{(b)}$ can be found in time and space $\mathcal{O}(|C_b| + \gamma_0)$.

Proof. The number of items in $\tilde{I}^{(k,b)}$ and $\tilde{I}^{(b)}$ can be derived like the number of items in $I_{L,\text{red}}^{(b)}$ in Theorem 5.16. The running time of Algorithm 5.5 is obviously dominated by the second for-loop. It is in $\mathcal{O}(\kappa_{\max}^{(b)} \cdot \gamma_0^2)$. The storage complexity is dominated by the space to save the $\tilde{a}_\gamma^{(k,b)}$ and the backtracking information, which is again asymptotically equal to the number of items in $\tilde{I}^{(b)}$.

Note that the items $a_{\gamma,\text{e}-\text{c}}^{(k,b)}$ have profits of at least $2^{\kappa_{\min}^{(b)}-2} T_b$ by Lemma 5.26 and Definition 5.27, i.e. they have profits in $[2^{\kappa_{\min}^{(b)}-2} T_b, 2\bar{P}_{c_{\max}^{(b)}}]$. This interval is covered by the profit intervals $L^{(k,b)}$ for $\kappa_{\min}^{(b)} - 2 \leq k \leq \kappa_{\max}^{(b)} + 1$, and each $L^{(k,b)}$ has $\mathcal{O}(\gamma_0)$ sub-intervals $L_\gamma^{(k,b)}$. Since each $L_\gamma^{(k,b)}$ yields at most one item $a_{\gamma,\text{e}-\text{c}}^{(k,b)}$, their number is in $\mathcal{O}((\kappa_{\max}^{(b)} + 1 - (\kappa_{\min}^{(b)} - 2) + 1) \cdot \gamma_0) = \mathcal{O}((\kappa_{\max}^{(b)} - \kappa_{\min}^{(b)}) \cdot \gamma_0) = \mathcal{O}(\gamma_0)$ by Lemma 5.13. The cardinality of $I_{S,\text{red}-\text{c}}^{(b)}$ can be bounded by $\mathcal{O}(\min\{|C_b|, \gamma_0\})$ if only the items $a_{\gamma,\text{e}-\text{c}}^{(k,b)} \neq \emptyset$ are saved.

The time to construct one $a_{\text{eff}-\text{c}}^{(l)}$ is obviously in $\mathcal{O}(1)$. The construction can easily be combined with an algorithm similar to Algorithm 5.4 to save only the $a_{\gamma,\text{e}-\text{c}}^{(k,b)}$. As we only have to consider profit sub-intervals $L_\gamma^{(k,b)}$ for $\kappa_{\min}^{(b)} - 2 \leq k$, the running time and space bound are easy to see. □

5.7 Finding an Approximate Structured Solution by Dynamic Programming

Fix one $c_l \in C_b$. In the last section, we have seen that there is an approximate solution to $\tilde{I}^{(b)} \cup I_{S,\text{red}-\text{c}}^{(b)} \cup \{a_{\text{eff}}^{(l)}\}$ where the large items are a structured solution with a lower bound (see Theorem 5.29). Regarding the large items, at most one item from every $\tilde{I}^{(k,b)}$ for $k \in \{0, \ldots, \kappa_{\max}^{(b)} + 1\}$ is therefore used together with at most one item in $I_{S,\text{red}-\text{c}}^{(b)}$. Furthermore, at least one item $\tilde{a} \in \tilde{I}^{(\kappa_{\min}^{(b)}-2,b)} \cup \ldots \cup \tilde{I}^{(\kappa_{\max}^{(b)}+1,b)} \cup I_{S,\text{red}-\text{c}}^{(b)}$ is part of the solution. (See Definition 5.28.) Note that the set of large items used in the solution is for every c_l a subset of the same set $\tilde{I}^{(b)} \cup I_{S,\text{red}-\text{c}}^{(b)}$. The idea is to find for all $c_l \in C_b$ at once the necessary values $\overline{\text{OPT}}_{\text{St}}(\tilde{I}^{(b)} \cup I_{S,\text{red}-\text{c}}^{(b)}, v)$.

We do this by dynamic programming. For convenience, let $\tilde{I}^{(\kappa_{\max}^{(b)}+2,b)} := I_{S,\text{red}-\text{c}}^{(b)}$. Similar to Lawler [63], we introduce tuples (p,s,k). For profit p with $0 \leq p \leq 2\bar{P}_{c_{\max}^{(b)}}$ and size $0 \leq s \leq c_{\max}^{(b)}$, the tuple (p,s,k) states that there is an item set of size s whose total profit is p. Moreover, the set has only items in $\tilde{I}^{(k,b)} \cup \cdots \cup \tilde{I}^{(\kappa_{\max}^{(b)}+2,b)}$ and respects the structure above.

The dynamic program is almost identical to the one in Section 4.7: start with the dummy tuple set $F^{(\kappa_{\max}^{(b)}+3)} := \{(0,0,\kappa_{\max}^{(b)}+3)\}$. For $k = \kappa_{\max}^{(b)}+2,\ldots,\kappa_{\min}^{(b)}-2$, the tuples in $F^{(k)}$ are recursively constructed by

$$\begin{aligned} F^{(k)} := &\left\{(p,s,k) \mid (p,s,k+1) \in F^{(k+1)}\right\} \\ &\cup \left\{(p+p(\tilde{a}),s+s(\tilde{a}),k) \mid (p,s,k+1) \in F^{(k+1)}, \tilde{a} \in \tilde{I}^{(k,b)}, s+s(\tilde{a}) \leq c_{\max}^{(b)}\right\} \ . \end{aligned}$$

Note that $(0,0,k+1) \in F^{(k+1)}$, which guarantees that $F^{(k)}$ also contains the tuples $(p(\tilde{a}),s(\tilde{a}),k)$ for $\tilde{a} \in \tilde{I}^{(k,b)}$ if $k \in \{\kappa_{\max}^{(b)}+2,\ldots,\kappa_{\min}^{(b)}-2\}$. For $k = \kappa_{\min}^{(b)}-3,\ldots,0$, this tuple $(0,0,k+1)$ is no longer considered to form the new tuples, which guarantees that tuples of the form $(p+p(\tilde{a}),s+s(\tilde{a}),k)$ for $\tilde{a} \in \tilde{I}^{(k,b)}$ have $p,s \neq 0$. The recursion becomes

$$\begin{aligned} F^{(k)} := &\left\{(p,s,k) \mid (p,s,k+1) \in F^{(k+1)}\right\} \\ &\cup \left\{(p+p(\tilde{a}),s+s(\tilde{a}),k) \mid (p,s,k+1) \in F^{(k+1)} \setminus \{(0,0,k+1)\}, \right. \\ &\qquad \left. \tilde{a} \in \tilde{I}^{(k,b)}, s+s(\tilde{a}) \leq c_{\max}^{(b)}\right\} \ . \end{aligned}$$

The actual item set corresponding to (p,s,k) can be reconstructed by saving backtracking information.

Definition 5.31. *A tuple (p_2,s_2,k) is dominated by (p_1,s_1,k) if $p_2 \leq p_1$ and $s_2 \geq s_1$.*

As in Section 4.7 and in [63], dominated tuples $(p,s,k+1)$ are now removed from $F^{(k+1)}$ before $F^{(k)}$ is constructed. This does not affect the outcome: dominated tuples only stand for sets of items with a profit not larger and a size not smaller than non-dominated tuples. A non-dominated tuple (p,s,k) is therefore optimal, i.e. profit p can only be obtained with items of size at least s if items in $\tilde{I}^{(k,b)},\ldots,\tilde{I}^{(\kappa_{\max}^{(b)}+2,b)}$ are considered.

Lemma 5.32. *A tuple $(p,s,k) \in F^{(k)}$ stands for a structured solution with a lower bound (see Definition 5.28). Therefore, we have $p \geq 2^{\kappa_{\min}^{(b)}-2}T_b$ if $p > 0$. For every $v \leq c_{\max}^{(b)}$, there is a tuple $(p,s,0) \in F^{(0)}$ with $p = \overline{\text{OPT}}_{\text{St}}(\tilde{I}^{(b)} \cup I_{S,\text{red}-\text{c}}^{(b)}, v)$ and $s \leq v$.*

Proof. This lemma directly follows from the dynamic program: tuples use at most one item from every $\tilde{I}^{(k,b)}$. For $k \in \{\kappa_{\min}^{(b)} - 2, \ldots, \kappa_{\max}^{(b)} + 2\}$, a tuple with $p > 0$ represents an item set that uses at least one item in $\tilde{I}^{(k,b)}, \ldots, \tilde{I}^{(\kappa_{\max}^{(b)}+2,b)}$. Tuples for $k \leq \kappa_{\min}^{(b)} - 3$ with $p > 0$ are only derived from tuples that use at least one item in $\tilde{I}^{(\kappa_{\min}^{(b)}-2,b)}, \ldots, \tilde{I}^{(\kappa_{\max}^{(b)}+2,b)}$. If dominated sets are not removed, the dynamic program obviously constructs tuples for all possible structured solutions with a lower bound, especially the optimum combinations for every $0 \leq v \leq c_{\max}^{(b)}$. Removing dominated tuples does not affect the tuples that stand for the optimum item combinations so that the second property still holds. □

While the dynamic program above constructs the desired tuples, their number may increase dramatically until $F^{(0)}$ is obtained. We therefore use approximate dynamic programming for the tuples with profits in $[2^{\kappa_{\min}^{(b)}-2}T_b, 2\bar{P}_{c_{\max}^{(b)}}]$. Such a method is also used in Chapter 4 and was inspired by the dynamic programming in [58] (see also [61, pp. 97–112]). We change it slightly compared to Chapter 4.

Definition 5.28 and Lemma 5.32 state that a tuple (p, s, k) with $p > 0$ satisfies $p \geq 2^{\kappa_{\min}^{(b)}-2}T_b$. Apart from $(0, 0, k)$, all tuples have therefore profits in the interval $[2^{\kappa_{\min}^{(b)}-2}T, 2\bar{P}_{c_{\max}^{(b)}}]$. We have already partitioned it into sub-intervals $L_\gamma^{(\bar{k},b)}$ for $\bar{k} \geq \kappa_{\min}^{(b)} - 2$ (see (5.6)). The approximate dynamic program keeps for every $\bar{k}$ and γ only the tuple (p, s, k) with $p \in L_\gamma^{(\bar{k},b)}$ that has the smallest size s. The dominated tuples are removed when all tuples for k have been constructed. The modified dynamic program is presented in Algorithm 5.6. The sets of non-dominated tuples are denoted by $D^{(k)}$.

For convenience, $(p(\bar{k}, \gamma), s(\bar{k}, \gamma), k) \in D^{(k)}$ denotes the smallest tuple with a profit in $L_\gamma^{(\bar{k},b)}$. We again save the backtracking information during the execution of the algorithm.

Lemma 5.33. *Let $\tilde{D}^{(k)}$ be the set $D^{(k)}$ from Algorithm 5.6 before the dominated tuples are removed. A tuple $(p, s, k) \in \tilde{D}^{(k)}$ for $k = \kappa_{\max}^{(b)} + 2, \ldots, 0$ stands for a structured solution with a lower bound. Therefore, we have $p \geq 2^{\kappa_{\min}^{(b)}-2}T_b$ if $p > 0$. This is also true for $(p, s, k) \in D^{(k)}$.*

Proof. The proof is almost identical to the one of Lemma 5.32. In fact, the proof is not influenced by keeping only the tuples of smallest size in every profit interval $L_\gamma^{(\bar{k},b)}$.□

Theorem 5.34. *Let $k \in \{0, \ldots, \kappa_{\max}^{(b)} + 2\}$. For every (non-dominated) tuple $(\bar{p}, \bar{s}, k) \in F^{(k)}$, there is a tuple $(p, s, k) \in D^{(k)}$ such that*

$$p \geq \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2-k} \bar{p} \quad \text{and} \quad s \leq \bar{s} \ .$$

Algorithm 5.6: The approximate dynamic programming

$D^{(\kappa_{\max}^{(b)}+3)} := \left\{(0,0,\kappa_{\max}^{(b)}+3)\right\}$;
Backtrack$(0,0,\kappa_{\max}^{(b)}+3) := \emptyset$;
for $k = \kappa_{\max}^{(b)}+2,\ldots,0$ **do**
  $D^{(k)} := \emptyset$;
  for $(p(\bar{k},\gamma),s(\bar{k},\gamma),k+1) \in D^{(k+1)}$ **do**
    $D^{(k)} := D^{(k)} \cup \{(p(\bar{k},\gamma),s(\bar{k},\gamma),k)\}$;
    Backtrack$(p(\bar{k},\gamma),s(\bar{k},\gamma),k)$:= Backtrack$(p(\bar{k},\gamma),s(\bar{k},\gamma),k+1)$;
  for $\tilde{a} \in \tilde{I}^{(k,b)}$ **do**
    for $(p,s,k+1) \in D^{(k+1)} \setminus \{(0,0,k+1)\}$ **do**
      `// Construction of new tuples`
      $(p',s',k) := (p+p(\tilde{a}),s+s(\tilde{a}),k)$;
      Determine $(\bar{k},\gamma)$ for (p',s',k) such that $p' \in L_{\gamma}^{(\bar{k},b)}$;
      if $s' < s(\bar{k},\gamma)$ *or* $(p(\bar{k},\gamma),s(\bar{k},\gamma),k) = \emptyset$ **then**
        `// Only new tuples of smaller size are kept`
        $D^{(k)} := D^{(k)} \setminus \{(p(\bar{k},\gamma),s(\bar{k},\gamma),k)\}$;
        $(p(\bar{k},\gamma),s(\bar{k},\gamma),k) := (p',s',k)$;
        Backtrack$(p(\bar{k},\gamma),s(\bar{k},\gamma),k) := ((p,s,k+1),\tilde{a})$;
        $D^{(k)} := D^{(k)} \cup \{(p(\bar{k},\gamma),s(\bar{k},\gamma),k)\}$;
    if $k \geq \kappa_{\min}^{(b)} - 2$ **then**
      `// Construction of (possible) tuples` $(p(\tilde{a}),s(\tilde{a}),k)$ `for` $k \geq \kappa_{\min}^{(b)} - 2$
      Determine $(\bar{k},\gamma)$ for $p(\tilde{a})$ such that $p(\tilde{a}) \in L_{\gamma}^{(\bar{k},b)}$;
      if $s(\tilde{a}) < s(\bar{k},\gamma)$ *or* $(p(\bar{k},\gamma),s(\bar{k},\gamma),k) = \emptyset$ **then**
        $D^{(k)} := D^{(k)} \setminus \{(p(\bar{k},\gamma),s(\bar{k},\gamma),k)\}$;
        $(p(\bar{k},\gamma),s(\bar{k},\gamma),k) := (p(\tilde{a}),s(\tilde{a}),k)$;
        Backtrack$(p(\bar{k},\gamma),s(\bar{k},\gamma),k) := (\tilde{a})$;
        $D^{(k)} := D^{(k)} \cup \{(p(\bar{k},\gamma),s(\bar{k},\gamma),k)\}$;
  Remove dominated tuples from $D^{(k)}$;

Proof. This statement is trivial for $(\bar{p}, \bar{s}, k) = (0, 0, k)$ because also $(0, 0, k) \in D^{(k)}$ (this tuple is never removed in the construction of $F^{(k)}$ and $D^{(k)}$).

Suppose now that $(\bar{p}, \bar{s}, k) \neq (0, 0, k)$. The theorem is proved by induction for $k = \kappa_{\max}^{(b)} + 2, \ldots, 0$.

The statement is evident for $k = \kappa_{\max}^{(b)} + 2$. In fact, $\tilde{I}^{(\kappa_{\max}^{(b)}+2)} = I_{S,\text{red}-\text{c}}^{(b)}$ because $I_{S,\text{red}-\text{c}}^{(b)}$ has at most one item for every $L_\gamma^{(\bar{k},b)}$ by Definition 5.27. Hence, $F^{(\kappa_{\max}^{(b)}+2)} = D^{(\kappa_{\max}^{(b)}+2)}$ holds.

Suppose that the statement is true for $k+1, \ldots, \kappa_{\max}^{(b)} + 2$. As defined in Lemma 5.33, $\tilde{D}^{(k)}$ is the set $D^{(k)}$ before the dominated tuples have been removed. Let $(\bar{p}, \bar{s}, k) \in F^{(k)}$.

There are two cases. In the first case, we have $(\bar{p}, \bar{s}, k+1) \in F^{(k+1)}$. There has to be a tuple $(p_1, s_1, k+1) \in D^{(k+1)}$ by the induction hypothesis such that $p_1 \geq \bar{p} \cdot (1 - \frac{K_b}{T_b})^{\kappa_{\max}^{(b)}+2-(k+1)}$ and $s_1 \leq \bar{s}$. Note that this implies $(p_1, s_1, k+1) \neq (0, 0, k+1)$ and therefore $p_1 \geq 2^{\kappa_{\min}^{(b)}-2} T_b$ by Lemma 5.33. Let $(\bar{k}, \gamma)$ be the index pair such that $p_1 \in L_\gamma^{(\bar{k},b)}$. During the execution of Algorithm 5.6, $(p_1, s_1, k+1)$ yields the tuple (p_1, s_1, k), which may only be replaced in $\tilde{D}^{(k)}$ by a tuple of smaller size, but with a profit still in $L_\gamma^{(\bar{k},b)}$. Thus, there must be a tuple $(p_2, s_2, k) \in \tilde{D}^{(k)}$ with $s_2 \leq s_1$ and $p_2 \in L_\gamma^{(\bar{k},b)}$. Let now $(p, s, k) \in D^{(k)}$ be the tuple that dominates (p_2, s_2, k) (which can of course be (p_2, s_2, k) itself), i.e. $p \geq p_2$ and $s \leq s_2$. For the profit, we have

$$\begin{aligned} p \geq p_2 \geq p_1 - 2^{\bar{k}} K_b &\overset{p_1 \neq 0}{=} p_1 \cdot \left(1 - \frac{2^{\bar{k}} K_b}{p_1}\right) \overset{p_1 \geq 2^{\bar{k}} T_b}{\geq} p_1 \cdot \left(1 - \frac{2^{\bar{k}} K_b}{2^{\bar{k}} T_b}\right) \\ &\geq \bar{p} \cdot \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2-k} . \end{aligned}$$

The lower bound on the profit is therefore true for (p, s, k). As for the bound on the size, we have $s \leq s_2 \leq s_1 \leq \bar{s}$.

Consider now the second case where $(\bar{p}, \bar{s}, k) \in F^{(k)}$, but $(\bar{p}, \bar{s}, k+1) \notin F^{(k+1)}$. Hence, $(\bar{p}, \bar{s}, k)$ is a new (non-dominated) tuple with $(\bar{p}, \bar{s}, k) = (\tilde{p} + p(\tilde{a}), \tilde{s} + s(\tilde{a}), k)$ for the right item $\tilde{a} \in \tilde{I}^{(k,b)}$ and tuple $(\tilde{p}, \tilde{s}, k+1) \in F^{(k+1)}$. By the induction hypothesis, there must be a tuple $(p_1, s_1, k+1) \in D^{(k+1)}$ so that we have $p_1 \geq \tilde{p} \cdot (1 - \frac{K_b}{T_b})^{\kappa_{\max}^{(b)}+2-(k+1)}$ and $s_1 \leq \tilde{s}$. Thus, the following inequality holds:

$$\begin{aligned} p_1 + p(\tilde{a}) &\geq p(\tilde{a}) + \tilde{p} \cdot \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2-(k+1)} \geq (p(\tilde{a}) + \tilde{p}) \cdot \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2-(k+1)} \\ &= \bar{p} \cdot \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2-(k+1)} . \end{aligned}$$

There are two possibilities: either $k \geq \kappa_{\min}^{(b)} - 2$, i.e. $p(\tilde{a}) \geq 2^{\kappa_{\min}^{(b)}-2} T_b$ holds, and $p_1 + p(\tilde{a}) \geq 2^{\kappa_{\min}^{(b)}-2} T_b$ directly follows. Otherwise, we have $k \leq \kappa_{\min}^{(b)} - 3$. Then,

$(\bar{p}, \bar{s}, k) = (\tilde{p} + p(\tilde{a}), \tilde{s} + s(\tilde{a}), k) \neq (0, 0, k)$ implies that $(\tilde{p}, \tilde{s}, k+1) \neq (0, 0, k+1)$ because the tuple $(0, 0, k+1)$ is not used to form any new tuple in $\tilde{D}^{(k)}$ and therefore in $D^{(k)}$. Because of $p_1 \geq \tilde{p} \cdot (1 - \frac{K_b}{T_b})^{\kappa_{\max}^{(b)}+2-(k+1)}$, this again implies that $p_1 \neq 0$ and therefore $p_1 + p(\tilde{a}) \geq p_1 \geq 2^{\kappa_{\min}^{(b)}-2} T_b$ as seen in Lemma 5.33.

Hence, there is an index pair $(\bar{k}, \gamma)$ such that $p_1 + p(\tilde{a}) \in L_\gamma^{(\bar{k},b)}$. Similar to above, the tuple $(p_1 + p(\tilde{a}), s_1 + s(\tilde{a}), k)$ is formed during the construction of $\tilde{D}^{(k)}$. It may only be replaced by a tuple of smaller size. Hence, there must be $(p_2, s_2, k) \in \tilde{D}^{(k)}$ with $p_2 \in L_\gamma^{(\bar{k},b)}$. Let $(p, s, k) \in D^{(k)}$ be the tuple that dominates (p_2, s_2, k). We get

$$\begin{aligned} p \quad &\geq \quad p_2 \geq p_1 + p(\tilde{a}) - 2^{\bar{k}} K_b \overset{p_1+p(\tilde{a})\neq 0}{=} (p_1 + p(\tilde{a})) \cdot \left(1 - \frac{2^{\bar{k}} K_b}{p_1 + p(\tilde{a})}\right) \\ &\overset{p_1+p(\tilde{a})\geq 2^{\bar{k}} T_b}{\geq} (p_1 + p(\tilde{a})) \cdot \left(1 - \frac{2^{\bar{k}} K_b}{2^{\bar{k}} T_b}\right) \\ &\geq \quad \bar{p} \cdot \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2-k} . \end{aligned}$$

For the bound on the size, we have similar to above $s \leq s_2 \leq s_1 + s(\tilde{a}) \leq \tilde{s} + s(\tilde{a}) = \bar{s}$.□

Corollary 5.35. *For every $v \leq c_{\max}^{(b)}$, there is a tuple $(p, s, 0) \in D^{(0)}$ such that $s \leq v$ and*

$$p \geq \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}_{\mathrm{St}}\left(\tilde{I}^{(b)} \cup I_{S,\mathrm{red-c}}^{(b)}, v\right) .$$

Proof. Lemma 5.32 states that there is a tuple $(\bar{p}, \bar{s}, 0) \in F^{(0)}$ with $\bar{s} \leq v$ and $\bar{p} = \overline{\mathrm{OPT}}_{\mathrm{St}}(\tilde{I}^{(b)} \cup I_{S,\mathrm{red-c}}^{(b)}, v)$. Theorem 5.34 implies that there is a tuple $(p, s, 0) \in D^{(0)}$ with the desired property. □

Theorem 5.36. *Algorithm 5.6 constructs all tuple sets $D^{(k)}$ for $k = \kappa_{\max}^{(b)} + 2, \ldots, 0$ in time $\mathcal{O}(\kappa_{\max}^{(b)} \cdot \gamma_0^2)$. The space needed for the algorithm and to save the $D^{(k)}$ as well as the backtracking information is in $\mathcal{O}(\gamma_0 \cdot \kappa_{\max}^{(b)})$, where $D^{(k)}$ contains $\mathcal{O}(\gamma_0)$ tuples.*

Proof. Let us first bound the space complexity. The profit interval $[2^{\kappa_{\min}^{(b)}-2} T_b, 2\bar{P}_{c_{\max}^{(b)}}]$ is partitioned into $\kappa_{\max}^{(b)} + 1 - (\kappa_{\min}^{(b)} - 2) + 1 \in \mathcal{O}(\kappa_{\max}^{(b)} - \kappa_{\min}^{(b)}) = \mathcal{O}(1)$ intervals $L^{(\bar{k},b)}$ (see Lemma 5.13). Each $L^{(\bar{k},b)}$ has γ_0 (or $\gamma_1 \leq \gamma_0$ in the case of $L^{(\kappa_{\max}^{(b)}+1,b)}$) sub-intervals $L_\gamma^{(\bar{k},b)}$. The set $D^{(k)}$ saves at most one tuple with the corresponding backtracking information for every $L_\gamma^{(\bar{k},b)}$ or the information that a tuple does not exist. Hence, one $D^{(k)}$ has a cardinality in $\mathcal{O}((\kappa_{\max}^{(b)} - \kappa_{\min}^{(b)}) \cdot \gamma_0) = \mathcal{O}(\gamma_0)$. Thus, the space needed for all $D^{(k)}$ and their backtracking data is bounded by $\mathcal{O}((\kappa_{\max}^{(b)} + 3) \cdot (\kappa_{\max}^{(b)} - \kappa_{\min}^{(b)}) \cdot \gamma_0) = \mathcal{O}(\gamma_0 \cdot \kappa_{\max}^{(b)})$. All other information of the algorithm are only temporarily saved and need $\mathcal{O}(1)$.

The loops dominate the running time. Apart from removing the dominated tuples, they need in total

$$\mathcal{O}\Big(\kappa_{\max}^{(b)} \cdot \gamma_0 \cdot (\kappa_{\max}^{(b)} - \kappa_{\min}^{(b)})\Big) + \mathcal{O}\Big(\kappa_{\max}^{(b)} \cdot \gamma_0 \cdot \gamma_0 \cdot (\kappa_{\max}^{(b)} - \kappa_{\min}^{(b)})\Big) = \mathcal{O}\Big(\gamma_0^2 \cdot \kappa_{\max}^{(b)}\Big) \ ,$$

where we have used that $|\tilde{I}^{(k,b)}| \in \mathcal{O}(\gamma_0)$ by the definition of $\tilde{I}^{(k,b)}$ in Section 5.6 and that $|\tilde{I}^{(\kappa_{\min}^{(b)}+2,b)}| = |I_{S,\text{red}-\text{c}}^{(b)}| \in \mathcal{O}(\gamma_0)$ by Theorem 5.30.

As stated in [63] and Lemma 3.7, non-dominated tuples (p, s, k) can be removed in linear time in the number of tuples if the tuples are different and sorted by profit. This is the case because every tuple in $D^{(k)}$ is stored in an array sorted according to the corresponding $p(\bar{k}, \gamma)$. The total time for removing the dominated tuples from all $D^{(k)}$ for $k \in \{0, \dots, \kappa_{\max}^{(b)} + 2\}$ is therefore in $\mathcal{O}(\gamma_0 \cdot \kappa_{\max}^{(b)})$, which is dominated by the overall running time. □

Remark 5.37. In Section 4.7, the dynamic program is slightly different: the interval $[2^{\kappa-2}T, 2P_0]$ that corresponds to $[2^{\kappa_{\min}^{(b)}-2}T_b, 2\bar{P}_{c_{\max}^{(b)}}]$ is not partitioned into sub-intervals $L_\gamma^{(\bar{k},b)}$. Instead, it is divided into sub-intervals $\tilde{L}_\xi^{(\kappa-2)}$ of the same length $2^{\kappa-2}K$ with $\tilde{L}_\xi^{(\kappa-2)} = [2^{\kappa-2}T + \xi \cdot 2^{\kappa-2}K,\ 2^{\kappa-2}T + (\xi+1) \cdot 2^{\kappa-2}K)$. The dynamic program works in the same way as presented in this chapter, i.e. only the smallest tuple with a profit in $\tilde{L}_\xi^{(\kappa-2)}$ is kept. The number of intervals $\tilde{L}_\xi^{(\kappa-2)}$ is still in $\mathcal{O}(\frac{2^{\kappa-2}T}{2^{\kappa-2}K}) = \mathcal{O}(\frac{T}{K})$, which corresponds to $\mathcal{O}(\gamma_0)$ in our algorithm. Hence, an improved partitioning of $[2^{\kappa-2}T, 2P_0]$ as done in this chapter does not change the asymptotic time and space complexity of the FPTAS in Chapter 4. The values of T_b and K_b we will derive later will also asymptotically be the same as T and K in Chapter 4, and vice versa.

However, the fact that p_1 or $p_1 + p(\tilde{a})$ is at least $2^{\kappa_{\min}^{(b)}-2}T_b$ is no longer necessary for the proof of Theorem 5.34, i.e. the bound on the approximation ratio of the approximate dynamic program. On the other hand, this was essential for the dynamic program in Section 4.7 as explained in Remark 4.23. Still, the solution structure improves the running time of Algorithm 5.6: we only have tuples (p, s, k) with $p \geq 2^{\kappa_{\min}^{(b)}-2}T_b$. If this were not the case, the dynamic program would also have to generate tuples (p, s, k) with $p < 2^k T_b$ for $k \leq \kappa_{\min}^{(b)} - 3$, which would increase the running time and storage space by an additional factor of $\mathcal{O}(\kappa_{\max}^{(b)})$: the time complexity would be $\mathcal{O}((\kappa_{\max}^{(b)})^2 \cdot \gamma_0^2)$ and the space complexity $\mathcal{O}((\kappa_{\max}^{(b)})^2 \cdot \gamma_0)$ as can be seen in the proof of Theorem 5.36. We leave it as an open question whether the approximate dynamic programming of this section leads to an overall faster algorithm (e.g. because less pre-processing may be necessary).[1]

[1]The observation that p_1 or $p_1 + p(\tilde{a})$ no longer needs to be at least $2^{\kappa_{\min}^{(b)}-2}T_b$ for the approximation ratio was made during the final proofreading of this thesis. An improvement of the whole FPTAS

For completeness, we present a short overview how p_1 or $p_1 + p(\tilde{a}) \geq 2^{\kappa_{\min}^{(b)}-2} T_b$ is achieved. First, the glued item set $\tilde{I}^{(b)}$ with its structured solution (Definition 5.19 and Theorem 5.20) is constructed. Assumption 5.5 with $\frac{1}{4}\bar{P}_{c_{\min}^{(b)}} \geq T_b$ allows us to prove Lemma 5.24 and to introduce $I_{S,\text{red}-\text{c}}^{(b)}$ with its items $a_{\gamma,\text{e}-\text{c}}^{(k,b)} \geq \frac{1}{4}\bar{P}_{c_l} \geq 2^{\kappa_{\min}^{(b)}-2} T_b$. Hence, we have structured solutions with a lower bound (Definition 5.28 and Theorem 5.29). This shows that $p_1 \geq 2^{\bar{k}} T_b \geq 2^{\kappa_{\min}^{(b)}-2} T_b$ or $p_1 + p(\tilde{a}) \geq 2^{\bar{k}} T_b \geq 2^{\kappa_{\min}^{(b)}-2} T_b$ (see also Lemma 5.32 and 5.33).

5.8 The Algorithm

We can now put together the approximation algorithm, which is shown in Algorithm 5.7.

5.8.1 Solution Quality and Check of Assumptions

Lemma 5.38. *Take one $C_b \neq \emptyset$ and let $c_l \in C_b$. The value $\bar{P}' = \bar{P}'(c_l)$ for c_l in Step 13 of Algorithm 5.7 is at least*

$$\bar{P}'(c_l) \geq \left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} \overline{\text{OPT}}\,(I, c_l) - \left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} T_b \ .$$

The corresponding set of items fits into c_l.

Proof. $(p, s, 0)$ represents an item set J' of size s taken from $\tilde{I}^{(b)} \cup I_{S,\text{red}-\text{c}}^{(b)}$. If items $\tilde{a} \in \tilde{I}^{(b)}$ derived from gluing are part of the solution, their ungluing does not change the total size nor the total profit (see Remark 5.18). This is obviously also true for $\tilde{a} \in I_{S,\text{red}-\text{c}}^{(b)}$. Hence, the unglued items together with the items $a_{\text{eff}}^{(l)}$ from $\overline{\text{OPT}}(\{a_{\text{eff}}^{(l)}\}, c_l - s)$ yields the set J that fits into c_l.

We prove the solution quality. Let v be the volume from Theorem 5.29. Corollary 5.35 guarantees the existence of one $(p, s, 0) \in D^{(0)}$ with $s \leq v$ such that

$$p \geq \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\text{OPT}}_{\text{St}}\left(\tilde{I}^{(b)} \cup I_{S,\text{red}-\text{c}}^{(b)}, v\right) \ .$$

may not be straightforward: the solution structure shows that we have to add only single items in $\tilde{I}^{(k,b)}$ to the tuples in $D^{(k+1)}$ to get the new tuples $D^{(k)}$. Without the structure, we also may have to add several items to each tuple, which would complicate the approximate dynamic program.

Algorithm 5.7: The complete algorithm

Input: Item set I, sorted knapsacks $C = \{c_1, \ldots, c_M\}$

Output: Profit P, solution set J and knapsack size c_{sol}

Determine $a^{(l)}_{\text{meff}}$, the sets S_l, the approximations $\bar{P}_{c_l}$ and P_0 and adapt C accordingly (Algorithm 5.1);
Partition C into the sets C_b with Algorithm 5.2;
Set $P := 0$, $c_{\text{sol}} := \emptyset$, $s := 0$, $J := \emptyset$ and $J' := \emptyset$;
for *all* $C_b \neq \emptyset$ **do**
    Set NewBestValueFound := *false*;
    Define T_b and K_b and determine $\kappa^{(b)}_{\min}$ and $\kappa^{(b)}_{\max}$;
    Find the set $I^{(b)}_L$ and the $a^{(l)}_{\text{eff}}$ with Algorithm 5.3;
    Reduce $I^{(b)}_L$ to $I^{(b)}_{L,\text{red}}$ with Algorithm 5.4;
    Construct the $\tilde{I}^{(k,b)}$ and $\tilde{I}^{(b)}$ with Algorithm 5.5;
    Define the $a^{(l)}_{\text{eff}-\text{c}}$ and reduce them to $I^{(b)}_{S,\text{red}-\text{c}}$ (see Theorem 5.30);
    Determine the tuples $D^{(\kappa^{(b)}_{\max}+2)}, \ldots, D^{(0)}$ with the approximate dynamic program (see Algorithm 5.6);
    for *all* $c_l \in C_b$ **do**
        Find $(p, s, 0) \in D^{(0)}$ such that
        $\bar{P}' := p + \overline{\text{OPT}}(\{a^{(l)}_{\text{eff}}\}, c_l - s) = \max_{(p',s',0)\in D^{(0)}} p' + \overline{\text{OPT}}(\{a^{(l)}_{\text{eff}}\}, c_l - s')$;
        if $\frac{\bar{P}'}{c_l} > P$ **then**
            Set $P := \frac{\bar{P}'}{c_l}$, $c_{\text{sol}} := c_l$ as well as $(\bar{p}, \bar{s}, 0) := (p, s, 0)$;
            NewBestValueFound := *true*;
    if NewBestValueFound = *true* **then**
        Backtrack the tuple $(\bar{p}, \bar{s}, 0)$ to find the large items $J' \subseteq \tilde{I}^{(b)} \cup I^{(b)}_{S,\text{red}-\text{c}}$ of the corresponding structured solution with a lower bound;
        Undo the gluing for the items in J' to get J;
        Add the items of $\overline{\text{OPT}}(\{a^{(l)}_{\text{eff}}\}, c_{\text{sol}} - \bar{s})$ to J;
return *P, J and* c_{sol};

Moreover, we have $\overline{\mathrm{OPT}}(\{a_{\mathrm{eff}}^{(l)}\}, c_l - s) \geq \overline{\mathrm{OPT}}(\{a_{\mathrm{eff}}^{(l)}\}, c_l - v)$ because $c_l - s \geq c_l - v$. Thus, the following inequality holds for this $(p, s, 0)$:

$$\begin{aligned}
& p + \overline{\mathrm{OPT}}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - s\right) \\
& \quad \geq \quad \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}_{\mathrm{St}}\left(\tilde{I}^{(b)} \cup I_{\mathrm{S,red-c}}^{(b)}, v\right) + \mathrm{OPT}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v\right) \\
& \quad \geq \quad \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \left(\overline{\mathrm{OPT}}_{\mathrm{St}}\left(\tilde{I}^{(b)} \cup I_{\mathrm{S,red-c}}^{(b)}, v\right) + \mathrm{OPT}\left(\left\{a_{\mathrm{eff}}^{(l)}\right\}, c_l - v\right)\right) \\
& \overset{\text{Thm. 5.29}}{\geq} \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \left(\left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} \overline{\mathrm{OPT}}\left(I, c_l\right) - \left(1 - \frac{K_b}{T_b}\right)^{\kappa_{\max}^{(b)}+2} T_b\right) \\
& \quad = \quad \left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} \overline{\mathrm{OPT}}\left(I, c_l\right) - \left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} T_b \ .
\end{aligned}$$

Since the maximum over all $(p', s', 0) \in D^{(0)}$ is taken, the lemma follows. □

It is not too difficult to see that Algorithm 5.7 is an implementation of `MaxSolution` (i.e. Algorithm 3.1). We only have to make sure that $\bar{P}'(c_l) \geq (1-\varepsilon)\overline{\mathrm{OPT}}(I, c_l)$ for every c_l. Thus, we have to choose T_b and K_b for C_b such that the assumptions on T_b and K_b we have used so far are all satisfied and

$$\left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} \overline{\mathrm{OPT}}\left(I, c_l\right) - \left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} T_b \overset{!}{\geq} (1-\varepsilon)\,\overline{\mathrm{OPT}}(I, c_l) \ . \tag{5.16}$$

We use Assumption 5.3 to get similar to the proof of Theorem 4.26

$$\begin{aligned}
& \left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} \overline{\mathrm{OPT}}\left(I, c_l\right) - \left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} T_b \\
& \quad \geq \left(1 - (2\kappa_{\max}^{(b)} + 4)\frac{K_b}{T_b}\right) \overline{\mathrm{OPT}}\left(I, c_l\right) - \left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} T_b \overset{!}{\geq} (1-\varepsilon)\,\overline{\mathrm{OPT}}(I, c_l) \ .
\end{aligned} \tag{5.17}$$

Similar to Lawler [63] and to Subsection 3.5.3 (see (3.18)), the following conditions have therefore to be satisfied:

$$\left(2\kappa_{\max}^{(b)} + 4\right)\frac{K_b}{T_b} \leq \lambda\varepsilon \quad \text{and} \quad \left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} T_b \leq (1-\lambda)\,\varepsilon\overline{\mathrm{OPT}}\left(I, c_l\right) \tag{5.18}$$

for one $\lambda \in (0,1)$ and all $c_l \in C_b$.

Let us set $T_b = (1-\lambda)\varepsilon\bar{P}_{c_{\min}^{(b)}}$. It implies the second condition so that the first condition is satisfied for

$$K_b = \frac{\lambda\varepsilon T_b}{2\kappa_{\max}^{(b)} + 4} = \frac{\lambda(1-\lambda)}{2\kappa_{\max}^{(b)} + 4}\varepsilon^2\bar{P}_{c_{\min}^{(b)}} \ .$$

We set $\lambda = \frac{1}{2}$, which maximizes T_b and K_b and therefore minimizes the value of γ_0 and the number of $L_\gamma^{(k,b)}$ (again similar to Subsection 3.5.3 and [63]). We therefore have

$$T_b = \frac{1}{2}\varepsilon \bar{P}_{c_{\min}^{(b)}} \quad \text{and} \quad K_b = \frac{\varepsilon}{4}\frac{1}{\kappa_{\max}^{(b)}+2}T_b \ . \tag{5.19}$$

Theorem 5.39. *Algorithm 5.7 finds a solution of value at least* $(1-\varepsilon)\mathrm{OPT}(I)$.

Proof. For every C_b, we have $\bar{P}'(c_l) \geq (1-\varepsilon)\overline{\mathrm{OPT}}\,(I, c_l)$ for all $c_l \in C_b$ with $\bar{P}_{c_l} \geq \frac{1}{2}P_0$. (Algorithm 5.7 does not determine solutions for $\bar{P}_{c_l} < \frac{1}{2}P_0$ because such knapsack sizes are discarded, which is justified by Corollary 5.5.) Hence, Algorithm 5.7 is a correct implementation of `MaxSolution`, i.e. Algorithm 3.1, which finds a $(1-\varepsilon)$ approximate solution according to Theorem 3.6. □

Theorem 5.40. *We have* $\kappa = \kappa_{\min}^{(b)}$ *because of Assumption 5.1. Assumptions 5.2 to 5.5 are satisfied by the choice of* T_b *and* K_b*. Finally,* $\gamma_0 \in \mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$.

Proof. Note that Assumption 5.1 only states that $\varepsilon = \frac{1}{2^{\kappa-1}}$ for $\kappa \geq 3$ whereas the other assumptions ask for special properties of T_b, K_b, and γ_0.

Since $\varepsilon = \frac{1}{2^{\kappa-1}}$ and $2^{\kappa_{\min}^{(b)}+1}T_b > \bar{P}_{c_{\min}^{(b)}} \geq 2^{\kappa_{\min}^{(b)}}T_b$, the identity $\kappa = \kappa_{\min}^{(b)}$ follows immediately. Assumption 5.3 is obviously satisfied: as $\frac{\varepsilon}{4}\frac{1}{\kappa_{\max}^{(b)}+2} \cdot K_b = T_b$ holds by Definition (5.19), we get $\frac{1}{\gamma_0+1} \overset{!}{=} \frac{1}{4}\frac{\varepsilon}{\kappa_{\max}^{(b)}+2} = \frac{1}{2^{\kappa+1}}\frac{1}{\kappa_{\max}^{(b)}+2}$, i.e. $\gamma_0 = 2^{\kappa+1}(\kappa_{\max}^{(b)}+2) - 1 \in \mathbb{N}_{\geq 1}$ and $\gamma_0 \in \mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$.

It is also obvious that

$$\bar{P}_{c_{\min}^{(b)}} \geq \frac{1}{4}\bar{P}_{c_{\min}^{(b)}} \overset{\varepsilon\leq 1/4}{\geq} \varepsilon\bar{P}_{c_{\min}^{(b)}} = \frac{1}{2^{\kappa-1}}\bar{P}_{c_{\min}^{(b)}} = \frac{1}{2^{\kappa_{\min}^{(b)}-1}}\bar{P}_{c_{\min}^{(b)}} \geq \frac{1}{2^{\kappa_{\min}^{(b)}}}\bar{P}_{c_{\min}^{(b)}} \geq T_b \ ,$$

i.e. the Assumptions 5.2 and 5.5 are true. As for Assumption 5.4, it is easy to see that it is satisfied because of (5.17). □

We finish this subsection with a remark regarding the overall approximation ratio.

Remark 5.41. The multiplicative error of $(1-\frac{K_b}{T_b})^{2\kappa_{\max}^{(b)}+4}$ in front of $\overline{\mathrm{OPT}}(I, c_l)$ in Lemma 5.38 is caused by the multiplicative error $(1-\frac{K_b}{T_b})$ that we make $2\kappa_{\max}^{(b)}+4$ times. Such an error occurs when I is replaced by $I_{L,\mathrm{red}}^{(b)}$ at the beginning (Lemma 5.15), in each of the $\kappa_{\max}^{(b)}+1$ iterations in which $\tilde{I}^{(b)}$ is constructed (Theorem 5.20), and in $\kappa_{\max}^{(b)}+2$ of the $\kappa_{\max}^{(b)}+3$ iterations of the dynamic program (Theorem 5.34 and Corollary 5.35).

The additive error of $(1 - \frac{K_b}{T_b})^{2\kappa_{\max}^{(b)}+4} T_b$ is mainly due to the additive error of T_b in Lemma 5.15. The multiplicative factor appears when the small items are combined with the large ones. First, T_b is multiplied by $(1 - \frac{K_b}{T_b})^{\kappa_{\max}^{(b)}+2}$ in Theorem 5.29, and it is again multiplied by the same value in Lemma 5.38. Interestingly, the error in Theorem 5.29 has the same bound as the error in Lemma 5.23. In Theorem 5.29, the reduced set $I_{S,\text{red}-\text{c}}^{(b)}$ is used for the structured solutions with a lower bound. The error of $(1 - \frac{K_b}{T_b})$ caused by this is subsumed by the multiplicative error $(1 - \frac{K_b}{T_b})^{\kappa_{\max}^{(b)}+2}$.

5.8.2 Running Time and Space Complexity

Lemma 5.42. *Fix one C_b. The inner for-loop of Algorithm 5.7 (Steps 12–16) has for one $c_l \in C_b$ a running time in $\mathcal{O}(\gamma_0)$ and a space complexity in $\mathcal{O}(1)$. For all $c_l \in C_b$, the running time is in $\mathcal{O}(|C_b|\gamma_0) = \mathcal{O}(|C_b|\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ and the space needed still in $\mathcal{O}(1)$.*

Proof. The space complexity is easy to see: only $\bar{P}'$, P, c_{sol}, and $(\bar{p}, \bar{s}, 0)$ are saved.

The if-condition needs time $\mathcal{O}(1)$ in every iteration of the for-loop (even if the body is executed). One value $p' + \overline{\text{OPT}}(\{a_{\text{eff}}^{(l)}\}, c_l - s')$ can be determined in $\mathcal{O}(1)$. Thus, the maximum can be found for one c_l in $\mathcal{O}(|D^{(0)}|) = \mathcal{O}(\gamma_0)$ (see Theorem 5.36). For all $c_l \in C_b$, the running time is in $\mathcal{O}(|C_b| \cdot \gamma_0) = \mathcal{O}(|C_b| \cdot \frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ (see Theorem 5.40). □

Lemma 5.43. *For one C_b, the outer for-loop of Algorithm 5.7 (Steps 4–20) needs time in $\mathcal{O}(\frac{1}{\varepsilon^2}\log^3\frac{1}{\varepsilon} + |C_b| \cdot \frac{1}{\varepsilon}\log\frac{1}{\varepsilon} + n)$ and storage space in $\mathcal{O}(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon} + |C_b| + n)$.*

Proof. The definition of T_b, K_b and determining $\kappa_{\min}^{(b)}$ and $\kappa_{\max}^{(b)}$ can be done in time and space $\mathcal{O}(1)$.

Algorithm 5.3 needs time and space in $\mathcal{O}(n + |C_b|)$ as seen in Theorem 5.11. Then, Algorithm 5.4 reduces $I_L^{(b)}$ to $I_{L,\text{red}}^{(b)}$ in time $\mathcal{O}(n + \gamma_0 \cdot \kappa_{\max}^{(b)})$ and space $\mathcal{O}(\gamma_0 \cdot \kappa_{\max}^{(b)})$ as explained in Theorem 5.16.

Algorithm 5.5 constructs the sets $\tilde{I}^{(k,b)}$ and $\tilde{I}^{(b)}$ in time $\mathcal{O}(\gamma_0^2 \cdot \kappa_{\max}^{(b)})$ and space $\mathcal{O}(\gamma_0 \cdot \kappa_{\max}^{(b)})$. This is shown in Theorem 5.30, which also states that the items $a_{\text{eff}-\text{c}}^{(l)}$ and therefore the set $I_{S,\text{red}-\text{c}}^{(b)}$ can be found in time and space $\mathcal{O}(|C_b| + \gamma_0)$.

The approximate dynamic programming (Algorithm 5.6) needs time in $\mathcal{O}(\gamma_0^2 \cdot \kappa_{\max}^{(b)})$ and space in $\mathcal{O}(\gamma_0 \cdot \kappa_{\max}^{(b)})$ as stated in Theorem 5.36.

The inner for-loop needs time in $\mathcal{O}(|C_b| \cdot \gamma_0)$ and space in $\mathcal{O}(1)$ as seen above in Lemma 5.42.

The backtracking is in $\mathcal{O}(\kappa_{\max}^{(b)})$: every entry Backtrack$(p', s', k)$ for $k = 0, \ldots, \kappa_{\max}^{(b)} + 2$ states whether the tuple was formed by adding an item $\tilde{a} \in \tilde{I}^{(k,b)}$ and with which tuple $(p'', s'', k+1)$ to continue. Hence, the item set J' has at most $\mathcal{O}(\kappa_{\max}^{(b)})$ items in $\tilde{I}^{(b)} \cup I_{S,\text{red}-\text{c}}^{(b)}$.

The time and space for the ungluing have to be bounded. Consider one item $\tilde{a} \in \tilde{I}^{(b)}$. The backtracking information Backtrack($\tilde{a}$) returns two items $(\bar{a}_1, \bar{a}_2)$ (with $\bar{a}_1, \bar{a}_2 \in I^{(b)}_{L,\text{red}} \cup \tilde{I}^{(b)}$) on which the backtracking can be recursively applied. The recursive ungluing of the items can be represented as a binary tree where the root is the original item $\tilde{a}$ and the (two) children of each node are the items $(\bar{a}', \bar{a}'')$ returned by the backtracking information. The leaves of the tree are the original items in $I^{(b)}_{L,\text{red}}$. This binary tree obviously has a height in $\mathcal{O}(\kappa^{(b)}_{\max})$ because the children $(\bar{a}', \bar{a}'')$ for one $\bar{a} \in \tilde{I}^{(k,b)}$ are in $\tilde{I}^{(k-1,b)} \cup I^{(b)}_{L,\text{red}}$. A binary tree of height $\mathcal{O}(\kappa^{(b)}_{\max})$ has at most $\mathcal{O}(2^{\kappa^{(b)}_{\max}})$ nodes. The backtracking or ungluing of $\tilde{a}$ can therefore be done in time and space $\mathcal{O}(2^{\kappa^{(b)}_{\max}})$, which also includes saving the items $\bar{a} \in I^{(b)}_{L,\text{red}}$ of which $\tilde{a}$ is composed. Since J' has $\mathcal{O}(\kappa^{(b)}_{\max})$ items, the original items $I^{(b)}_{L,\text{red}}$ of the approximate solution can be found in time and space $\mathcal{O}(\kappa^{(b)}_{\max} \cdot 2^{\kappa^{(b)}_{\max}})$.

Finally, the number of items $\text{OPT}(\{a^{(l)}_{\text{eff}}\}, c_{\text{sol}} - \bar{s})$ can be found in $\mathcal{O}(1)$, which is also the time needed to unglue $a^{(k,b)}_{\gamma,\text{e-c}}$ should it be part of the solution.

Hence, we get an overall running time in $\mathcal{O}(\gamma_0^2 \cdot \kappa^{(b)}_{\max} + \gamma_0 \cdot |C_b| + \kappa^{(b)}_{\max} \cdot 2^{\kappa^{(b)}_{\max}} + n)$ and a space complexity in $\mathcal{O}(|C_b| + \gamma_0 \cdot \kappa^{(b)}_{\max} + \kappa^{(b)}_{\max} \cdot 2^{\kappa^{(b)}_{\max}} + n)$. The lemma follows by using $\kappa^{(b)}_{\max} \leq \kappa^{(b)}_{\min} + 2 = \kappa + 2$ (see Lemma 5.13 and Theorem 5.40), the property $\gamma_0 \in \mathcal{O}(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ (see again Theorem 5.40), and $2^\kappa = \frac{2}{\varepsilon}$, i.e. $\kappa = \kappa^{(b)}_{\min} \in \mathcal{O}(\log \frac{1}{\varepsilon})$. □

We are now able to prove Theorem 5.1, i.e. the overall time and space complexity.

Proof of Theorem 5.1. As stated in Theorem 5.7, Algorithm 5.1 has a time complexity in $\mathcal{O}(M + n \log M)$ and a space complexity in $\mathcal{O}(M + n)$. This also bounds time and space of Algorithm 5.2 (see Theorem 5.10).

As stated by Lemma 5.43, the overall running time of the outer for-loop is in

$$\begin{aligned}
&\mathcal{O}\left(\sum_{C_b}\left(\frac{1}{\varepsilon^2}\log^3\frac{1}{\varepsilon} + |C_b| \cdot \frac{1}{\varepsilon}\log\frac{1}{\varepsilon} + n\right)\right) \\
&\quad = \mathcal{O}\Bigg(\min\left\{\log\left\lfloor\frac{1}{c_{\min}}\right\rfloor + 1, M\right\} \cdot \frac{1}{\varepsilon^2}\log^3\frac{1}{\varepsilon} + M \cdot \frac{1}{\varepsilon}\log\frac{1}{\varepsilon} \\
&\qquad + \min\left\{\log\left\lfloor\frac{1}{c_{\min}}\right\rfloor + 1, M\right\} n\Bigg) \ ,
\end{aligned}$$

where we have used that the number of C_b is bounded by $\min\{\log\lfloor\frac{1}{c_{\min}}\rfloor + 1, M\}$ as stated at the beginning of Subsection 5.5.1. The overall running time follows.

For the space complexity, we use the fact that only P, J and c_{sol} have to be stored permanently while the other values and sets saved during one execution of the outer for-loop can be discarded. Hence, the space complexity for the outer for-loop is

bounded by the space complexity of one of its runs, which is stated in Lemma 5.43. We get the overall storage space bound. □

You may recall that we wanted the bound $(1-\frac{K_b}{T_b})^{2\kappa_{\max}^{(b)}+4} T_b \leq (1-\lambda)\varepsilon\,\overline{\mathrm{OPT}}(I,c_l)$ to be satisfied (see (5.18)). We then set $T_b = (1-\lambda)\varepsilon \bar{P}_{c_{\min}^{(b)}}$, which was sufficient for the Bound (5.16) on the approximation ratio. However, our choice of T_b seems quite generous at first sight: can even a larger value of T_b be chosen? This would also imply a larger K_b and a smaller value for γ_0 and might therefore improve the overall running time.

We now prove that this is not the case. The following proof was found in collaboration with the student Dennis Papesch. The main idea is due to him.

Theorem 5.44. *If the Bound* (5.16) *has to be satisfied (together with the Assumptions 5.1–5.5), then* $T_b \in \mathcal{O}(\varepsilon \cdot \overline{\mathrm{OPT}}(I,c_l)) = \mathcal{O}(\varepsilon \bar{P}_{c_l}) = \mathcal{O}(\varepsilon \bar{P}_{c_{\min}^{(b)}})$ *holds for* $\varepsilon \to 0$.

Proof. The identities $\mathcal{O}(\varepsilon \cdot \overline{\mathrm{OPT}}(I,c_l)) = \mathcal{O}(\varepsilon \bar{P}_{c_l}) = \mathcal{O}(\varepsilon \bar{P}_{c_{\min}^{(b)}})$ follow from Theorem 5.4 and Corollary 5.5 as well as Theorem 5.7. We prove that $T_b \in \mathcal{O}(\varepsilon \cdot \overline{\mathrm{OPT}}(I,c_l))$.

Clearly, the Bound (5.16) implies that

$$\left(1-\frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} \overline{\mathrm{OPT}}\,(I,c_l) \geq (1-\varepsilon)\,\overline{\mathrm{OPT}}\,(I,c_l) + \alpha$$
$$\text{for} \quad \alpha := \left(1-\frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} T_b \geq 0 \ .$$

We assume here that $T_b \geq 0$ together with $\overline{\mathrm{OPT}}\,(I,c_l) > 0$. The bound can be rewritten as

$$\left(1-\frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} \overline{\mathrm{OPT}}\,(I,c_l) \geq \left(1-\varepsilon\cdot\underbrace{\left(1-\frac{\alpha}{\varepsilon\cdot\overline{\mathrm{OPT}}\,(I,c_l)}\right)}_{=:\lambda}\right)\cdot\overline{\mathrm{OPT}}\,(I,c_l) \ .$$

Note that

$$\overline{\mathrm{OPT}}\,(I,c_l) \geq \left(1-\frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} \overline{\mathrm{OPT}}\,(I,c_l) \geq (1-\lambda\varepsilon)\,\overline{\mathrm{OPT}}\,(I,c_l)$$

so that $\lambda \geq 0$ holds, where we use Assumption 5.3, i.e. $\frac{K_b}{T_b} < 1$. By definition, we also see that $\lambda \leq 1$. To sum up, we have—similar to (5.18)—for every $\varepsilon > 0$

$$\left(1-\frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} \overline{\mathrm{OPT}}\,(I,c_l) \geq (1-\lambda\varepsilon)\,\overline{\mathrm{OPT}}\,(I,c_l)$$
$$\text{and} \quad \left(1-\frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} T_b = \alpha = (1-\lambda)\,\varepsilon\cdot\overline{\mathrm{OPT}}\,(I,c_l)$$

for $\lambda = \lambda(\varepsilon) \in [0,1]$. This implies

$$\left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} \geq (1 - \lambda\varepsilon) \quad . \tag{5.20}$$

Let $\varepsilon \leq \frac{1}{4}$. For the sake of contradiction, assume that $T_b = T_b(\varepsilon) \notin \mathcal{O}(\varepsilon \cdot \overline{\mathrm{OPT}}(I, c_l))$. Hence, there is for every $C > 0$ and every $\varepsilon > 0$ a value $\varepsilon' = \varepsilon'(\varepsilon, C) \leq \varepsilon$ such that

$$T_b(\varepsilon') > C \cdot \varepsilon' \cdot \overline{\mathrm{OPT}}\,(I, c_l) \quad .$$

On the other hand, we have

$$\begin{aligned}
(1-\lambda)\,\varepsilon' \cdot \overline{\mathrm{OPT}}\,(I, c_l) &= \left(1 - \frac{K_b}{T_b}\right)^{2\kappa_{\max}^{(b)}+4} T_b = \left(1 - \frac{K_b(\varepsilon')}{T_b(\varepsilon')}\right)^{2\kappa_{\max}^{(b)}+4} T_b \\
&\overset{(5.20)}{\geq} \left(1 - \lambda\varepsilon'\right) T_b \\
&> \left(1 - \lambda\varepsilon'\right) C \cdot \varepsilon' \cdot \overline{\mathrm{OPT}}\,(I, c_l) \quad .
\end{aligned}$$

Note that λ may depend on ε', which however does not influence the reasoning. Hence, $1 - \lambda \geq (1 - \lambda\varepsilon')C = C - \lambda\varepsilon' C$ holds. We get

$$1 \geq 1 - \lambda \geq C - \lambda\varepsilon' C \geq C - \varepsilon' C \geq C - \frac{1}{4}C = \frac{3}{4}C \quad .$$

This obviously is not satisfied for $C > \frac{4}{3}$. We get a contradiction, therefore $T_b \in \mathcal{O}(\varepsilon \cdot \overline{\mathrm{OPT}}(I, c_l))$. □

Finally, we can now prove Corollary 5.2. The idea is quite simple: we just preprocess and reduce the set of knapsacks C. This preprocessing is actually close to the one used for VBP in [78], which we also use in Subsection 2.6.2. The proof is therefore similar.

Proof of Corollary 5.2. Since we want to reduce the set C, we introduce (with a slight abuse of notation) the value $\mathrm{OPT}(I, \bar{C})$: it is the optimum for the UKPIP instance with items I and knapsack sizes $\bar{C}$, i.e. $\mathrm{OPT}(I, \bar{C}) = \max_{c \in \bar{C}} \mathrm{OPT}_c(I)$. Until now, we wrote $\mathrm{OPT}(I)$ because the set of knapsacks C did not change.

First, we set $\varepsilon' := \frac{\varepsilon}{3}$, and let $C' \subseteq C$ be the knapsacks with sizes of at least $c \geq \varepsilon'$, i.e. $C' = \{c \in C \mid c \geq \varepsilon'\}$. Then, we partition C' into intervals $((1+\varepsilon')^{-(l+1)}, (1+\varepsilon')^{-l}]$ for $l \in \{0, \ldots, \lfloor \log_{1+\varepsilon'}(\frac{1}{\varepsilon'}) \rfloor\}$. Let $C'' \subseteq C'$ be the set that has only the largest knapsack size in every interval, i.e.

$$C'' := \bigcup_l \left\{ \max \left\{ c \mid c \in \left((1+\varepsilon')^{-(l+1)}, (1+\varepsilon')^{-l} \right] \right\} \right\} \quad .$$

As shown in the proof of Lemma 2.22, we have $\mathcal{O}(\frac{1}{\varepsilon'} \log \frac{1}{\varepsilon'}) = \mathcal{O}(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ knapsack sizes.

The time to construct C' and then C'' is therefore in $\mathcal{O}(M + \frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ because we suppose that the logarithm can be determined in $\mathcal{O}(1)$. This is also a bound on the space complexity. (We can alternatively directly set $C'' = C'$ if C' has less than $\lfloor \log_{1+\varepsilon'}\left(\frac{1}{\varepsilon'}\right)\rfloor + 1$ knapsack sizes.)

Let c_0 be the optimum knapsack size for $\mathrm{OPT}(I, C')$ with $\mathrm{OPT}_{c_0}(I) = \mathrm{OPT}(I, C')$. If $c_0 \in C''$, the identity $\mathrm{OPT}(I, C'') = \mathrm{OPT}(I, C')$ holds. Otherwise, there is another knapsack size $c_1 \in C'$ in the same interval $((1+\varepsilon')^{-(l+1)}, (1+\varepsilon')^{-l}]$ as c_0 and where c_1 is in C'' instead of c_0. We have $c_1 \geq c_0 \geq \frac{1}{1+\varepsilon'}c_1$, i.e. $\frac{1}{c_1} \leq \frac{1}{c_0} \leq \frac{1+\varepsilon'}{c_1}$, which allows us to prove the following lower bound:

$$\begin{aligned}
\mathrm{OPT}\left(I, C''\right) &\geq \mathrm{OPT}_{c_1}(I) \overset{\text{Lem. 3.5}}{=} \frac{1}{c_1} \cdot \overline{\mathrm{OPT}}\left(I, c_1\right) \geq \frac{1}{c_1} \cdot \overline{\mathrm{OPT}}\left(I, c_0\right) \\
&\geq \frac{1}{1+\varepsilon'}\frac{1}{c_0} \cdot \overline{\mathrm{OPT}}\left(I, c_0\right) \geq \left(1-\varepsilon'\right)\frac{1}{c_0} \cdot \overline{\mathrm{OPT}}\left(I, c_0\right) \\
&= \left(1-\varepsilon'\right)\mathrm{OPT}_{c_0}(I) = \left(1-\varepsilon'\right)\mathrm{OPT}\left(I, C'\right) \quad . \qquad (5.21)
\end{aligned}$$

We have used the inequality $\frac{1}{1+\varepsilon'} \geq 1-\varepsilon'$.

Let c_2 be the optimum knapsack size for C such that $\mathrm{OPT}_{c_2}(I) = \mathrm{OPT}(I, C)$. If $c_2 \geq \varepsilon'$, then $c_2 \in C'$, and the identity $\mathrm{OPT}(I, C) = \mathrm{OPT}(I, C')$ follows. Otherwise, we have $c_2 < \varepsilon'$. The knapsack $1 = c_M$ is part of C, C' and C''. Take $\lfloor \frac{c_M}{c_2} \rfloor = \lfloor \frac{1}{c_2} \rfloor$ copies of the optimum solution of value $\mathrm{OPT}_{c_2}(I)$. The copies fit into the knapsack c_M, which yields a solution of profit $\lfloor \frac{1}{c_2} \rfloor \cdot \overline{\mathrm{OPT}}(I, c_2)$. (The items have in knapsack c_M only their basic profit p_j such that one copy of the solution has only profit $\overline{\mathrm{OPT}}(I, c_2)$; see also Lemma 3.5.) We get

$$\begin{aligned}
\mathrm{OPT}\left(I, C'\right) \geq \mathrm{OPT}_{c_M}(I) \geq \left\lfloor \frac{1}{c_2} \right\rfloor \cdot \overline{\mathrm{OPT}}\left(I, c_2\right) &\overset{\text{Lem. 3.5}}{\geq} \left(\frac{1}{c_2} - 1\right) \cdot c_2 \cdot \mathrm{OPT}_{c_2}(I) \\
= \left(1-c_2\right)\mathrm{OPT}_{c_2}(I) &\overset{c_2 < \varepsilon'}{\geq} \left(1-\varepsilon'\right)\mathrm{OPT}_{c_2}(I) = \left(1-\varepsilon'\right)\mathrm{OPT}\left(I, C\right) \quad . \qquad (5.22)
\end{aligned}$$

Thus, the new FPTAS works as follows: we first reduce C to C'', for which the running time and space complexity have been stated above. We now call our FPTAS from Theorem 5.1 with the knapsack set C'' and accuracy ε'. A solution is returned of value

$$\begin{aligned}
A_{\varepsilon'}(I, C'') &\geq \left(1-\varepsilon'\right)\mathrm{OPT}(I, C'') \overset{(5.21)}{\geq} \left(1-\varepsilon'\right)^2\mathrm{OPT}(I, C') \overset{(5.22)}{\geq} \left(1-\varepsilon'\right)^3\mathrm{OPT}(I, C) \\
&\geq \left(1-3\varepsilon'\right)\mathrm{OPT}(I, C) = \left(1-\varepsilon\right)\mathrm{OPT}(I, C) \quad .
\end{aligned}$$

The original FPTAS of Theorem 5.1 is called with $|C''| \in \mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ knapsack sizes and a minimum knapsack size of at least $\varepsilon' \in \Theta(\varepsilon)$, from which the time and space complexity for the corollary follow. □

5.9 Final Observations

We conclude this chapter with three remarks. First, we again compare the algorithm in this paper with the algorithm in Chapter 4.

Remark 5.45. The normal UKP is the UKPIP with $M = 1$ and $c_1 = c_{\min} = c = 1$. The algorithm presented in this chapter has for UKP a running time in $\mathcal{O}(\frac{1}{\varepsilon^2}\log^3(\frac{1}{\varepsilon}) + n)$ and a space complexity in $\mathcal{O}(\frac{1}{\varepsilon}\log^2(\frac{1}{\varepsilon}) + n)$ as seen in Theorem 5.1. Moreover, we have $\bar{P}_{c_{\min}^{(b)}} = P_0$ and only one knapsack set $C_b = C_0 = \{c\}$ such that $T = T_b = \frac{1}{2}\varepsilon P_0$ and $K = K_b = \frac{1}{4}\frac{1}{\log_2 \frac{1}{\varepsilon}+2}\varepsilon T$. The bounds on the running time, the space complexity, and T in this chapter are equal to the ones of the original UKP algorithm in Chapter 4 (see Theorem 4.1 and (4.1)). The constant K in this chapter is asymptotically equal to the constant $K = \frac{1}{4}\frac{1}{\log_2 \frac{1}{\varepsilon}+1}\varepsilon T$ defined in (4.2).

In general, the calculations in this chapter for T_b and K_b can be also done for UKP in Chapter 4 and indeed yield the values of T and K stated there (and therefore the same asymptotic running time and space complexity). The difference between K in Chapter 4 and K_b in this chapter is caused by the fact that we also have to consider $\tilde{I}^{(\kappa_{\max}^{(b)}+1,b)}$ (see Remark 5.18) while the corresponding set $\tilde{I}^{(\kappa+1)}$ is empty for UKP: two special cases are checked instead (see the explanation at the beginning of Section 4.7). Because of the additional set $\tilde{I}^{(\kappa_{\max}^{(b)}+1,b)}$, we get in Theorem 5.29 the lower bound of $(1-\frac{K_b}{T_b})^{\kappa_{\max}^{(b)}+2}\overline{\mathrm{OPT}}(I,c_l) - (1-\frac{K_b}{T_b})^{\kappa_{\max}^{(b)}+2}T_b$ and then the lower bound of $(1-\frac{K_b}{T_b})^{2\kappa_{\max}^{(b)}+4}\overline{\mathrm{OPT}}(I,c_l) - (1-\frac{K_b}{T_b})^{2\kappa_{\max}^{(b)}+4}T_b$ on the approximation (Lemma 5.38) from which K_b is derived. In Chapter 4, we instead get—because of $\tilde{I}^{(\kappa+1)} = \emptyset$—the lower bound of $(1-\frac{K}{T})^{2\kappa+2}\mathrm{OPT}(I) - (1-\frac{K}{T})^{2\kappa+2}T$ on the approximate solution, and therefore the slightly larger value for K.

To sum up, we have confirmed Remark 5.37.

We also want discuss the encoding of the output.

Remark 5.46. For the input $I = \{a_1,\ldots,a_n\}$, a natural representation of a solution is the multi-set $\{x_1 : a_1,\ldots,x_n : a_n\}$. The ungluing of the large items returns the individual item copies. They can be converted into a multi-set in $\mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon} + n)$ because there are at most $\mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ large items as seen in the proof of Lemma 5.43. The small items are added in $\mathcal{O}(1)$ to the multi-set by determining their number $\lfloor\frac{c_{\mathrm{sol}}-\bar{s}}{s(a_{\mathrm{eff}}^{(l)})}\rfloor$.

As a final remark, we bound the number of the large items in a solution.

Remark 5.47. Note that there are in fact $\mathcal{O}(\frac{1}{\varepsilon})$ large items. Let $c_l \in C_b$ be the knapsack size chosen by the algorithm. We have $P_{c_l}, P_{c_{\min}^{(b)}} \leq \mathrm{OPT}(I) \leq 2P_0$, $\frac{1}{2}P_0 \leq \bar{P}_{c_{\min}^{(b)}}/c_{\min}^{(b)}$

and $\frac{1}{2}P_0 \leq \bar{P}_{c_l}/c_l$ by Theorem 5.4 and 5.7 as well as Corollary 5.5. Since large items have a basic profit of at least $T_b = \frac{1}{2}\varepsilon \bar{P}_{c_{\min}^{(b)}}$, i.e. a profit of at least $\frac{T_b}{c_{l'}}$ in knapsack $c_{l'} \in C_b$, we have

$$\mathcal{O}\left(\frac{\mathrm{OPT}(I)}{T_b/c_l}\right) = \mathcal{O}\left(\frac{\bar{P}_{c_l}/c_l}{1/2\,\varepsilon\,\bar{P}_{c_{\min}^{(b)}}/c_l}\right) \overset{c_{\min}^{(b)}, c_l \in C_b}{=} \mathcal{O}\left(\frac{\bar{P}_{c_l}/c_l}{1/2\,\varepsilon\,\bar{P}_{c_{\min}^{(b)}}/c_{\min}^{(b)}}\right) = \mathcal{O}\left(\frac{P_0}{1/4\,\varepsilon P_0}\right)$$
$$= \mathcal{O}\left(\frac{1}{\varepsilon}\right)$$

large items.

6 Scheduling on Unrelated Machines of Few Different Types

6.1 Introduction

The result in this chapter was discovered by Jan Clemens Gehrke, Klaus Jansen, Stefan Kraft, and Jakob Schikowski during the lecture "Effiziente Algorithmen" in the summer term 2014. This chapter is based on the technical report [26] created by Stefan Kraft, which is an extension and a revision of the report [28] written by Jan Clemens Gehrke and Jakob Schikowski in German. Hence, parts of this chapter have been (almost) directly translated from the German report.

6.1.1 Known Results

Even Scheduling on Identical Machines ($P\,||\,C_{\max}$) (where, as the name suggests, all machines are of the same type) is NP-complete [25]. Thus, finding the optimum objective value $\mathrm{OPT}(I)$ and a corresponding schedule efficiently seems unlikely for the general case. We are therefore looking for efficient approximation algorithms. List Scheduling is a well-known heuristic with the approximation ratio $2 - \frac{1}{m}$ for $P\,||\,C_{\max}$. Hochbaum and Shmoys [36] presented the first Polynomial Time Approximation Scheme (PTAS).

Unfortunately, Scheduling on Unrelated Machines ($R\,||\,C_{\max}$) does not allow for a PTAS unless P = NP: a polynomial algorithm cannot in general have an approximation ratio $c < \frac{3}{2}$ as shown by Lenstra, Shmoys, and Tardos [64]. Approximation algorithms with a ratio of 2 were presented by Lenstra et al. [64], by Shmoys and Tardos [81], and by Gairing, Monien, and Woclaw [23]. A $2 - \frac{1}{m}$ approximation algorithm was found by Shchepin and Vakhania [79]. These algorithms are based on solving a linear program (LP) and rounding the solution to an integer one, with the exception of the purely combinatorial algorithm in [23]. Recently, Arad, Mordechai, and Shachnai [2] have presented a new algorithm that decides that a schedule σ with a makespan of at most T and an average machine load $L = \frac{\sum_{i \in \mathcal{M}} \sum_{j:\sigma(j)=i} p_{ij}}{m}$ does not exist, or it finds one with a makespan of at most $\min\{T + \frac{L}{h}, 2T\}$, where $h = h(T)$ is the so-called feasibility factor.

No algorithm is known for the general problem with a ratio better than 2. For a long time, this was even true for the Restricted Assignment Problem, a special case where $p_{ij} \in \{p_j, \infty\}$. A breakthrough was the estimation algorithm by Svensson [85]. The algorithm does not return an actual solution, but it can estimate the optimal makespan within $\frac{33}{17} + \varepsilon \approx 1.9412 + \varepsilon$, i.e. with a ratio better than 2. Chakrabarty, Khanna, and Li [10] have presented for a constant $\delta^* > 0$ a $(2 - \delta^*)$-approximation algorithm (that also returns a solution) for the $(1, \bar{\varepsilon})$-Restricted Assignment Problem. In this case of Restricted Assignment, the finite processing times are additionally either $p_j = 1$ or $p_j = \bar{\varepsilon}$ for constant $\bar{\varepsilon} > 0$.

Bhaskara et al. [6] studied the matrix $P = (p_{ij})_{m \times n}$ of the processing times, more precisely the influence of its rank on the non-approximability of $\mathrm{R}\,||\,C_{\max}$. Rank 1 is the case of identical or uniform machines (where the processing times are of the form $p_{ij} = \frac{p_j}{s_i}$), which allows for PTAS (see above for identical and e.g. [37, 51] for uniform machines). Unless P = NP, rank 4 is already APX-hard (i.e. a PTAS cannot exist), and rank 7 cannot be better approximated than $\frac{3}{2}$, as in the general case (see above). This was improved by Chen, Ye, and Zhang [11] who showed that already rank 4 does not allow for a polynomial-time approximation algorithm better than $\frac{3}{2}$ unless P = NP.

If the number m of machines is constant (i.e. $\mathrm{R}m\,||\,C_{\max}$ is considered), the problem has a PTAS [64] and a Fully Polynomial Time Approximation Scheme (FPTAS) [38]. Faster FPTAS were successively found [21, 50], and the fastest known FPTAS has a running time in $\mathcal{O}(n) + (\frac{m}{\varepsilon})^{\mathcal{O}(m)} \leq \mathcal{O}(n) + (\frac{\log m}{\varepsilon})^{\mathcal{O}(m \log m)}$ [49]. It should be noted that the algorithm by Lenstra, Shmoys, and Tardos [64], while "only" being a PTAS, has a space complexity only polynomial in m, $\log \frac{1}{\varepsilon}$, and the input length.

Interestingly, the special case of Scheduling on a constant number of m identical machines ($\mathrm{P}m\,||\,C_{\max}$) has a lower bound of $n^{\mathcal{O}(1)} + (\frac{1}{\varepsilon})^{\mathcal{O}(m)}$ on the running time unless the Exponential Time Hypothesis fails [11]. For ε small enough, e.g. $\varepsilon \leq \frac{1}{m}$, the running time of the algorithm in [49] can be bounded by $\mathcal{O}(n) + (\frac{1}{\varepsilon})^{\mathcal{O}(m)}$ and therefore attains this lower bound.

Finally, Imreh [40] considered the Scheduling Problem on $K = 2$ types. He presented heuristic algorithms with ratios $2 + \frac{m-1}{k}$ and $4 - \frac{2}{m}$, where m is the number of processors of the first and k the number of processors of the second type. Bleuse et al. [7] described an algorithm with the approximation ratio $\frac{4}{3} + \frac{1}{3k} + \varepsilon$ for scheduling on m cores (CPUs) and k GPUs. If all jobs are accelerated when executed on a GPU, the algorithm has the ratio $\frac{3}{2} + \varepsilon$. Wiese, Bonifaci, and Baruah [87] presented a PTAS for Scheduling on Unrelated Machines of Few Different Types ($(\mathrm{P}m_1, \ldots, \mathrm{P}m_K)\,||\,C_{\max}$) (where $K = \mathcal{O}(1)$). It has to solve $m^{\mathcal{O}(K \cdot ((1/\varepsilon)^{1/\varepsilon \log 1/\varepsilon}))}$ linear programs, which is therefore a lower bound on the overall running time. It is double exponential in $\frac{1}{\varepsilon}$. (An earlier paper [8] with a more sophisticated rounding of the relaxed ILP solution pre-

sented an algorithm for Δ-dimensional jobs.) Baruah [3] and Raravi and Nélis [74] presented for $K = 2$ PTAS that are single exponential in $\frac{1}{\varepsilon}$.

6.1.2 Our Result

We present a PTAS for the one-dimensional case that is only single exponential in $\frac{1}{\varepsilon}$ for general $K = \mathcal{O}(1)$.

Theorem 6.1. *There is a PTAS for* $(\mathrm{P}m_1, \ldots, \mathrm{P}m_K)||\,C_{\max}$ *with a running time in*

$$\mathcal{O}(K \cdot n) + m^{\mathcal{O}(K/\varepsilon^2)} \cdot \left(\frac{\log m}{\varepsilon}\right)^{\mathcal{O}(K^2)} .$$

6.2 Overview

Since the processing times are identical on machines of the same type, they are denoted by p_{kj} for $k \in \{1, \ldots, K\}$ and $j \in \{1, \ldots, n\}$. Section 6.4 explains the preprocessing of the instance I to get a new instance I^{merge} whose set of jobs $\mathcal{J}(I^{\text{merge}}) = \mathcal{J}^{\text{merge}}$ has a bounded cardinality. The method used was presented in [21, 49] and works as follows: first, the smallest processing time d_j over all machine types is determined for every job j. The jobs are scaled such that $1 \leq \mathrm{OPT}(I) \leq m$ holds, then divided into fast and slow for every machine type k and rounded accordingly based on a value $\varepsilon' \in \Theta(\varepsilon)$. We have a new instance I^{round} with jobs $\mathcal{J}^{\text{round}}$. Every job with its processing times p_{kj}^{round} has a profile $(\Pi_{1,j}, \ldots, \Pi_{k,j})$. Jobs with the same profile are then iteratively combined, i.e. merged, until we get the instance I^{merge}. As its job set $\mathcal{J}^{\text{merge}}$ has a bounded cardinality, this improves the running time of the next step.

Section 6.5 shows how to find for $\delta \in \Theta(\varepsilon)$ a solution with a value of at most $(1+\delta)\mathrm{OPT}(I^{\text{merge}})$ (and therefore an approximate solution to I). It uses the well-known dual approximation approach [36, 37, 64] a binary search with an oracle `Oracle`: in each iteration, a value T is tested. If there is a schedule with a makespan of at most T, the oracle returns for $\delta' \in \Theta(\varepsilon)$ a solution of value at most $(1 + C\delta')T$ (and T is decreased in the next iteration). If there is not a schedule, the oracle does not return any solution (and T is increased because it was too small). This can be iterated until a solution close enough to the optimum is found. The principle is explained in Subsection 6.5.1.

Subsection 6.5.2 gives an overview of `Oracle` and explains its preprocessing: I^{merge} (called I in Section 6.5, a slight abuse of notation) is first scaled based on T to get I^{scale}. Next, the oracle partitions the jobs into large and small jobs for every machine type k. The processing times of the jobs are then rounded so that they have discrete values, and we get the instance I^{r} with job processing times p_{kj}^{r}. Every feasible schedule σ

for I^{r} has a profile. As the job processing times are discrete, so are the profiles. One profile states for every machine type k the total processing time $as_k(\sigma)$ of the small jobs assigned to k. Moreover, it states for machine type k that $ab_k(\sigma, \gamma)$ machines have the processing time $\gamma \cdot (\delta')^2$, and it does so for all relevant $\gamma \in \mathbb{N}$.

The dynamic program `DynProg` of the oracle is presented in Subsection 6.5.3: it is the main contribution in this chapter. `DynProg` iteratively constructs all possible profiles $TS_0, \ldots, TS_{n'}$, where TS_j contains all profiles for the first jobs $\{1, \ldots, j\}$. Each constructed profile t is represented like above: it has the entries $AS_k = AS_k(t)$ that correspond to the $as_k(\cdot)$ and the entries $(AB_k(t))_\gamma$ that correspond to the $ab_k(\cdot, \gamma)$. The number of different profiles is denoted by κ.

Subsection 6.5.4 first introduces the instance I^{p} with its processing times p^{p}_{kj}, which is similar to I^{r}. Then, the function `CreateSchedule` is presented. It first checks a simple condition for every constructed profile $t' \in TS_{n'}$ to see whether the small jobs can be greedily assigned to the machines. If yes, it calls the function `Backtracking` that constructs the schedules σ_k for the large jobs and the set $\mathcal{J}_k$ of small jobs assigned to machine type k. This is done by backtracking (as suggested by the function name). Afterwards, `CreateSchedule` greedily assigns the jobs in $\mathcal{J}_k$ to the machines, which yields a solution with a makespan of at most $(1 + \Theta(\varepsilon))T$. If no profile has been generated by the dynamic program or no profile allows for a distribution of the small jobs, the value T is too small. The binary search adapts T according to the output of the oracle until the optimum has been approximated.

Finally, Section 6.6 puts the preprocessing of Section 6.4 and the dual approach of Section 6.5 together and shows that the overall algorithm is indeed a PTAS.

Note that the principle of our algorithm is similar to [3, 21, 74], but it was found independently of [3, 74]. We think that the algorithm presented in this chapter can be considered to be less complicated and to have an easier analysis than the algorithm in [74] and possibly in [3]. Moreover, we use a different dynamic program.

6.3 General Remarks and Notation

Since we are in the case of K machine types, it is sufficient to state for a job j its processing time on every machine type and not on every individual machine. The processing time of job j on the machine type $k \in \{1, \ldots, K\}$ is therefore denoted by p_{kj}. The value $k(i)$ is the type of a machine $i \in \mathcal{M}$.

We suppose that K is constant, that $0 < \varepsilon \leq \frac{1}{2}$ and (still) that computing the logarithm needs time in $\mathcal{O}(1)$.

We can assume without loss of generality that $m_k \leq n$ for all $k \in \{1, \ldots, K\}$ and therefore $m = m_1 + \cdots + m_K \leq n \cdot K$. In fact, a solution cannot use more than n

machines of a type k because there are only n jobs. For one type, machines whose number exceeds n can therefore be discarded.

6.4 Preprocessing of the Instance

The first step of the algorithm is a preprocessing to reduce the number of items. The technique in this section is taken from [21, 49]. Let $0 < \varepsilon' \leq \frac{1}{3}$ with $\varepsilon' \in \Theta(\varepsilon)$. The actual value of ε' will be determined later.

First, let

$$d_j := \min_{k \in \{1,\dots,K\}} p_{kj}$$

be the smallest processing time of a job j over all machine types, and let $D := \sum_{j \in \mathcal{J}} d_j$. We have $\frac{D}{m} \leq \mathrm{OPT}(I)$ because the jobs could ideally be scheduled uniformly on all machines, where each job is executed on one machine of its fastest type. On the other hand, we have $\mathrm{OPT}(I) \leq D$: a feasible solution is obtained by scheduling each job to one of its fastest machine. In the worst case, all jobs have the same fastest machine type, and there is only one machine of this type.

Hence, we can divide all processing times p_{kj} by $\frac{D}{m}$ such that we get the following:

Assumption 6.1. Without loss of generality, the jobs are scaled such that we have $1 \leq \mathrm{OPT}(I) \leq m$ and $D = \sum_{j \in \mathcal{J}} d_j = m$.

The jobs are now partitioned into fast and slow ones for each type k. A job is slow on type k if $p_{kj} \geq \frac{m}{\varepsilon'} d_j$, otherwise it is fast on type k. Should j be slow on type k, we set $p_{kj}^{\text{round}} := \infty$ (or to a sufficiently large value like $2m$) such that a reasonable algorithm will not schedule j on such a machine. If j is fast on type k, we round it down to

$$\text{the nearest lower value } p_{kj}^{\text{round}} := d_j \left(1 + \varepsilon'\right)^h \text{ for } h \in \mathbb{N} \ .$$

We therefore have $d_j(1+\varepsilon')^h \leq p_{kj} < d_j(1+\varepsilon')^{h+1}$ and $h = \lfloor \log_{1+\varepsilon'} \frac{p_{kj}}{d_j} \rfloor$. The new instance of scaled and rounded jobs $\mathcal{J}^{\text{round}}$ together with the (unchanged) machines is called I^{round}.

Remark 6.2. In this chapter, we will sometimes directly refer to "fast jobs" and "slow jobs" although we mean e.g. "jobs scheduled on machines where they are fast". We may also call jobs "fast" or "slow" when we refer to their processing times, which should therefore be "the processing times of one job on machines where the job is fast" and "the processing times of one job on machines where it is slow." Similarly, we may also use expressions like "jobs on slow/fast machines" when we mean "jobs scheduled on machines where they are slow/fast."

Finally, jobs will later on also be called "large" and "small" such that similar expressions will be used.

Lemma 6.3 ([49, Lemma 2.1]). *We have* $\mathrm{OPT}(I^{\mathrm{round}}) \leq (1+\varepsilon')\,\mathrm{OPT}(I)$.

Proof. Let I' be the instance where the fast jobs are rounded, but the processing times of slow jobs have not been set to ∞. On the one hand, we obviously have $\mathrm{OPT}(I') \leq \mathrm{OPT}(I)$ because the processing times may only have decreased. On the other hand, we can take an optimum solution to I' and replace every rounded processing time p_{kj}^{round} by its original processing time p_{kj}. Then the schedule increases only by a factor of $1+\varepsilon'$. We get $\mathrm{OPT}(I') \leq \mathrm{OPT}(I) \leq (1+\varepsilon')\mathrm{OPT}(I')$.

Let $\sigma' : \mathcal{J} \to \mathcal{M}$ be an optimum schedule for I', i.e. with a makespan of $\mathrm{OPT}(I')$. We transform it into a schedule σ'' for I^{round}. If a job j is scheduled by σ' to a machine on which it is slow, it is moved to an arbitrary machine where it is processed in time d_j (i.e. to one of the fastest machines for the job). Let S be the set of jobs that have been scheduled by σ' on slow machines. In the worst case, the processing time for one machine increases with the modified schedule σ'' by

$$\begin{aligned}\sum_{j\in S} d_j &\leq \sum_{j\in S} \frac{\varepsilon'}{m}\cdot p_{k(\sigma'(j))j} = \sum_{i\in\mathcal{M}} \sum_{j:\sigma'(j)=i} \frac{\varepsilon'}{m}\cdot p_{k(i)j} \leq \sum_{i\in\mathcal{M}} \frac{\varepsilon'}{m}\cdot \mathrm{OPT}(I') \\ &= \frac{\varepsilon'}{m}\cdot m\cdot \mathrm{OPT}(I') = \varepsilon'\mathrm{OPT}(I') \leq \varepsilon'\mathrm{OPT}(I)\ .\end{aligned}$$

The modified schedule is obviously a schedule for I^{round}. To sum up, we get

$$\begin{aligned}\mathrm{OPT}(I^{\mathrm{round}}) &\leq \mathrm{OPT}(I') + \varepsilon'\mathrm{OPT}(I') \leq \mathrm{OPT}(I) + \varepsilon'\mathrm{OPT}(I) \\ &= \left(1+\varepsilon'\right)\mathrm{OPT}(I)\ .\end{aligned}$$

□

The following lemma allows us to derive an approximation algorithm for I from an algorithm for I^{round}.

Lemma 6.4 ([49, Lemma 2.3]). *If there is an approximation algorithm A_r for I^{round} such that $A_r(I^{\mathrm{round}}) \leq \alpha\mathrm{OPT}(I) + \beta$, then there is also an approximation algorithm A for I with*

$$A(I) \leq \alpha\left(1+\varepsilon'\right)^2\mathrm{OPT}(I) + \beta(1+\varepsilon') \leq \left(\alpha(1+\varepsilon')^2 + \beta(1+\varepsilon')\right)\mathrm{OPT}(I)\ .$$

Proof (Sketch). The proof is constructive: find a schedule for I^{round} and replace the modified processing times by the unmodified ones. □

Now, jobs are grouped together in a special way to reduce their overall number. We first introduce the notion of profiles for jobs in I^{round}: a rounded job $j \in \mathcal{J}(I^{\mathrm{round}})$ has the profile $(\Pi_{1,j},\ldots,\Pi_{k,j})$ where $\Pi_{k,j} \in \mathbb{N}$ is the exponent such that $p_{kj}^{\mathrm{round}} = d_j\left(1+\varepsilon'\right)^{\Pi_{k,j}}$. We set $\Pi_{k,j} = \infty$ if $p_{kj}^{\mathrm{round}} = \infty$.

Lemma 6.5 ([21, Lemma 2]). *Let l be the number of profiles for the jobs in I^{round}. We have the bound $l \leq (2 + \log_{1+\varepsilon'}(\frac{m}{\varepsilon'}))^K$.*

Proof. If a job j is fast on the machine type k, its maximum processing time is $\frac{m}{\varepsilon'} d_j$. Hence, we have for the largest exponent $h_{\max}$ that $d_j(1+\varepsilon')^{h_{\max}} \leq \frac{m}{\varepsilon'} d_j$ such that $h_{\max} = \lfloor \log_{1+\varepsilon'}(\frac{m}{\varepsilon'}) \rfloor$ follows. The smallest exponent is $h_{\min} = 0$. If j is slow on type k, then its exponent is ∞. Thus, there are $2 + \lfloor \log_{1+\varepsilon'}(\frac{m}{\varepsilon'}) \rfloor$ possible exponents on K positions, which yields $(2 + \log_{1+\varepsilon'}(\frac{m}{\varepsilon'}))^K$ possible profiles. □

We derive from I^{round} a modified instance I^{merge}. Let $\nu := \frac{1}{\lceil m/\varepsilon' \rceil}$. The jobs in I^{round} are divided into the set of large jobs $L := \{j \in \mathcal{J}(I^{\text{round}}) \mid d_j > \nu\}$ and the set of small jobs $S := \{j \in \mathcal{J}(I^{\text{round}}) \mid d_j \leq \nu\}$. Take an enumeration of the l profiles such that we can denote a profile directly by its number $\varsigma \in \{1, \ldots, l\}$. The set S is then further partitioned into the sub-sets $S_\varsigma = \{j \in \mathcal{J}(I^{\text{round}}) \mid j \text{ has the profile } \varsigma\}$ for $\varsigma \in \{1, \ldots, l\}$.

Two jobs j_a and j_b with the same profile ς and for which $d_{j_a}, d_{j_b} \leq \frac{\nu}{2}$ holds are now grouped together to a new composed job j_c with $p_{kj_c} := p_{kj_a} + p_{kj_b}$ for every k. The composing is repeated until there is at most one job $j \in S_\varsigma$ with $d_j \leq \frac{\nu}{2}$ for every profile ς. The other jobs (including the jobs in L) now have all a processing time of at least $\frac{\nu}{2}$. The set of all jobs is called $\mathcal{J}^{\text{merge}}$, which yields together with the (unchanged) machines the instance I^{merge}.

Lemma 6.6. *If two jobs j_a and j_b are grouped together to j_c, then j_c has the same profile as j_a and j_b.*

Proof. We have $\Pi_{k,j_a} = \Pi_{k,j_b}$ for all $k \in \{1, \ldots, K\}$ because j_a and j_b have the same profile. Let k' be a machine type where $\Pi_{k',j_a} = \Pi_{k',j_b} = 0$, i.e. both jobs have their fastest processing time d_{j_a} and d_{j_b} on the machine type k'. We have

$$\begin{aligned} p^{\text{round}}_{kj_c} &= p^{\text{round}}_{kj_a} + p^{\text{round}}_{kj_b} = d_{j_a}\left(1+\varepsilon'\right)^{\Pi_{k,ja}} + d_{j_b}\left(1+\varepsilon'\right)^{\Pi_{k,jb}} \\ &= \left(d_{j_a} + d_{j_b}\right)\left(1+\varepsilon'\right)^{\Pi_{k,ja}} \quad \text{for all } k \in \{1, \ldots, K\} \quad . \end{aligned}$$

Thus, we have $p^{\text{round}}_{k'j_c} = d_{j_a} + d_{j_b}$, and $d_{j_c} = d_{j_a} + d_{j_b}$. The job j_c has therefore the same profile as j_a and j_b. □

Lemma 6.7. *After composing the items, we still have $1 \leq \text{OPT}(I^{\text{merge}}) \leq m$.*

Proof. Composed jobs have values d_{j_c} that are the sum of several d_j for $j \in \mathcal{J}(I^{\text{round}})$. Hence, we get $D = m = \sum_{j \in \mathcal{J}} d_j = \sum_{j \in \mathcal{J}^{\text{merge}}} d_j$ (see Assumption 6.1) because one $j \in \mathcal{J}(I^{\text{round}})$ can only be used in the composition of one job $j_c \in \mathcal{J}^{\text{merge}} = \mathcal{J}(I^{\text{merge}})$. The upper and lower bound now follow as before. □

Theorem 6.8. *There are* $\min\{n, \frac{2D}{\nu} + l\} = \min\{n, \mathcal{O}(\frac{m^2}{\varepsilon'}) + (\frac{\log m}{\varepsilon'})^{\mathcal{O}(K)}\}$ *jobs in* $\mathcal{J}^{merge}$ *(i.e. in* I^{merge}*).*

Proof. The number of jobs does obviously not increase so that n is an upper bound. The number of jobs in $\mathcal{J}^{\text{merge}}$ that have a shortest processing time of $d_j > \frac{\nu}{2}$ is bounded by $\frac{\sum_{j \in \mathcal{J}} d_j}{\nu/2} = \frac{2D}{\nu}$. Moreover, there is at most one job left with $d_j \leq \frac{\nu}{2}$ for every profile ς. Therefore, we have at most $l + \frac{2D}{\nu}$ jobs. Note that $0 < \varepsilon' \leq \frac{1}{3}$ such that $\ln(1+\varepsilon') \geq \varepsilon' - (\varepsilon')^2 = \varepsilon'(1-\varepsilon') \geq \frac{1}{2}\varepsilon'$ holds. We get

$$
\begin{array}{ccccc}
 & \frac{2D}{\nu} & + & l & \\
= & \underbrace{2m \cdot \left\lceil \frac{m}{\varepsilon'} \right\rceil} & + & \underbrace{\left(2 + \log_{1+\varepsilon'}\left(\frac{m}{\varepsilon'}\right)\right)^K} & \in \mathcal{O}\left(\frac{m^2}{\varepsilon'}\right) + \left(\frac{\log m}{\varepsilon'}\right)^{\mathcal{O}(K)} \quad . \\
\leq & 2m \cdot \left(\frac{m}{\varepsilon'} + 1\right) & = & \left(2 + \frac{\ln(\frac{m}{\varepsilon'})}{\ln(1+\varepsilon')}\right)^K & \\
\leq & \frac{2m^2}{\varepsilon'} + \frac{m^2}{\varepsilon'} & \leq & \left(2 + \frac{2}{\varepsilon'} \cdot \ln\left(\frac{m}{\varepsilon'}\right)\right)^K & \\
\in & \mathcal{O}\left(\frac{m^2}{\varepsilon'}\right) & \in & \left(\frac{\log m}{\varepsilon'}\right)^{\mathcal{O}(K)} &
\end{array}
$$

The proof is an extension of the proof in [21, 49]. □

Theorem 6.9 ([49, Lemma 2.4]). *We have*

$$\text{OPT}(I^{\text{round}}) \leq \text{OPT}(I^{\text{merge}}) \leq \text{OPT}(I^{\text{round}}) + \varepsilon' \quad .$$

We state the running time to construct I^{merge}.

Theorem 6.10. I^{merge} *can be constructed from* I *in time* $\mathcal{O}(n \cdot K)$*.*

Proof. We have the following parts:

- Finding the values d_j for all jobs j, the calculation of D as well as the scaling of the instance can be done in $\mathcal{O}(n \cdot K)$.
- We suppose that determining the logarithm can be done in $\mathcal{O}(1)$ such that the exponent h for which $d_j(1+\varepsilon')^h \leq p_{kj} < d_j(1+\varepsilon')^{h+1}$ holds can be found in $\mathcal{O}(1)$. In fact, we have $h = \lfloor \log_{1+\varepsilon'} \frac{p_{kj}}{d_j} \rfloor$. Hence, the jobs can be rounded and their profiles determined in $\mathcal{O}(n \cdot K)$. A rounded job is then directly added to the stack that corresponds to its profile. Thus, there are at most $\mathcal{O}(n)$ profiles to be considered.
- The partitioning of the jobs into L and S is done in $\mathcal{O}(n)$.

- The composition of the jobs is described in Algorithm 6.1. First, list(j) is created for every $j \in S$, a list that is used at the end of the algorithm to replace a composed item by the the ones it consists of. The jobs in S are already grouped into their respective profile sets S_ς (see above). Then, the jobs are further partitioned into $S_\varsigma^1 := \{j \in S_\varsigma \mid d_j > \frac{\nu}{2}\}$ and $S_\varsigma^2 := \{j \in S_\varsigma \mid d_j \leq \frac{\nu}{2}\}$. The jobs are iteratively combined into larger ones in each S_ς^2 until at most one job is left in each S_ς^2, i.e. there is at most one job in S_ς with $d_j \leq \frac{\nu}{2}$. The correctness of the algorithm is obvious because each S_ς^2 will always contain all small jobs of profile ς with $d_j \leq \frac{\nu}{2}$.

  As for the running time, the division of the small jobs into the sets S_ς^1 and S_ς^2 needs time in $\mathcal{O}(n)$ over all profiles ς. Fix one set S_ς^2. The combination of two jobs in S_ς^2 can be done in $\mathcal{O}(K)$. For combining list(j_a) and list(j_b), suppose that linked lists are used for list$(\cdot)$. Since j_a and j_b are not used afterwards, we just concatenate list(j_a) and list(j_b) and save the result as list(j_c). With the right implementation of linked lists, this needs only $\mathcal{O}(1)$. Finally, checking whether j_c satisfies $d_{j_c} > \frac{\nu}{2}$ (and adapting S_ς^1 and S_ς^2) can be done in $\mathcal{O}(1)$. Hence, one iteration of the while-loop needs time in $\mathcal{O}(K)$. The while-loop can only be executed at most n times in total over all S_ς because all jobs will then be merged into one. Hence, we get a running time in $\mathcal{O}(n \cdot K)$ for the entire algorithm.

Algorithm 6.1: This procedure combines the items in S into larger ones until at most one item with $d_j \leq \frac{\nu}{2}$ is contained in every S_ς.

for $j \in S$ **do**
  list$(j) := \{j\}$; `// Information for undoing the composition of every item`
for *every* S_ς **do**
  Partition S_ς into $S_\varsigma^1 = \{j \in S_\varsigma \mid d_j > \frac{\nu}{2}\}$ and $S_\varsigma^2 = \{j \in S_\varsigma \mid d_j \leq \frac{\nu}{2}\}$;
  while *There are (at least) two jobs* $j_a, j_b \in S_\varsigma^2$ **do**
    Group j_a and j_b together to j_c;
    list$(j_c) :=$ list$(j_a) \cup$ list(j_b);
    if $d_{j_c} > \frac{\nu}{2}$ **then**
      $S_\varsigma^2 := S_\varsigma^2 \setminus \{j_c\}$ and $S_\varsigma^1 := S_\varsigma^1 \cup \{j_c\}$;

(This proof is close to the explanations in [21, 49].) □

6.5 The Main Algorithm

Let $0 < \delta \leq \varepsilon' \leq \frac{1}{3}$ with $\delta \in \Theta(\varepsilon')$. We present our algorithm for an instance I with n' items and $1 \leq \mathrm{OPT}(I) \leq m$: it finds a solution of value at most $(1+\delta)\mathrm{OPT}(I)$. Later on, I will in fact be I^{merge} such that we make the following assumption:

Assumption 6.2. I has $n' \in \mathcal{O}(\frac{m^2}{\varepsilon'}) + (\frac{\log m}{\varepsilon'})^{\mathcal{O}(K)} = \mathcal{O}(\frac{m^2}{\delta}) + (\frac{\log m}{\delta})^{\mathcal{O}(K)}$ items.

6.5.1 Approximating the Optimum by Binary Search

We introduce another value $\delta' \in \Theta(\delta)$ with $0 < \delta' \leq \delta \leq \varepsilon' \leq \frac{1}{3}$. Suppose that we have an oracle `Oracle(`I, T`)` that returns for a given makespan T and a constant $C > 0$ either a solution of value at most $(1 + C\delta')T$ or $\perp$ (false). The answer $\perp$ implies that there is not a solution of value (at most) T, i.e. $T < \mathrm{OPT}(I)$. We employ the well-known dual approximation approach [36, 37, 64], a binary search with such an oracle, to approximate the optimum $\mathrm{OPT}(I)$ up to a given approximation ratio (see Algorithm 6.2). We start with the lower bound $LB = 1$ and the upper bound $UB = m$ and first check whether $LB = 1$ yields a solution. If not, we iteratively adapt LB and UB until the difference $UB - LB$ is small enough. At the same time, we ensure that $LB < OPT(I)$ always holds and that there is a solution for the value $T = UB$: the solution for UB is returned at the end. Note that the oracle can construct a solution for $T = m$ in $\mathcal{O}(n' + m)$ as described at the beginning of Section 6.4.

Algorithm 6.2: Binary search to approximate $\mathrm{OPT}(I)$ with the oracle

```
LB := 1;
UB := m;
if Oracle(I,LB) ≠ ⊥ then // Solution for T = LB exists
    return Oracle(I,LB);
while UB − LB > δ' do
    T := (UB+LB)/2;
    result := Oracle(I, T);
    if result = ⊥ then // T is too small
        LB := T;
    else // T is large enough
        UB := T;
T := UB;
return Oracle(I,T);
```

Lemma 6.11. *Suppose that the* `Oracle` *function has the properties above, i.e. it returns for given T either a solution with a makespan of at most $(1 + C\delta')T$ or $\perp$. In the second case, we have $T < \mathrm{OPT}(I)$. Then, the dual approximation approach, i.e. the binary search, finds a schedule with a makespan of at most $\left(1 + (C+1)\delta' + C(\delta')^2\right)\mathrm{OPT}(I)$ and needs $\mathcal{O}(\log(\frac{m}{\delta'}))$ calls of the oracle.*

Proof. It is clear that the algorithm finishes after $\mathcal{O}(\log(\frac{m}{\delta'}))$ iterations of the binary search.

The correctness of the binary search has already been shown in [36, 37, 64] and is easy to see: first, we have at the beginning $1 = LB \leq \mathrm{OPT}(I)$ and that there is a solution for $UB = m$. If the oracle does not return a solution for $T = LB$, we have $LB < \mathrm{OPT}(I)$. The algorithm then makes sure by the adaptation of T that we always have $LB < \mathrm{OPT}(I)$ and that the oracle returns for $T = UB$ a solution (of value at most $(1 + C\delta')T$). Suppose that the binary search terminates, i.e. $UB - LB \leq \delta'$ holds. The oracle then returns a solution with a makespan of at most $(1 + C\delta')UB$. Since we have $UB \leq LB + \delta' \leq \mathrm{OPT}(I) + \delta'$, we get

$$\begin{aligned}
\left(1 + C\delta'\right) UB &\leq \left(1 + C\delta'\right)\left(\mathrm{OPT}(I) + \delta'\right) \\
&= \left(1 + C\delta'\right)\mathrm{OPT}(I) + \left(1 + C\delta'\right)\delta' \\
&\leq \left(1 + C\delta'\right)\mathrm{OPT}(I) + \left(1 + C\delta'\right)\delta'\mathrm{OPT}(I) \\
&= \left(1 + C\delta'\right)\left(1 + \delta'\right)\mathrm{OPT}(I) \\
&= \left(1 + (C+1)\delta' + C(\delta')^2\right)\mathrm{OPT}(I)\ . \qquad \square
\end{aligned}$$

6.5.2 The Oracle

We now describe the principle of the oracle. As a first step, the processing times of the jobs in I are divided by T. We get a new instance I^{scale} with $\mathrm{OPT}(I^{\mathrm{scale}}) \leq 1$ (if $\mathrm{OPT}(I) \leq T$). Then, the jobs are rounded to get the instance I^{r} with $\mathrm{OPT}(I^{\mathrm{r}}) \leq (1 + \delta')$. A dynamic program `DynProg` is used to iteratively construct the sets $TS_0, \ldots, TS_{n'}$ of *profiles*, where each profile represents several (real) schedules. (A formal definition is given below.) The profiles in TS_j consider the first j jobs $\{1, \ldots, j\}$. At the end, a function `CreateSchedule` tries to construct a discrete schedule σ for the instance I^{p} (which is similar to I^{r}) from each profile $t \in TS_{n'}$, where the profiles in $TS_{n'}$ consider all n' jobs. If there is a solution to I with a makespan of at most T, one discrete schedule σ for I with a makespan of at most $(1 + C\delta')T$ will be found by `CreateSchedule`. An overview is shown in Algorithm 6.3. We first have the following obvious lemma:

Lemma 6.12. *The set I^{scale} can be constructed in $\mathcal{O}(n' \cdot K)$.*

Algorithm 6.3: An overview of the `Oracle(`I, T`)`

```
Construct I^scale from I;
Round I^scale to I^r;
TS = (TS_0, ..., TS_{n'}) := DynProg(I^r);
for all t ∈ TS_{n'} do
    σ := CreateSchedule(t);
    if σ ≠ ⊥ then
        // A schedule of value at most (1 + Cδ') has been found for I^p
        return σ;
return ⊥ ; // No schedule for I with a makespan ≤ T
```

In a slight abuse of notation, we still denote the scaled processing times by p_{kj}.

Definition 6.13. *A (scaled) job j is large on a machine type k if $p_{kj} \geq \delta'$. Otherwise, it is small.*

Take one job j. If its processing time is large on a machine type k, it is rounded up to the next $\gamma \cdot (\delta')^2$ for $\gamma \in \mathbb{N}$. If the processing time is small, i.e. $p_{kj} < \delta'$, the processing time is rounded *down* to the next multiple of $\frac{m_k \cdot \delta'}{n'}$. This new instance with processing times p^{r}_{kj} is denoted by I^{r}.

Note that jobs are large (or small) on a machine type in I^{r} if they are large (or small) in I^{scale}, and vice versa.

Lemma 6.14. *If I^{scale} has a schedule with a makespan of at most 1, then I^r has a schedule with a makespan of at most $1 + \delta'$. I^r can be constructed in $\mathcal{O}(n' \cdot K)$.*

Proof. Let σ be a schedule for I^{scale} with a makespan $T \leq 1$. By definition, there can only be $\frac{1}{\delta'}$ large jobs on one machine. Replace all jobs by their rounded counterpart in I^{r}. The processing times of small jobs may only decrease while each large job increases by at most $(\delta')^2$. Hence, the increase of the total processing time on a machine is bounded by $(\delta')^2 \cdot \frac{1}{\delta'} = \delta'$. The new total processing time of machine i is therefore at most $T + \delta' \leq 1 + \delta'$. Since this holds for all machines, the bound on the makespan follows for I^{r}.

The running time to obtain I^{r} is obvious. □

Take one schedule for I^{r} with a makespan of at most $1 + \delta'$. If only large jobs of I^{r} are scheduled on a machine i, its total processing time is a multiple of $(\delta')^2$. In fact, it must be one of the values

$$\{0\} \cup \left\{\gamma \cdot (\delta')^2 \mid \gamma \in \mathbb{N} \text{ and } \delta' \leq \gamma \cdot (\delta')^2 \leq 1 + \delta'\right\} \quad .$$

These processing times can be numbered with $\gamma = 0$ (for the total processing time 0) and $\gamma \in \{\gamma_0 := \lceil \frac{1}{\delta'} \rceil, \gamma_0 + 1, \ldots, \gamma_1 - 1, \gamma_1 := \lfloor \frac{1+\delta'}{(\delta')^2} \rfloor\}$. If j is large, the value $\gamma(k,j)$ is the factor such that $p^{\mathrm{r}}_{kj} = \gamma(k,j) \cdot (\delta')^2$.

Lemma 6.15. *For I^r, there are $\mathcal{O}(\frac{1}{(\delta')^2})$ processing times of the form $\gamma \cdot (\delta')^2$ of large jobs on a machine.*

Similarly, take all small jobs assigned to a machine type k. Their total processing time is at most $m_k \cdot (1 + \delta')$ because we consider a schedule with a makespan of at most $1 + \delta'$. Moreover, the total processing time is also a multiple of $\frac{m_k \cdot \delta'}{n'}$ because of the rounding, i.e. it is one of the values in

$$\Sigma_k := \left\{ \tau \cdot \frac{m_k \cdot \delta'}{n'} \;\middle|\; \tau \in \mathbb{N} \text{ and } 0 \leq \tau \cdot \frac{m_k \cdot \delta'}{n'} \leq m_k \cdot (1 + \delta') \right\} \ .$$

Lemma 6.16. *For one machine type k of I^r, there are $\mathcal{O}(\frac{n'}{\delta'}\,(1+\delta')) = \mathcal{O}(\frac{n'}{\delta'})$ possible total processing times of small jobs in Σ_k.*

Proof. The small jobs have a total processing time in the interval $[0, (1 + \delta') \cdot m_k]$. Since the processing times are a multiple of $\frac{m_k \cdot \delta'}{n'}$, the lemma follows. □

Based on the observations above, we introduce several useful definitions.

Definition 6.17. *Let I' be a sub-instance of I^r, i.e. an instance whose jobs $\mathcal{J}'$ are a subset of the jobs in I^r, and which has the same machines as I^r. Let $\sigma : \mathcal{J}' \to \mathcal{M}$ be a feasible schedule with a makespan of at most $1 + \delta'$. Let b_i be the total processing time of the large jobs assigned to machine i, i.e.*

$$b_i = b_i(\sigma) := \sum_{j:\sigma(j)=i, p^{\mathrm{r}}_{k(i)j} \geq \delta'} p^{\mathrm{r}}_{k(i)j} \ .$$

We also introduce the remaining processing time (or remaining machine capacity) of every machine type k for the makespan $1 + \delta'$:

$$r_k = r_k(\sigma) := \sum_{i \in \mathcal{M}_k} (1 + \delta' - b_i) = m_k \cdot (1 + \delta') - \sum_{i \in \mathcal{M}_k} b_i \ .$$

Moreover, the value $ab_k(\sigma, \gamma)$ denotes the number of machines of type k where $b_i = \gamma \cdot (\delta')^2$:

$$ab_k(\sigma, \gamma) := \left| \left\{ i \in \mathcal{M}_k \mid b_i = \gamma \cdot (\delta')^2 \right\} \right| \text{ for } \gamma \in \{0, \gamma_0, \ldots, \gamma_1\} \ .$$

The vector $ab_k(\sigma) = (ab_k(\sigma, \gamma))_{\gamma=0,\gamma_0,\ldots,\gamma_1}$ contains all entries $ab_k(\sigma, \gamma)$.

Furthermore, $as_k(\sigma)$ is the total processing time of all small jobs assigned to machine type k, i.e.

$$as_k(\sigma) := \sum_{j:\sigma(j)\in\mathcal{M}_k, p^{\mathrm{r}}_{kj}<\delta'} p^{\mathrm{r}}_{kj} \ .$$

As seen above, we have $as_k(\sigma) \in \Sigma_k$. Since the small jobs have to fit into the remaining processing time, $as_k(\sigma) \leq r_k(\sigma)$ holds for all $k \in \{1,\dots,K\}$.

The values $ab_k(\sigma,\gamma)$ and $as_k(\sigma)$ form the profile *of σ:*

$$\Big(\Big[\Big(ab_1(\sigma,0), ab_1(\sigma,\gamma_0),\dots,ab_1(\sigma,\gamma_1)\Big), as_1(\sigma)\Big], \dots, \Big[\Big(ab_K(\sigma,0), ab_K(\sigma,\gamma_0),\dots,ab_K(\sigma,\gamma_1)\Big), as_K(\sigma)\Big]\Big) \ .$$

6.5.3 Dynamic Programming

The dynamic program we introduce determines all possible profiles for I^{r}. It is together with the definitions above the main contribution of this chapter. One profile t for a sub-instance I' is represented like above: it is a tuple of K tuples, one for each machine type:

$$t = ((AB_1, AS_1), \dots, (AB_k, AS_k), \dots, (AB_K, AS_K)) \ .$$

For each $k \in \{1,\dots,K\}$, the entry AS_k denotes the total processing time of all small jobs that are assigned to the machines of type k.

One AB_k is again a tuple

$$AB_k = (q_0, q_{\gamma_0}, \dots, q_\gamma, \dots, q_{\gamma_1}) \ .$$

Each entry q_γ denotes the number of machines of type k where the large jobs have the total processing time $\gamma \cdot (\delta')^2$. Obviously, $q_0 + \sum_{\gamma=\gamma_0}^{\gamma_1} q_\gamma = m_k$ holds.

For convenience, $AS_k(t)$ denotes the entry AS_k of a profile t. Similarly, $AB_k(t)$ stands for the tuple AB_k of profile t. Additionally, $(AB_k(t))_\gamma$ is the entry q_γ in the tuple $AB_k(t)$.

Lemma 6.18. *One profile t has $\mathcal{O}(\frac{K}{(\delta')^2})$ entries.*

Proof. There are K entries $AS_k(t)$ and K tuples $AB_k(t)$. Each $AB_k(t)$ has again $\mathcal{O}(\gamma_1) = \mathcal{O}(\frac{1}{(\delta')^2})$ entries (see Lemma 6.15). The overall bound follows. □

The idea of the dynamic program DynProg (as shown in Algorithm 6.5) is quite simple. We start with the profile set

$$TS_0 = \left\{ \left(\underbrace{(\overbrace{m_1}^{q_0}, 0, \ldots, 0)}_{AB_1}, \underbrace{0}_{AS_1} \right), \ldots, \left(\underbrace{(\overbrace{m_K}^{q_0}, 0, \ldots, 0)}_{AB_K}, \underbrace{0}_{AS_K} \right) \right\} \tag{6.1}$$

that represents the empty schedule: for every machine type k, small jobs have not been assigned ($AS_k = 0$), and m_k machines (i.e. all machines of type k) have a total processing time of large jobs equal to 0.

Suppose that the set TS_{j-1} has been determined: it contains all profiles that can be obtained for the first $j-1$ jobs $\{1, \ldots, j-1\}$. The profiles for $\{1, \ldots, j\}$ are constructed by considering for each $t \in TS_{j-1}$ all possibilities to add j to t. Fix one $t \in TS_{j-1}$. We go over all k. If j is small on type k, then $AS_k(t)$ is simply increased by p^{r}_{kj} (Steps 6–12): we have a new profile t' where additionally j is assigned to the machine type k. If j is large on type k, all $q_\gamma = (AB_k(t))_\gamma > 0$ are taken into account (Steps 13–20): there are q_γ machines of type k in t where each has the total processing time $\gamma \cdot (\delta')^2$. If we add j to one of these machines, there is obviously one machine less with the processing time $\gamma \cdot (\delta')^2$ and one machine more with the processing time $\gamma \cdot (\delta')^2 + p^{\mathrm{r}}_{kj} = \gamma \cdot (\delta')^2 + \gamma(k,j) \cdot (\delta')^2$. Hence, q_γ decreases and $q_{\gamma+\gamma(k,j)}$ increases by one. The add operation is shown in Algorithm 6.4. Thus, each $q_\gamma > 0$ in $AB_k(t)$ generates a new profile $t' \in TS_j$.

Algorithm 6.4: add(AB_k, γ, j)

Input: AB_k, γ, j
Set $\gamma' := \gamma + \gamma(k,j)$;
$(AB_k)_{\gamma'} := (AB_k)_{\gamma'} + 1$;
$(AB_k)_\gamma := (AB_k)_\gamma - 1$;
return AB_k;

When t' has been constructed, it is checked whether $t' \in TS_j$ already holds (Steps 10 and 18), i.e. whether we have already obtained the tuple t' in another way (e.g. from another $t \in TS_{j-1}$). If no, we save t' together with the corresponding backtracking information to later construct a schedule (Steps 11–12 and 19–20). If yes, we only keep the old backtracking information. Note that the add operation is only executed if $\gamma \cdot (\delta')^2 + p^{\mathrm{r}}_{kj} \leq (1+\delta')$, i.e. $\gamma + \gamma(k,j) \leq \gamma_1$ (see Step 15): we only want to find the profiles representing schedules with makespans of at most $1+\delta'$. Similarly, the bound $AS_k(t) + p^{\mathrm{r}}_{kj} \leq m_k \cdot (1+\delta')$ for AS_k is checked in Step 7. It is therefore possible that there is not any profile $t \in TS_{j-1}$ such that j can be assigned: the value T is too small.

Then, DynProg returns the empty set (Step 22), and Oracle will therefore return $\perp$. We show that a schedule for I^{r} with a makespan of at most $(1+\delta')$ corresponds to at least one profile.

Algorithm 6.5: The dynamic program DynProg

Input: Instance I^{r}, $\delta' > 0$

1 Set TS_0 as seen in Equation (6.1);
2 **for** $j = 1, \ldots, n'$ **do**
3 $TS_j := \emptyset$;
4 **for** $t \in TS_{j-1}$ **do**
5 **for** $k = 1, \ldots, K$ **do**
6 **if** *j is small on type k* **then**
7 **if** $AS_k(t) + p^{\mathrm{r}}_{kj} \leq m_k \cdot (1+\delta')$ **then**
8 $t' := t$;
9 $AS_k(t') := AS_k(t') + p^{\mathrm{r}}_{kj}$;
10 **if** $t' \notin TS_j$ **then**
11 Backtrack$(t') := (t, k)$;
12 $TS_j := TS_j \cup \{t'\}$;
13 **else** // j `is large on type` k
14 **for** $\gamma = 0, \gamma_0, \ldots, \gamma_1 - 1$ **do**
15 **if** $q_\gamma = (AB_k(t))_\gamma > 0$ *and* $\gamma + \gamma(k, j) \leq \gamma_1$ **then**
16 $t' := t$;
17 `add`$(AB_k(t'), \gamma, j)$;
18 **if** $t' \notin TS_j$ **then**
19 Backtrack$(t') := (t, k, \gamma)$;
20 $TS_j := TS_j \cup \{t'\}$;
21 **if** $TS_j = \emptyset$ **then**
22 **return** $\emptyset$;
23 **return** $TS = (TS_0, TS_1, \ldots, TS_{n'})$;

Lemma 6.19. *Let σ be a schedule for I^r with a makespan of at most $1+\delta'$. Then the dynamic program* DynProg *generates a profile t for I^r where*

- $(AB_k(t))_\gamma = ab_k(\sigma, \gamma)$ *for all* $\gamma \in \{0, \gamma_0, \ldots, \gamma_1\}$ *and* $k \in \{1, \ldots, K\}$, *and*
- $AS_k(t) = as_k(\sigma)$ *for all* $k \in \{1, \ldots, K\}$.

Proof. The statement is easy to see because the dynamic program does exactly what a natural algorithm to construct all schedules would do: it takes all currently constructed schedules for the first $j-1$ jobs and tries for each of these schedules to assign job j to every machine i. However, only the corresponding profiles are saved.

Since σ has a makespan of at most $1+\delta'$, all conditions checked during the construction of t_σ will be satisfied. □

Lemma 6.20. *The number κ of profiles in TS_j is bounded by*

$$\kappa \leq m^{\mathcal{O}(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)}.$$

Proof. Fix one machine type k. The number of possible vectors AB_k is bounded by $(m+1)^{\mathcal{O}(1/(\delta')^2)} = m^{\mathcal{O}(1/(\delta')^2)}$ because AB_k has $\mathcal{O}(\frac{1}{(\delta')^2})$ entries (see Lemma 6.15), and each entry $(AB_k)_\gamma$ is in $\{0,1,\ldots,m_k\} \subseteq \{0,\ldots,m\}$.

For every value of AB_k, we will only save $\mathcal{O}(\frac{n'}{\delta'})$ possible values for AS_k (see Lemma 6.16). Note that we have $n' \in \mathcal{O}(\frac{m^2}{\delta}) + (\frac{\log m}{\delta})^{\mathcal{O}(K)} = \mathcal{O}(\frac{m^2}{\delta'}) + (\frac{\log m}{\delta'})^{\mathcal{O}(K)}$ because of Assumption 6.2 and $\delta' \in \Theta(\delta)$. We get

$$\mathcal{O}\left(\frac{n'}{\delta'}\right) = \mathcal{O}\left(\frac{m^2}{(\delta')^2}\right) + \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K)}.$$

If we consider all machine types k, we have the upper bound on the number of profiles of

$$\begin{aligned}
\left(m^{\mathcal{O}(1/(\delta')^2)} \cdot \mathcal{O}\left(\frac{n'}{\delta'}\right)\right)^K &= \left(m^{\mathcal{O}(1/(\delta')^2)} \cdot \left(\mathcal{O}\left(\frac{m^2}{(\delta')^2}\right) + \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K)}\right)\right)^K \\
&= \left(m^{\mathcal{O}(1/(\delta')^2)} \cdot \frac{m^2}{(\delta')^2} + m^{\mathcal{O}(1/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K)}\right)^K \\
&= \left(m^{\mathcal{O}(1/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K)}\right)^K \\
&= m^{\mathcal{O}(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)}.
\end{aligned}$$

We have used that $\frac{1}{(\delta')^2} = 2^{2\log(1/(\delta'))} \leq 2^{\mathcal{O}(1/\delta')} \leq m^{\mathcal{O}(1/\delta')}$ so that $\frac{m^2}{(\delta')^2} \cdot m^{\mathcal{O}(1/(\delta')^2)} = m^{\mathcal{O}(1/(\delta')^2)}$. □

Lemma 6.21. *The dynamic program (Algorithm 6.5) needs time in $\mathcal{O}(\frac{K}{(\delta')^2} \cdot m \cdot \kappa \cdot n') = m^{\mathcal{O}(K/(\delta')^2)}(\frac{\log m}{\delta'})^{\mathcal{O}(K^2)}$ to construct all profiles $TS_0,\ldots,TS_{n'}$.*

Proof. The for-loop for the items needs n' iterations. When an item is processed in the for-loop, it is tried to assign it to each of the $\mathcal{O}(\kappa)$ profiles in TS_{j-1}. Fix one profile $t \in TS_{j-1}$ and one machine type k. If the item is small on machine type k, we need $\mathcal{O}(1)$ to check whether j can be added to type k. If the item is large, we need $\mathcal{O}(\gamma_1) \overset{\text{Lem. 6.15}}{=} \mathcal{O}(\frac{1}{(\delta')^2})$ to check the entries q_γ and the conditions whether j can be added. Note that there are $\mathcal{O}(m_k)$ entries $q_\gamma > 0$ and therefore only $\mathcal{O}(m_k)$ cases where an item is added.

When an item is added, we need time in $\mathcal{O}(\frac{K}{(\delta')^2})$ to create the new profile t' and add the item to it (see Lemma 6.18; note that the add operation of Algorithm 6.4 needs time in $\mathcal{O}(1)$). Additionally, it has to be checked whether t' is new in TS_j. We can suppose that this can be done in the size of the profile $\mathcal{O}(\frac{K}{(\delta')^2})$: all profiles $t'' \in TS_j$ can be saved in one array where the position of t'' is given by its values $(AB_k(t''))_\gamma$ and $AS_k(t'')$. Finally, the backtracking information only has to save a pointer to the old profile from which the new one was obtained as well as the values k and/or γ. In total, the time needed to add an item is bounded by $\mathcal{O}(\frac{K}{(\delta')^2})$.

Obviously, the case where j is large on type k dominates the running time for one machine type. The dynamic program therefore needs time in

$$\mathcal{O}\left(n' \cdot \kappa \cdot \sum_{k=1}^{K} \left(\underbrace{\frac{1}{(\delta')^2}}_{\text{checks of } q_\gamma} + \underbrace{m_k}_{\text{number of } q_\gamma > 0} \cdot \underbrace{\frac{K}{(\delta')^2}}_{\text{time to add an item}} \right)\right) = \mathcal{O}\left(n' \cdot \kappa \cdot m \cdot \frac{K}{(\delta')^2}\right) .$$

We get

$$\begin{aligned}
&\mathcal{O}\left(\frac{K}{(\delta')^2} \cdot m \cdot \kappa \cdot n'\right) \\
&= \mathcal{O}\left(\frac{K}{(\delta')^2} \cdot m \cdot \left[m^{\mathcal{O}(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)}\right] \cdot \left[\left(\frac{m^2}{\delta'}\right) + \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K)}\right]\right) \\
&= \frac{K}{(\delta')^2} \cdot m^{\mathcal{O}(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)} = K \cdot m^{\mathcal{O}(K/(\delta')^2)} \cdot \frac{1}{(\delta')^2} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)} \\
&= m^{\mathcal{O}(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)}
\end{aligned}$$

with Assumption 6.2 and Lemma 6.20. □

Remark 6.22. Profiles can of course be stored in a more compact form: we only have to save the strictly positive $ab_k(\sigma, \gamma)$ and $(AB_k(t))_\gamma$. The number of entries in one profile is then bounded by $\mathcal{O}(K + m)$. However, the number of profiles κ does not decrease: it is independent of the fact whether we save the $q_\gamma = 0$ or not. Lemma 6.20 still holds so that the asymptotic running time also remains unchanged (see the proof

of Lemma 6.21). (Lemma 6.20 is obviously a rough estimate, but it is sufficient to prove that the final algorithm is single exponential in $\frac{1}{\varepsilon}$.)

6.5.4 Construction of a Schedule

The goal of this section is the construction of a schedule with a makespan of at most $1 + 3\delta'$ from a suitable profile. We first state two definitions.

Definition 6.23. *For a given profile t', the remaining total processing time for every machine type k is defined by*

$$R_k = R_k(t') := m_k \cdot (1 + \delta') - \sum_{\gamma} (AB_k(t))_\gamma \cdot \gamma \cdot (\delta')^2 \ .$$

The definition corresponds to the one of r_k in Definition 6.17.

Definition 6.24. *Take one job j in I. If its processing time is large on machine type k, it is rounded up to the next $\gamma \cdot (\delta')^2$ for $\gamma \in \mathbb{N}$. The small processing times of a job are not changed. This is the new instance I^{p} whose processing times are denoted by p^{p}_{kj}.*

I^{p} is therefore the instance I^r without the rounding of the small jobs. We have $p^{\mathrm{p}}_{kj} = p_{kj}$ if j is small on type k, and we have $p^{\mathrm{p}}_{kj} = p^{\mathrm{r}}_{kj}$ if j is large on type k.

After `DynProg`, `Oracle` (Algorithm 6.3) calls for every $t' \in TS_{n'}$ `CreateSchedule`, a function that is shown in Algorithm 6.6. First, `CreateSchedule(`t'`)` checks whether enough processing time is left for the small jobs (Steps 1–4), by controlling whether $AS_k(t') \leq R_k(t')$ holds for all $k \in \{1, \ldots, K\}$. If yes, this is sufficient to construct a schedule for I^{p} of value at most $1 + 3\delta'$ (Steps 5–9) as will be shown below in Lemma 6.25. In Step 6, the function `Backtracking` uses the backtracking information to find the set of small jobs $\mathcal{J}_k$ assigned to machine type k for all $k \in \{1, \ldots, K\}$. Moreover, it constructs a schedule σ_k for the large jobs assigned to type k for all $k \in \{1, \ldots, K\}$. The function `Backtracking` is explained in detail below (see also Algorithm 6.7). When `Backtracking(`t'`)` has finished, σ is updated according to each σ_k (Step 8). The small jobs assigned to k are greedily added to the machines $\mathcal{M}_k$ (Step 9): a machine in $\mathcal{M}_k$ gets assigned jobs in $\mathcal{J}_k$ until the total processing time of the machine exceeds $1 + 2\delta'$. Then, the next machine in $\mathcal{M}_k$ is processed in the same way. As mentioned above, it will be shown in Lemma 6.25 that this procedure will be successful.

Note that `Oracle` returns $\perp$ if it cannot construct a schedule (because $TS_{n'} = \emptyset$ or no t' satisfies $R_k(t') \leq AS_k(t')$ for all k). It will be shown in Theorem 6.27 that `Oracle` indeed satisfies the properties of Lemma 6.11.

Algorithm 6.6: `CreateSchedule(`t'`)` from `Oracle` (Algorithm 6.3)

Input: Profile t'

1 **for** $k = 1, \ldots, K$ **do**
2 Determine $R_k = R_k(t')$;
3 **if** $AS_k(t') > R_k$ **then**
4 **return** $\perp$;
5 $\sigma := \emptyset$; `// The empty schedule`
6 $(\sigma_1, \mathcal{J}_1), \ldots, (\sigma_K, \mathcal{J}_K) :=$ `Backtracking(`t'`);`
7 **for** $k = 1, \ldots, K$ **do**
8 Set $\sigma(j) := \sigma_k(j)$ for the jobs j scheduled by σ_k;
9 Assign the jobs in $\mathcal{J}_k$ greedily to the machines in $\mathcal{M}_k$ and adapt σ accordingly;
10 **return** σ;

Let us now present the `Backtracking` function as shown in Algorithm 6.7. The backtracking information of `DynProg` allows us to go directly from $t' = t'_{n'} \in TS_{n'}$ to $t'_{n'-1} \in TS_{n'-1}$ from which t'_n has been constructed. We continue and iteratively get $t'_{n'-2} \in TS_{n'-2}, \ldots, t'_j \in TS_j, \ldots, t'_1 \in TS_1, t'_0 \in TS_0$, where t'_j has been obtained from t'_{j-1}. Next, we define the σ_k and $\mathcal{J}_k$ (Step 3).

Starting from t'_0 (which is the profile of the empty schedule, see (6.1)), we have two cases. Either t'_j has been constructed from t'_{j-1} by adding j to a machine type k where it is large, and in $\mathcal{M}_k$ to one machine with the current processing time $\gamma \cdot (\delta')^2$ for one γ. The corresponding value γ and the machine type k are exactly the values stored in Backtrack(t'_j) (see Step 19 of Algorithm 6.5). The current schedule σ_k is then updated accordingly by assigning j to one machine in $\mathcal{M}_k$ with a current processing time of $\gamma \cdot (\delta')^2$ (Steps 8–11). Otherwise, j has been assigned to a type k where it is small. The value of k is stored in Backtrack(t_j) (see Step 11 of Algorithm 6.5). The job j is then added to $\mathcal{J}_k$ (Steps 12–14). The assignment of large jobs to the machines is rendered more comfortable by the function extime$(\cdot)$: it keeps track of the current total processing time of large jobs on every machine.

Lemma 6.25. *Let $t \in TS_{n'}$ be a profile for which $AS_k(t) \leq R_k(t)$ holds for all k. Then,* `CreateSchedule` *and* `Backtracking` *(Algorithms 6.6 and 6.7) return a schedule σ for I^{P} with a makespan of at most $1 + 3\delta'$. It is also a schedule for I of value at most $(1 + 3\delta')T$.*

Proof. As we have $AS_k(t) \leq R_k(t)$, `CreateSchedule(`t`)` calls `Backtracking(`t`)`, which returns the schedules σ_k and the job sets $\mathcal{J}_k$ for all $k \in \{1, \ldots, K\}$. For each $k \in \{1, \ldots, K\}$, the algorithm takes the schedule σ_k for the machines $\mathcal{M}_k$ and greedily

Algorithm 6.7: `Backtracking` (t') from `CreateSchedule` (t') (Algorithm 6.6)

Input: Profile t'

Determine by backtracking the profiles $t' = t'_{n'} \in TS_{n'}, t'_{n'-1} \in TS_{n'-1}, \ldots, t'_j \in TS_j, \ldots, t'_1 \in TS_1, t'_0 \in TS_0$ from which t' has been constructed;

for $k = 1, \ldots, K$ **do**

  $\sigma_k := \emptyset$ and $\mathcal{J}_k := \emptyset$;

for $i = 1, \ldots, m$ **do**

  extime$(i) := 0$; `// Total processing time of large jobs on machine` i

for $j = 1, \ldots, n'$ **do**

  if Backtrack(t'_j) *is of the form* (t'_{j-1}, k, γ) **then**

    `//` j `is added to a machine type where it is large`

    Take k and γ from Backtrack(t'_j);

    Use extime$(\cdot)$ to find one machine i in $\mathcal{M}_k$ with a total processing time of large jobs equal to $\gamma \cdot (\delta')^2$;

    Add j to i: $\sigma_k(j) := i$;

    extime$(i) :=$ extime$(i) + p^{\mathrm{r}}_{kj}$;

  else `//` j `is added to a machine type where it is small`

    Take k from Backtrack(t'_j) ; `//` Backtrack$(t'_j) = (t'_{j-1}, k)$

    $\mathcal{J}_k := \mathcal{J}_k \cup \{j\}$;

return $(\sigma_1, \mathcal{J}_1), \ldots, (\sigma_K, \mathcal{J}_K)$;

assigns the small jobs in $\mathcal{J}_k$ as explained above. When this is done for all $k \in \{1, \ldots, K\}$, the schedule σ for I^{p} has been constructed. Technically, the σ_k are schedules for jobs in I^{r}, and the sets $\mathcal{J}_k$ are also taken from $\mathcal{J}(I^{\mathrm{r}})$. However, they can be trivially adapted to I^{p}: the instances I^{p} and I^{r} have a one-to-one correspondence of the jobs and differ only in the processing times p^{p}_{kj} and p^{r}_{kj}.

We have to show that all small jobs in the sets $\mathcal{J}_k$ can indeed be greedily assigned and that the resulting schedule has a makespan of at most $1 + 3\delta'$. We first state several identities. For the total processing time of large jobs on a machine $i \in \mathcal{M}_k$, $b_i(\sigma_k) = b_i(\sigma)$ holds: we have $p^{\mathrm{r}}_{kj} = p^{\mathrm{p}}_{kj}$ for jobs j that are large on a machine type k, and σ_k is σ restricted to the large jobs assigned to machine type k. Hence, we also have $ab_k(\sigma, \gamma) = ab_k(\sigma_k, \gamma)$ for each γ and k. Moreover, $ab_k(\sigma_k, \gamma) = (AB_k(t))_\gamma$ holds: it is clear that the backtracking constructs σ_k in such a way that it has a profile for the large jobs on k that corresponds to t. By induction, it can indeed be shown that $(AB_k(t_j))_\gamma = ab_k((\sigma_k)^{(j)}, \gamma)$, where $(\sigma_k)^{(j)}$ is the schedule σ_k restricted to the large jobs in $\{1, \ldots, j\}$, and $t_j \in TS_j$ is the profile of which t is constructed. To sum up, we have $ab_k(\sigma, \gamma) = ab_k(\sigma_k, \gamma) = (AB_k(t))_\gamma$ for all $\gamma \in \{0, \gamma_0, \ldots, \gamma_1\}$.

From this, we can directly derive the following identity for the remaining machine capacity on machines of type k:

$$\begin{aligned} r_k(\sigma) &= \sum_{i \in \mathcal{M}_k} \left(1 + \delta' - b_i(\sigma)\right) = \left(1 + \delta'\right) \cdot m_k - \sum_{\gamma} \sum_{i : b_i(\sigma) = \gamma \cdot (\delta')^2} \gamma \cdot (\delta')^2 \\ &= \left(1 + \delta'\right) \cdot m_k - \sum_{\gamma} ab_k(\sigma, \gamma) \cdot \gamma \cdot (\delta')^2 \\ &= \left(1 + \delta'\right) \cdot m_k - \sum_{\gamma} (AB_k(t))_\gamma \cdot \gamma \cdot (\delta')^2 = R_k(t) \ . \end{aligned}$$

Note first that $p^{\mathrm{p}}_{kj} \leq p^{\mathrm{r}}_{kj} + \frac{m_k \cdot \delta'}{n'}$ holds for small jobs according to the definition of I^{r} and I^{p} (see Subsection 6.5.2 and Definition 6.24). Thus, we have $\sum_{j \in \mathcal{J}_k} p^{\mathrm{p}}_{kj} \leq \sum_{j \in \mathcal{J}_k} (p^{\mathrm{r}}_{kj} + m_k \cdot \delta' / n') \leq (\sum_{j \in \mathcal{J}_k} p^{\mathrm{r}}_{kj}) + m_k \cdot \delta'$. As the identity $AS_k(t) = \sum_{j \in \mathcal{J}_k} p^{\mathrm{r}}_{kj}$ holds for the total processing time of small jobs assigned to type k, we get $\sum_{j \in \mathcal{J}_k} p^{\mathrm{p}}_{kj} \leq AS_k(t) + m_k \cdot \delta'$. Since we have $AS_k(t) \leq R_k(t)$ by assumption, we finally get $\sum_{j \in \mathcal{J}_k} p^{\mathrm{p}}_{kj} \leq R_k(t) + m_k \cdot \delta'$: the small jobs in $\mathcal{J}_k$ with their processing times p^{p}_{kj} only slightly exceed the remaining capacity of the machines of type k.

Assume for the sake of contradiction that there is one type k for which all small jobs in $\mathcal{J}_k$ cannot be greedily scheduled. Thus, every machine $i \in \mathcal{M}_k$ has a processing

time larger than $1 + 2\delta'$, and there are still small jobs left. Let $\delta_i > 0$ be the processing time by which all jobs assigned to machine i exceed $1 + 2\delta'$. We have

$$\begin{aligned}\sum_{j \in \mathcal{J}_k} p^{\mathrm{p}}_{kj} &\geq \sum_{i \in \mathcal{M}_k} \big(1 + 2\delta' + \delta_i - b_i(\sigma)\big) > \sum_{i \in \mathcal{M}_k} \big(1 + 2\delta' - b_i\big) \\ &= \sum_{i \in \mathcal{M}_k} \big(1 + \delta' - b_i\big) + m_k \cdot \delta' = r_k(\sigma) + m_k \cdot \delta' = R_k(t) + m_k \cdot \delta' \\ &\geq \sum_{j \in \mathcal{M}_k} p^{\mathrm{p}}_{kj} \ .\end{aligned}$$

This is a contradiction. All small jobs can therefore be greedily scheduled. Let δ_i still be the amount by which the total processing time of machine i exceeds $1 + 2\delta'$. (It is of course possible that some machines may have a makespan smaller than $1 + 2\delta'$ even after the scheduling of the small jobs. Then, we set $\delta_i = 0$.) As we have $p^{\mathrm{p}}_{kj} < \delta'$ for small jobs on type k, the bound $\delta_i \leq \delta'$ holds: at most one small job is assigned to a machine i such that the processing time exceeds $1 + 2\delta'$. Hence, we have for the makespan $\max_k \max_{i \in \mathcal{M}_k} 1 + 2\delta' + \delta_i \leq 1 + 3\delta'$.

Note that the proof by contradiction above is in fact independent of whether some small jobs are still left or all small jobs have been assigned, but all machines $i \in \mathcal{M}_k$ have a processing time that exceeds $1 + 2\delta'$. It is shown that there must be at least one machine with a total processing time of at most $1 + 2\delta'$ because we get the contradiction otherwise.

Finally, the schedule σ for I^{p} is also one for I^{scale} with a makespan of at most $1 + 3\delta'$. In fact, small processing times are identical in I^{p} and I^{scale}, and if we undo the rounding of the large processing times, the makespan does not increase because the large jobs were rounded up in I^{p}. After undoing the scaling of I^{scale}, the schedule σ is a solution to I of value at most $(1 + 3\delta')T$. □

We now state the overall running time for `Backtracking` and `CreateSchedule`.

Lemma 6.26. *One call of* `Backtracking`(t') *(Algorithm 6.7) needs time in* $\mathcal{O}(K + n' \cdot m)$. *Therefore, the total running time for all calls of* `CreateSchedule` *(Algorithm 6.6), i.e. of the for-loop in* `Oracle` *(Algorithm 6.3), is in* $\mathcal{O}(\kappa \cdot \frac{K}{(\delta')^2} + n' \cdot m) = m^{\mathcal{O}(K/(\delta')^2)} \cdot (\frac{\log m}{\delta'})^{\mathcal{O}(K^2)}$.

Proof. Let us first consider `Backtracking`. Creating a list consisting of pointers to the t'_j can be done in $\mathcal{O}(n')$: we just follow the pointers given by the backtracking information $\mathrm{Backtrack}(t'_j)$. The creation of the σ_k and $\mathcal{J}_k$ is in $\mathcal{O}(K)$, and setting the values $\mathrm{extime}(i)$ to 0 in $\mathcal{O}(m)$. This is followed by the n' iterations of the for-loop. In one iteration, the case where j is small can be done in $\mathcal{O}(1)$. The case where j is large needs $\mathcal{O}(m)$: we have to check all $i \in \mathcal{M}_k$ and their corresponding value $\mathrm{extime}(i)$. In total, we need time $\mathcal{O}(K + n' \cdot m)$ for one call of `Backtracking`.

Let us now consider `CreateSchedule`. One call of `CreateSchedule`(t') first determines all R_k. Note that $R_k = \sum_\gamma (AB_k(t'))_\gamma \cdot \gamma \cdot (\delta')^2$ so that determining all R_k needs $\mathcal{O}(K \cdot \frac{1}{(\delta')^2})$ including the checks of the if-condition (see the bound on $\gamma \leq \gamma_1$ in Lemma 6.15). Should one of the if-conditions fail, the call of `CreateSchedule`(t') is terminated such that the oracle checks the next $t'' \in TS_{n'}$. In the worst case, `CreateSchedule` is therefore called κ times without constructing a schedule, which needs in total $\mathcal{O}(\kappa \cdot \frac{K}{(\delta')^2})$.

The schedule σ is constructed if we have $AS_k(t') \leq R_k(t')$ for all machine types k. `CreateSchedule` is not called again afterwards because the oracle terminates if σ has been found. The construction of σ first calls `Backtracking`(t'), which needs $\mathcal{O}(K + n' \cdot m)$, and then greedily assigns the small jobs to the machines. Let n_k be the number of large jobs assigned to $\mathcal{M}_k$. The greedy procedure then needs $\mathcal{O}(n_k + m_k + |\mathcal{J}_k|)$ for one k, i.e. $\mathcal{O}(n' + m)$ for all K machine types. In total, we have a time complexity first for all checks of the if-condition and then for the construction of σ in

$$\begin{aligned}
&\mathcal{O}\left(\kappa \cdot \frac{K}{(\delta')^2} + (K + n' \cdot m) + (n' + m)\right) = \mathcal{O}\left(\kappa \cdot \frac{K}{(\delta')^2} + n' \cdot m\right) \\
&= \frac{K}{(\delta')^2} \cdot m^{\mathcal{O}(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)} + m \cdot \left(\mathcal{O}\left(\frac{m^2}{\delta'}\right) + \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K)}\right) \\
&= m^{\mathcal{O}(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)},
\end{aligned}$$

where we have used Assumption 6.2 and Lemma 6.20. □

Theorem 6.27. *Let I be an instance with $1 \leq \mathrm{OPT}(I) \leq m$. The dual approximation approach (Algorithm 6.2) with the* `Oracle` *function of Algorithm 6.3 is a PTAS: it finds for given $\delta > 0$ and $\delta' := \frac{\delta}{5}$ a solution of value at most $(1+\delta)\mathrm{OPT}(I)$. The running time is in*

$$m^{\mathcal{O}(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)} = m^{\mathcal{O}(K/\delta^2)} \cdot \left(\frac{\log m}{\delta}\right)^{\mathcal{O}(K^2)}.$$

Proof. Lemma 6.11 states that the binary search returns a solution of value at most $(1 + (C+1)\delta' + C(\delta')^2)\mathrm{OPT}(I)$ if `Oracle` has the stated properties. These will now be shown.

Suppose first that `Oracle` (Algorithm 6.3) returns a solution σ for T and input I. First, the instance is scaled by `Oracle` and becomes I^{scale}, which is rounded to get I^{r}. The solution σ is then determined for I^{p}. It is clear that a schedule is only returned if there is a profile $t \in TS_{n'}$ for which $AS_k(t) \leq R_k(t)$ holds. If this is the case, the

returned schedule σ has a makespan of at most $1 + 3\delta'$ as seen in Lemma 6.25. It is easy to see that it is also a schedule for I with a makespan of at most $(1 + 3\delta')T$.

It remains to prove that there is not a solution to I of value (at most) T if `Oracle(`I, T`)` returns $\perp$. We show this by demonstrating that the oracle will always return a solution if there is a solution of value (at most) T:

- The instance I satisfies $\mathrm{OPT}(I) \leq T$, which implies that
- the instance I^{scale} has an optimum $\mathrm{OPT}(I^{\mathrm{scale}}) \leq 1$, which again implies that
- the instance I^{r} satisfies $\mathrm{OPT}(I^{\mathrm{r}}) \leq 1 + \delta'$ (see Lemma 6.14). Let σ be an optimal schedule for I^{r}. Then,
- `DynProg`, `CreateSchedule` and `Backtracking` will find a schedule: Lemma 6.19 states that there is a profile t_σ for which we have $ab_k(\sigma, \gamma) = (AB_k(t_\sigma))_\gamma$ and $as_k(\sigma) = AS_k(t_\sigma)$ for all k and γ. As in the proof of Lemma 6.25, the identity $ab_k(\sigma, \gamma) = (AB_k(t_\sigma))_\gamma$ implies $r_k(\sigma) = R_k(t_\sigma)$. Since the small jobs assigned by σ to the machine type k have to fit into the remaining machine capacity $r_k(\sigma)$, we have $AS_k(t_\sigma) = as_k(\sigma) \leq r_k(\sigma) = R_k(t_\sigma)$. By Lemma 6.25, the function `CreateSchedule(`t_σ`)` will therefore construct a schedule σ' for I^{p} with a makespan of at most $1 + 3\delta'$, which is also a solution to I with a makespan of at most $(1 + 3\delta')T$.

Hence, the oracle has the desired properties. We now want that the bound

$$(1 + (C + 1)\delta' + C(\delta')^2)\mathrm{OPT}(I) \leq (1 + \delta)\mathrm{OPT}(I)$$

is satisfied. Note that we have $C = 3$. Set $\delta' := \frac{\delta}{5}$ such that $(C + 1)\delta' + C(\delta')^2 = 4\delta' + 3(\delta')^2 \leq \delta$ holds because $\delta \leq \frac{1}{3}$.

Let us bound the overall running time of the binary search, i.e. Algorithm 6.2. The binary search needs $\mathcal{O}(\log(\frac{m}{\delta'})) = \mathcal{O}(\log(\frac{m}{\delta}))$ iterations as seen in Lemma 6.11. In each iteration, `Oracle` (Algorithm 6.3) is called. When called, `Oracle` needs $\mathcal{O}(K \cdot n')$ to obtain first I^{scale} and then I^{r} as stated in Lemma 6.12 and 6.14. Lemma 6.21 bounds the running time for the dynamic program by $m^{\mathcal{O}(K/(\delta')^2)}(\frac{\log m}{\delta'})^{\mathcal{O}(K^2)}$. The for-loop of `Oracle` also needs time in $m^{\mathcal{O}(K/(\delta')^2)} \cdot (\frac{\log m}{\delta'})^{\mathcal{O}(K^2)}$ (see Lemma 6.26). Overall, we have a time complexity in

$$\begin{aligned}
&\mathcal{O}\left(\log\left(\frac{m}{\delta'}\right)\right) \cdot \left(\mathcal{O}\left(K \cdot n'\right) + m^{\mathcal{O}(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)}\right) \\
&= \mathcal{O}\left(\log\left(\frac{m}{\delta'}\right)\right) \cdot m^{\mathcal{O}(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)} \\
&= m^{\mathcal{O}(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{\mathcal{O}(K^2)} = m^{\mathcal{O}(K/\delta^2)} \cdot \left(\frac{\log m}{\delta}\right)^{\mathcal{O}(K^2)} .
\end{aligned}$$

We have used Assumption 6.2 to see that $\mathcal{O}(K \cdot n') = \mathcal{O}(K \cdot [\frac{m^2}{\delta'} + (\frac{\log m}{\delta'})^{\mathcal{O}(K)}]) \subseteq m^{\mathcal{O}(K/(\delta')^2)} \cdot (\frac{\log m}{\delta'})^{\mathcal{O}(K^2)}$. □

6.6 The General PTAS

The PTAS is presented in Algorithm 6.8.

Algorithm 6.8: An overview of the PTAS

Input: Instance I, $\varepsilon > 0$

1 Set $\varepsilon' := \frac{\varepsilon}{5}$ and $\delta := \varepsilon'$;

2 Construct from I the instance I^{merge} as shown in Section 6.4;

3 Call the binary search for I^{merge} (Algorithm 6.2) with $\delta' = \frac{\delta}{5} = \frac{\varepsilon}{25}$;

4 Undo the combination of the items in I^{merge} and then the rounding of I^{round} to get a schedule σ for I;

5 **return** σ;

Lemma 6.28. *For the values of $\varepsilon' = \frac{\varepsilon}{5}, \delta = \varepsilon'$ and $\delta' = \frac{\varepsilon}{25}$, the algorithm returns a solution of value $A_\varepsilon(I) \leq (1+\varepsilon)\,\mathrm{OPT}(I)$.*

Proof. As seen in Theorem 6.27, the binary search returns a solution to I^{merge} whose makespan is bounded by $(1+\delta)\,\mathrm{OPT}(I^{\text{merge}})$. For convenience, we denote the makespan by $A_m(I^{\text{merge}})$. By undoing the combination of the items, we get a solution to I^{round} of the same makespan. Thus,

- transforming I^{round} into I^{merge},
- applying the binary search and then
- undoing the combination of items

is an algorithm A_r for I^{round} with

$$A_r\left(I^{\text{round}}\right) = A_m\left(I^{\text{merge}}\right) \overset{\text{Thm. 6.27}}{\leq} (1+\delta)\,\mathrm{OPT}\left(I^{\text{merge}}\right) = \left(1+\varepsilon'\right)\mathrm{OPT}\left(I^{\text{merge}}\right) \ .$$

Since $\mathrm{OPT}(I^{\text{round}}) \leq \mathrm{OPT}(I^{\text{merge}}) \leq \mathrm{OPT}(I^{\text{round}}) + \varepsilon'$ (see Theorem 6.9), we have an algorithm with

$$\begin{aligned} A_r\left(I^{\text{round}}\right) &\leq \left(1+\varepsilon'\right)\mathrm{OPT}\left(I^{\text{merge}}\right) \leq \left(1+\varepsilon'\right)\left(\mathrm{OPT}(I^{\text{round}}) + \varepsilon'\right) \\ &= \left(1+\varepsilon'\right)\mathrm{OPT}\left(I^{\text{round}}\right) + \varepsilon'\left(1+\varepsilon'\right) \ . \end{aligned}$$

By Lemma 6.4, this implies an algorithm for I with

$$A_\varepsilon(I) \leq \left(\left(1+\varepsilon'\right)^3 + \varepsilon' \cdot \left(1+\varepsilon'\right)^2\right) \mathrm{OPT}(I) \leq (1+\varepsilon)\,\mathrm{OPT}(I) \ .$$

This upper bound holds because

$$\begin{aligned}
\left(1+\varepsilon'\right)^3 + \varepsilon' \cdot \left(1+\varepsilon'\right)^2 &= \left(1+\frac{\varepsilon}{5}\right)^3 + \frac{\varepsilon}{5} \cdot \left(1+\frac{\varepsilon}{5}\right)^2 \\
&= \left(1+3\frac{\varepsilon}{5}+3\frac{\varepsilon^2}{25}+\frac{\varepsilon^3}{125}\right) + \frac{\varepsilon}{5} + 2\frac{\varepsilon^2}{25} + \frac{\varepsilon^3}{125} \\
&\overset{\varepsilon\leq 1/2}{\leq} 1+\frac{3}{5}\varepsilon+\frac{3}{50}\varepsilon+\frac{\varepsilon}{4\cdot 125}+\frac{\varepsilon}{5}+\frac{\varepsilon}{25}+\frac{\varepsilon}{4\cdot 125} \\
&\leq 1+\varepsilon \ .
\end{aligned}$$

(Main parts of the proof are taken from [49].) □

Lemma 6.29. *The PTAS has a running time in*

$$\mathcal{O}(K\cdot n) + m^{\mathcal{O}(K/\varepsilon^2)} \cdot \left(\frac{\log m}{\varepsilon}\right)^{\mathcal{O}(K^2)} \ .$$

Proof. The time to construct I^{merge} is in $\mathcal{O}(n\cdot K)$ as stated in Theorem 6.10. The running time for the binary search of $m^{\mathcal{O}(K/(\delta')^2)\cdot(\frac{\log m}{\delta'})^{\mathcal{O}(K^2)}} = m^{\mathcal{O}(K/\varepsilon^2)\cdot(\frac{\log m}{\varepsilon})^{\mathcal{O}(K^2)}}$ is shown in Theorem 6.27. When I^{merge} is constructed from I^{round}, it is saved for every item $j \in I^{\text{merge}}$ which items in I^{round} have been combined into j. (This is the set list(j) in Algorithm 6.1.) Reversing the combination of the items is therefore in $\mathcal{O}(n)$. Undoing the rounding can finally be done in $\mathcal{O}(K\cdot n)$. □

These two lemmas show Theorem 6.1.

7 Concluding Remarks

Chapter 2 presented an AFPTAS for the Bin Packing Problem (BP) and the Variable-sized Bin Packing Problem (VBP) with the approximation guarantee $(1+\varepsilon)\mathrm{OPT}(I,C) + \mathcal{O}(\log^2(\frac{1}{\varepsilon}))$, i.e. with a smaller additive term. This was achieved by a practical application of Shmonin's theoretical result [80], which is based on the algorithm by Karmarkar and Karp [56]. Contrary to Shmonin's result, the linear programs are only approximately solved with the max-min resource sharing algorithm by Grigoriadis et al. [33] applied to VBP. To obtain the asymptotic approximation ratio of $(1+\mathcal{O}(\varepsilon))$, we therefore had to show that the number of item sizes and therefore $\mathrm{Area}(I^{(k)})$ halves in every iteration. Note that the max-min resource sharing approach also improved the running time. An interesting question is whether these techniques can be applied to other problems to find better AFPTAS. Bin Covering might be such a problem, for which Jansen and Solis-Oba [52] found an AFPTAS with an additive term in $\mathcal{O}(\frac{1}{\varepsilon^3})$. Other examples may be Class Constrained Bin Packing [19], Bin Packing with Size Preserving Fragmentation and Bin Packing with Size Increasing Fragmentation [77, 78]. Improved approximation algorithms can also be expected e.g. for Bin Packing with Cardinality Constraints, Scheduling Multiprocessor Tasks and Resource-constrained Scheduling.

The crucial open question for VBP is the construction of an AFPTAS with an additive term in $o(\log^2 \frac{1}{\varepsilon})$. As already mentioned, Shmonin's bound $\mathrm{OPT}(I) \leq \mathrm{LIN}(I) + \mathcal{O}(\log^2 d)$ for Cutting Stock [80] we have used is based on an algorithm for Bin Packing by Karmarkar and Karp [56] with $A(I) \leq \mathrm{OPT}(I) + \mathcal{O}(\log^2 \mathrm{OPT}(I))$. Hoberg and Rothvoß [35] were recently able to show with a constructive proof that we have $\mathrm{OPT}(I) \leq \mathrm{LIN}(I) + \mathcal{O}(\log(\mathrm{LIN}(I)))$ for Bin Packing and $\mathrm{OPT}(I) \leq \mathrm{LIN}(I) + \mathcal{O}(\log(d))$ for Cutting Stock. It may be the basis of an AFPTAS for BP or VBP with a smaller additive constant in $\mathcal{O}(\log(\frac{1}{\varepsilon}))$. One important step for this is probably the derandomization of Hoberg's and Rothvoß's algorithm.

The Chapters 3, 4 and 5 decreased the running time of the AFPTAS for BP and VBP. This was done by improving the column generation subroutines of both algorithms. First, Chapter 3 introduced the Knapsack Problem with Inversely Proportional Profits (KPIP) and presented faster FPTAS for its variants: the running time is improved for large $M = |C|$ by using the values $F_j(i)$ determined by dynamic programming for all knapsack sizes in $C_b \subset (2^{-(b+1)}, 2^{-b}]$.

Chapter 4 presented a faster FPTAS for the Unbounded Knapsack Problem (UKP) to improve the column generation for BP. The most important steps in the FPTAS are the creation of the item set $\tilde{I}$ by gluing and the introduction of $a_{\text{eff}-c}$. This guarantees the existence of an approximate structured solution with a lower bound (see Definition 4.16). Therefore, the approximate dynamic program has to store less tuples (p, s, k) than in the case without the structure.

An open question is an improvement of the running time and/or the space complexity e.g. with the techniques by Magazine and Oguz [67] or by Kellerer and Pferschy [59–61]. Recently, Lokshtanov and Nederlof [65] showed that the 0-1 Knapsack Problem (0-1 KP) and the Subset Sum Problem have a pseudo-polynomial time and only polynomial space algorithm. Subset Sum is a special case of the Knapsack Problem where the profit of an item is equal to its size, i.e. $p_j = s_j$. Moreover, it was shown that Unary Subset Sum is in Logspace [18, 54]. Gál et al. [24] described an FPTAS for Subset Sum whose space complexity is in $\mathcal{O}(\frac{1}{\varepsilon})$, i.e. which does not depend on the actual input size, and whose running time is in $\mathcal{O}(\frac{1}{\varepsilon}n(n + \log n + \log \frac{1}{\varepsilon}))$. Can any of these results be further extended to improve the space complexity of an UKP FPTAS?

It is also an open question whether the ideas for UKP can be extended to the normal 0-1 KP or other KP variants as well as used for column generation of other optimization problems. The currently fastest known algorithm for 0-1 KP is due to Kellerer and Pferschy [59–61].

Chapter 5 finally presented the combination of the techniques of Chapters 3 and 4. The running time of the AFPTAS for VBP is now close to the AFPTAS for BP: they only differ by a multiplicative factor of $\mathcal{O}(\log \frac{1}{\varepsilon})$ (and the additive term $\mathcal{O}(M)$). It is expected that the better running time of the FPTAS for UKP and UKPIP will also improve the running time of AFPTAS for the problems we have mentioned above. Finally, there is the open question of Remark 5.37: can the modified approximate dynamic programming of Section 5.7 further improve the running time and space complexity of FPTAS for UKP, UKPIP or other variants of KP and KPIP?

Chapter 6 described a PTAS for Scheduling on Unrelated Machines of Few Different Types $((\mathrm{P}m_1, \ldots, \mathrm{P}m_K)||C_{\max})$ that is single exponential in $\frac{1}{\varepsilon}$. A natural question is of course the existence of an EPTAS. Another interesting task is the generalization of this algorithm to jobs with $\Delta \geq 2$ dimensions because Bonifaci and Wiese [8] first considered this more general case. However, we run into trouble when it comes to the preprocessing of jobs in Section 6.4. For one dimension $\Delta = 1$, jobs can be partitioned into fast and slow ones on every machine type. The partition is then explicitly used to set $p_{kj}^{\text{round}} = \infty$ for jobs that are slow on a machine type k. This allowed us to bound the number of profiles in Lemma 6.5 and therefore the number of jobs n' in I^{merge}. Hence, we were able to bound the time complexity of the oracle: the bound on n'

limited the overall running time of the dynamic program that generates all possible profiles. In $\Delta \geq 2$ dimensions, a partition into fast and slow jobs on a machine type may not be possible in a way similar to ours because a job may be fast in one, but slow in another dimension.

Optimizations for the running time of our algorithm are probably possible. For instance, the jobs large on a machine type are rounded linearly and then the rounded processing times are used to derive the profiles. An adapted rounding may lead to a smaller number of profiles. Geometric rounding may be an option, but then the processing times are not necessarily a discrete multiple of a value like $(\delta')^2$. One possibility may therefore be not to round the processing times. Instead, the interval $[0, 1 + \delta']$ is partitioned geometrically. For each interval of the partition, the number of machines is saved whose total processing time lies in this interval, which may lead to an improved running time.

Finally, an open question is whether the FPTAS for Scheduling on m Unrelated Machines ($\mathrm{R}m \,||\, C_{\max}$) (where m is constant) presented by Jansen and Mastrolilli [49] can also be adapted to $(\mathrm{P}m_1, \ldots, \mathrm{P}m_K) \,||\, C_{\max}$ and yield an FPTAS for a constant number of machine types K.

A Solving the LPs Approximately: The Details

We first introduce the max-min resource sharing algorithm by Grigoriadis et al. [33]. Then, the missing proofs of Subsection 2.5.1 are presented.

A.1 Max-Min Resource Sharing

Let $f_i : B \to \mathbb{R}_{\geq 0}, i \in \{1, \ldots, N\}$, be non-negative concave functions over a non-empty, convex and compact set $B \subset \mathbb{R}^L$. The goal is to find

$$\lambda^* := \max \{\lambda \mid \min \{f_i(v), i \in \{1, \ldots, N\}\} \geq \lambda, v \in B\} \ ,$$

i.e. to solve

$$\max \lambda \quad \text{s.t.} \quad f(v) := \begin{pmatrix} f_1(v) \\ \vdots \\ f_N(v) \end{pmatrix} \geq \lambda \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \ v \in B \ . \tag{A.1}$$

Let $e := (1, \ldots, 1)^{\mathrm{T}}$ be the all-1 vector in $\mathbb{R}^N$ and $\Gamma := \{p \in \mathbb{R}^N_{\geq 0} \mid p^{\mathrm{T}} e = 1\}$ be the standard simplex in $\mathbb{R}^N$. Then obviously $\lambda^* = \max_{v \in B} \min_{p \in \Gamma} p^{\mathrm{T}} f(v)$, and by duality

$$\lambda^* = \max_{v \in B} \min_{p \in \Gamma} p^{\mathrm{T}} f(v) = \min_{p \in \Gamma} \max_{v \in B} p^{\mathrm{T}} f(v) = \min_{p \in \Gamma} \Lambda(p)$$

where $\Lambda(p) := \max\{p^{\mathrm{T}} f(v) \mid v \in B\}$ is the block problem associated to the max-min resource sharing problem.

Let $\bar{\varepsilon} > 0$. We are looking for an approximate solution to (A.1):

$$\text{Find } v \in B \text{ such that } f(v) \geq (1 - \bar{\varepsilon})\, \lambda^* e \ . \tag{A.2}$$

This can be done with an algorithm by Grigoriadis et al. [33] (see also [41]). This algorithm relies on a blocksolver $ABS(p, t)$ that solves the block problem with the accuracy $t = \frac{\bar{\varepsilon}}{6}$:

$$\text{Find } v \in B \text{ for a given } p \in \Gamma, t = \Theta(\bar{\varepsilon}), \text{ such that } p^T f(v) \geq (1 - t)\, \Lambda(p).$$

First, the algorithm computes an initial solution

$$v^{(0)} := \frac{1}{N} \sum_{i=1}^{N} \hat{v}^{(i)} \text{ where } \hat{v}^{(i)} := ABS\left(e_i, \frac{1}{2}\right) \ . \tag{A.3}$$

The vector $e_i \in \mathbb{R}^N$ is the unit vector with 1 in the i.th component and 0 otherwise, therefore $\hat{v}^{(i)}$ satisfies $e_i^{\mathrm{T}} f(\hat{v}^{(i)}) = f_i(\hat{v}^{(i)}) \geq (1 - \frac{1}{2})\Lambda(e_i) = \frac{1}{2}\Lambda(e_i)$. The algorithm then improves upon this initial solution $v^{(0)}$: it sets $v := v^{(0)}$ and computes the solution $\theta(f(v))$ of

$$\frac{t}{N} \sum_{i=1}^{N} \frac{\theta}{f_i(v) - \theta} = 1,\ \theta \in (0, \min\{f_1(v), \ldots, f_N(v)\}) \ . \tag{A.4}$$

Note that the solution $\theta(f(v))$ is unique because the sum is strictly increasing in θ. Then, a new vector $p = p(f(v)) = (p_1, \ldots, p_N)^{\mathrm{T}} \in \Gamma$ is computed with

$$p_i := \frac{t}{N} \frac{\theta(f(v))}{f_i(v) - \theta(f(v))} \text{ for } i = 1, \ldots, N \ . \tag{A.5}$$

After this, $\hat{v} := ABS(p(f(v)), t)$ is calculated as well as the following stopping condition checked:

$$\nu(v, \hat{v}) := \frac{p^T f(\hat{v}) - p^T f(v)}{p^T f(\hat{v}) + p^T f(v)} \leq t \ . \tag{A.6}$$

If not satisfied, v is set to $v = (1 - \tau)v + \tau\hat{v}$, where

$$\tau = \tau(f(v)) := \frac{t\theta(f(v))\nu(v, \hat{v})}{2N(p^{\mathrm{T}} f(v) + p^{\mathrm{T}} f(\hat{v}))} \ . \tag{A.7}$$

The algorithm then restarts with the calculation of $\theta(f(v))$ and continues until the condition $\nu(v, \hat{v}) \leq t$ is satisfied. Then it returns v. A speed-up is possible by beginning with $\bar{\varepsilon}_0 = 1 - \frac{1}{2N}$ and setting $\bar{\varepsilon}_s = \frac{\bar{\varepsilon}_{s-1}}{2}$ when the stopping condition is met until the stopping condition is satisfied for an $\bar{\varepsilon}_s \leq \bar{\varepsilon}$. The algorithm is presented in Algorithm A.1. (The pseudocode representation was adapted from [14].)

Algorithm A.1: The max-min resource sharing algorithm (taken from [14])

```
Compute v^(0). Set v := v^(0), s := 0, ε_0 := 1 − 1/(2N);
repeat // scaling phase
    s := s + 1, ε̄_s := ε̄_{s−1}/2, t := ε̄_s/6, v := v^(s−1);
    while true do // coordination phase
        Compute θ(f(v)) and p(f(v));
        v̂ := ABS(p(f(v)), t);
        Compute ν(v, v̂);
        if ν(v, v̂) ≤ t then v^(s) := v, break;
        Compute step length τ and set v := (1 − τ)v + τv̂;
until ε̄_s ≤ ε̄;
return v^(s);
```

Remark A.1. It may not be possible to compute the root $\theta(f(v))$ of (A.4) exactly. It can be shown that it is sufficient to approximately determine it up to a relative accuracy of $\mathcal{O}(\frac{\varepsilon^2}{N})$ [41, p. 307]. The loss of accuracy because of this can be neglected, and the approximated root can e.g. be found with the Newton method in $\mathcal{O}(N \log\log(\frac{N}{\varepsilon}))$. Note that the approximation to $\theta(f(v))$ is then a number that can be expressed exactly.

Theorem A.2. *Let $\bar{\varepsilon} > 0$ and let $f_i : B \to \mathbb{R}_{\geq 0}$, $i \in \{1, \ldots, N\}$, be non-negative concave functions over a non-empty, convex and compact set $B \subset \mathbb{R}^L$. Then Algorithm A.1 finds an approximate solution $v \in B$ to the max-min resource sharing problem* (A.1)*, where v satisfies* (A.2)*. The algorithm needs $\mathcal{O}(N(\log N + \frac{1}{\bar{\varepsilon}^2}))$ iterations, where in each iteration a call to the blocksolver is needed as well as an overhead of $\mathcal{O}(N \log\log(N\frac{1}{\bar{\varepsilon}}))$ incurs.*

Proof. Correctness and running time were proved by Grigoriadis et al. [33] (see also [41]). □

A.2 Missing Proofs

Proof of Lemma 2.12. Note that the configuration generated by the blocksolver (2.7) does not depend on r, but only on the simplex vector p. For different values r, but the same vector p, the blocksolver will choose the same bin size and the same configuration and then set the corresponding variable $v_{j_0}^{(l_0)} := \frac{r}{c_{l_0}}$.

Using the notation of the Algorithm A.1, we have $ABS_{r_p}(p,t) = r_p \cdot ABS_1(p,t)$ where ABS_{r_p} is the blocksolver for $r = r_p$ and ABS_1 the blocksolver for $r = 1$. Moreover, we have $f_i(v) = \tilde{a}_i^{\mathrm{T}} v$, where $\tilde{a}_i$ is the i.th row of the matrix $\tilde{A}$ of (2.3).

This allows us to show the theorem by induction. (To simplify notation, we will denote all variables and constants for $r = 1$ and for $r = r_p$ by a corresponding index.) For $s = 0$, we have $v_{r_p}^{(0)} = r_p \cdot v_1^{(0)}$ because of Definition (A.3): the initial solution $v_r^{(0)}$ is a linear combination of the different $ABS_r(e_m, t)$, where $r \in \{1, r_p\}$.

Let us now assume that $v_{r_p}^{(s')} = r_p \cdot v_1^{(s')}$ for all $s' \leq s-1$, in particular $v_{r_p} = v_{r_p}^{(s-1)} = r_p \cdot v_1^{(s-1)} = r_p \cdot v_1$. We have $\theta_{r_p}(f(v_{r_p})) = r_p \cdot \theta_1(f(v_1))$ because the solution $\theta(f(v))$ of (A.4) is unique, and $r_p \cdot \theta_1(f(v_1))$ is a solution to it for $r = r_p$ because of the linearity of $f_i(v)$. Moreover, $\tilde{\theta}_{r_p}(f(v_p)) := r_p \cdot \tilde{\theta}_1(f(v_1))$ is an approximate solution to $\theta_{r_p}(f(v_{r_p}))$ with a relative error in $\mathcal{O}(\frac{\varepsilon^2}{N})$ if $\tilde{\theta}_1(f(v_1))$ is a $\mathcal{O}(\frac{\varepsilon^2}{N})$ approximation to $\theta_1(f(v_1))$, and vice versa. Hence, we can assume that $p_{r_p}(f(v_{r_p})) = p_1(f(v_1))$ according to Definition (A.5). Thus, we have $\hat{v}_{r_p} = ABS_{r_p}(p_{r_p}, t) = ABS_{r_p}(p_1, t) = r_p \cdot ABS_1(p_1, t) = r_p \cdot \hat{v}_1$ as explained above. Moreover, $\nu_{r_p}(v_{r_p}, \hat{v}_{r_p}) = \nu_1(v_1, \hat{v}_1)$ holds according to Definition (A.6). Similarly, the new $v_{r_p} = v_{r_p}^{(s)} := (1 - \tau_{r_p})v_{r_p} + \tau_{r_p}\hat{v}_{r_p}$ and the new $v_1 = v_1^{(s)} := (1 - \tau_1)v_1 + \tau_1\hat{v}_1$ satisfy $v_{r_p}^{(s)} = r_p \cdot v_1^{(s)}$ because we have $\tau_{r_p} = \tau_1$

according to Definition (A.7). Finally, the algorithm will finish for $r = 1$ and $r = r_p$ after the exact same number of iterations because $\nu_{r_p}(v_{r_p}, \hat{v}_{r_p}) = \nu_1(v_1, \hat{v}_1)$ holds. This finishes our proof. □

We finish this section with the proof of Theorem 2.13.

Proof of Theorem 2.13. For simplicity, we write $d_k = d$ and $M_1 = M$. First, we solve (2.2) with $r = 1$ as explained above to obtain a solution $\tilde{\tilde{v}}$ with

$$\begin{aligned} \sum_l c_l \sum_j \tilde{\tilde{v}}_j^{(l)} &= 1 \\ \sum_l \sum_j a(K_j^{(l)}, b_i) \tilde{\tilde{v}}_j^{(l)} &=: \beta_i \geq (1 - \bar{\varepsilon}) \lambda_1 n_i \quad \text{for } i \in \{1, \dots, d\} \ . \end{aligned} \tag{A.8}$$

Here, λ_1 is the optimum of (2.2) for $r = 1$. Note that we have set $\bar{\varepsilon} = \frac{\varepsilon}{4}$.

The max-min resource sharing algorithm needs $\mathcal{O}(d(\log d + \frac{1}{\bar{\varepsilon}^2})) = \mathcal{O}(d(\log d + \frac{1}{\varepsilon^2}))$ iterations, where a call to the blocksolver $UKPIP(d, M, c_{\min}, \frac{\bar{\varepsilon}}{6})$ is needed in each iteration and an overhead of $\mathcal{O}(d \log\log(d\frac{1}{\bar{\varepsilon}})) = \mathcal{O}(d \log\log(d\frac{1}{\varepsilon}))$ incurs (see [33, 41] and Theorem A.2). In total, we get a running time in

$$\mathcal{O}\left(d \left(\log(d) + \frac{1}{\varepsilon^2} \right) \max \left\{ UKPIP\left(d, M, c_{\min}, \frac{\bar{\varepsilon}}{6} \right), \mathcal{O}\left(d \log\log\left(d\frac{1}{\varepsilon} \right) \right) \right\} \right) \ .$$

Note that at most one new configuration is added in every iteration of the max-min resource sharing algorithm. Therefore, there are at most $\mathcal{O}(d(\log d + \frac{1}{\varepsilon^2}))$ columns in $\tilde{A}$, which is also an upper bound on the number of variables $\tilde{\tilde{v}}_j^{(l)} > 0$ of the final solution $\tilde{\tilde{v}}$.

After having determined $\tilde{\tilde{v}}$, we reduce the number of positive variables by finding a basic solution $\tilde{v}$ to (A.8) with only $d + 1$ variables $\tilde{v}_j^{(l)} > 0$. Beling and Megiddo [4] have found an implementation of the standard technique to do so with a running time in $\mathcal{O}(d^{1.594} d(\log d + \frac{1}{\varepsilon^2})) = \mathcal{O}(d^{2.594}(\log d + \frac{1}{\varepsilon^2}))$, and Ke, Zeng, Han, and Pan [57] have improved it to run in $\mathcal{O}(d^{1.5356} d(\log d + \frac{1}{\varepsilon^2})) = \mathcal{O}(d^{2.5356}(\log d + \frac{1}{\varepsilon^2}))$. Since $f(\tilde{\tilde{v}}) = \tilde{A}\tilde{\tilde{v}} = \tilde{A}\tilde{v} = f(\tilde{v})$, the value $r_0 = \frac{1-\bar{\varepsilon}}{\min\{f_1(\tilde{\tilde{v}}_1), \dots, f_d(\tilde{\tilde{v}}_1)\}}$ remains unchanged by this. Note that $\sum_l c_l \sum_j \tilde{\tilde{v}}_j^{(l)} = \sum_l c_l \sum_j \tilde{v}_j^{(l)} = 1$ still holds, too.

After this reduction, we multiply the $d + 1$ positive variables $\tilde{v}_j^{(l)}$ by $r_0 \cdot (1 + 4\bar{\varepsilon})$ to obtain the solution to (2.2). This can obviously be done in time $\mathcal{O}(d + 1)$. We get the solution v within the running time stated in Theorem 2.13. □

B KPIP: Adding the Small Items Efficiently

The ideas in the following section are taken from Lawler's paper [63].

Remark B.1. The results in this section are also valid for non-integral profits and sizes $p_j, s_j \in \mathbb{R}_{>0}$ as well as knapsacks $c_l \in \mathbb{R}_{>0}$.

First, let $J := \{a_1, \ldots, a_m\}$ be a set of knapsack items. Let $\sigma : \{1, \ldots, m\} \to \{1, \ldots, m\}$ be the permutation that sorts the items according to their efficiency $\frac{p(a)}{s(a)}$ in non-increasing order, i.e. $\frac{p(a_{\sigma(1)})}{s(a_{\sigma(1)})} \geq \ldots \geq \frac{p(a_{\sigma(j)})}{s(a_{\sigma(j)})} \geq \ldots \geq \frac{p(a_{\sigma(m)})}{s(a_{\sigma(m)})}$. (In case of ties, the item with the smaller index can be defined to have the smaller efficiency.) Moreover, let $\tilde{c}$ be a knapsack size. We now want to add the items greedily to knapsack $\tilde{c}$ until the knapsack is full without sorting the items according to their efficiency.

To do so, let $J^{(1)}$ the solution, i.e. the m' items such that $J^{(1)} = \{a_{\sigma(1)}, \ldots, a_{\sigma(m')}\}$ and $s(J^{(1)}) = \sum_{j=1}^{m'} s(a_{\sigma(j)}) \leq \tilde{c} < s(J^{(1)}) + s(a_{\sigma(m'+1)}) = \sum_{j=1}^{m'+1} s(a_{\sigma(j)})$. We will now iteratively construct pairwise disjoint item sets $\tilde{J}_1, \ldots, \tilde{J}_k$ and $\bar{J}_k$ such that

$$\bigcup_{j=1}^{k} \tilde{J}_j \subseteq J^{(1)} \subseteq \bigcup_{j=1}^{k} \tilde{J}_j \cup \bar{J}_k \tag{B.1}$$

until having found one $\bar{k}$ such that $\bigcup_{j=1}^{\bar{k}} \tilde{J}_j = J^{(1)}$ or $J^{(1)} = \bigcup_{j=1}^{\bar{k}} \tilde{J}_j \cup \bar{J}_{\bar{k}}$.

We introduce the notion of the *median set* $\tilde{J}$ of an item set $\bar{J} \subseteq J$: if $|\bar{J}| = r$, then the median set $\tilde{J}$ consists of the first $\lfloor \frac{r}{2} \rfloor + 1$ most efficient items of $\bar{J}$. Thus, the median set consists of the items sorted according to their efficiency up to and including the median. For instance, we have for $\bar{J} = J$ that $\tilde{J} = \{a_{\sigma(1)}, a_{\sigma(2)}, \ldots, a_{\sigma(\lfloor \frac{m}{2} \rfloor + 1)}\}$. Since the median of a set can be found in time and space $\mathcal{O}(|\bar{J}|)$, this is also the running time and space requirement for determining $\tilde{J}$. Note that

$$\frac{p(a_{\sigma(\lfloor \frac{r}{2} \rfloor + 1)})}{s(a_{\sigma(\lfloor \frac{r}{2} \rfloor + 1)})} = \frac{p(a_{\sigma(\lfloor \frac{r}{2} \rfloor + 2)})}{s(a_{\sigma(\lfloor \frac{r}{2} \rfloor + 2)})}$$

may hold, i.e. the median set is not always equal to

$$\left\{ a \in \tilde{J} \mid \frac{p(a)}{s(a)} \geq \frac{p(a_{\sigma(\lfloor \frac{r}{2} \rfloor + 1)})}{s(a_{\sigma(\lfloor \frac{r}{2} \rfloor + 1)})} \right\} .$$

Let us determine $J^{(1)}$. Set $\tilde{J}_1 := \emptyset$ and $\bar{J}_1 := J$. Obviously, $\tilde{J}_1 \subseteq J^{(1)} \subseteq \tilde{J}_1 \cup \bar{J}_1$, i.e. Property (B.1) holds, and the item sets are pairwise disjoint. Now, let $\bigcup_{j=1}^{k} \tilde{J}_j$ and $\bar{J}_k$ be the current sets. Let $\tilde{J}$ be the median set of $\bar{J}_k$. If the items in $\bigcup_{j=1}^{k} \tilde{J}_j \cup \tilde{J}$ fit into $\tilde{c}$, then the optimal set $J^{(1)}$ has to contain $\bigcup_{j=1}^{k} \tilde{J}_j \cup \tilde{J}$ as a subset. We set $\tilde{J}_{k+1} := \tilde{J}$ and $\bar{J}_{k+1} := \bar{J}_k \setminus \tilde{J}$ and replace k by $k+1$. Then Property (B.1) still holds, and the item sets are still pairwise disjoint.

On the other hand, if the items in $\bigcup_{j=1}^{k} \tilde{J}_j \cup \tilde{J}$ do not fit into $\tilde{c}$, then $\bigcup_{j=1}^{k} \tilde{J}_j \cup \tilde{J}$ contains too many items, i.e. $J^{(1)} \subsetneq \bigcup_{j=1}^{k} \tilde{J}_j \cup \tilde{J}$. Thus, we set $\bar{J}_k := \tilde{J}$. Property (B.1) holds, and the item sets are still pairwise disjoint.

The procedure continues until $\bigcup_{j=1}^{k} \tilde{J}_j = J^{(1)}$ or $\bigcup_{j=1}^{k} \tilde{J}_j \cup \bar{J}_k = J^{(1)}$. As the size of the set $\bar{J}_k$ halves in every iteration, we get a total running time in $\mathcal{O}(m + \frac{m}{2} + \frac{m}{4} + \cdots) = \mathcal{O}(m)$. Moreover, we only have to store the current union set $\bigcup_{j=1}^{k} \tilde{J}_j$ as well as the current sets $\bar{J}_k$ and $\tilde{J}$ together with their corresponding total sizes $s(\bigcup_{j=1}^{k} \tilde{J}_j), s(\bar{J}_k)$ and $s(\bar{J}_k)$. Thus, we only need space in $\mathcal{O}(m)$.

Lemma B.2. *Let $J := \{a_1, \ldots, a_m\}$ be a set of knapsack items together with a knapsack size $\tilde{c}$. We can find the greedy solution $J^{(1)}$ for this knapsack size by a median-based divide-and-conquer strategy in time and space $\mathcal{O}(|J|) = \mathcal{O}(m)$.*

Consider now a 0-1 knapsack instance with n items and the knapsack size c, which is solved with Lawler's FPTAS, i.e. the algorithms presented in Subsections 3.5.3 and 3.5.4 for only one knapsack size $C = \{c\}$. We assume that the $F_{m_b}(i)$ are not dominated (otherwise we discard dominated ones as seen in Lemma 3.7), i.e. $F_{m_b}(1) < \ldots < F_{m_b}(i_c)$. Here, i_c is the largest profit we have to consider, where we have $F_{m_b}(i_c) \leq c$.

Take two profits $i < i''$. Assume that the set of the small items $J^{(i)}$ and $J^{(i'')}$ that are added to the remaining knapsack space $c - F_{m_b}(i)$ and $c - F_{m_b}(i'')$ in Step 9 of Algorithm 3.4 (see also (3.10)) are known. Then obviously $J^{(i)} \supseteq J^{(i'')}$ holds because $c - F_{m_b}(i) > c - F_{m_b}(i'')$. Furthermore, we have by definition $\phi(c - F_{m_b}(i)) = p(J^{(i)})$ and $\phi(c - F_{m_b}(i'')) = p(J^{(i'')})$. Let us take now a profit i' with $i < i' < i''$ and therefore with $F_{m_b}(i) < F_{m_b}(i') < F_{m_b}(i'')$. We know that $J^{(i)} \supseteq J^{(i')} \supseteq J^{(i'')}$, i.e. we have $J^{(i')} = J^{(i'')} \cup \tilde{J}^{(i')}$ for the appropriate item set $\tilde{J}^{(i')} \subseteq \tilde{J}^{(i)} := J^{(i)} \setminus J^{(i'')}$. Thus, $\tilde{J}^{(i')}$ are the items that can greedily be added to $c - F_{m_b}(i')$ together with the items $J^{(i'')}$.

Therefore, it is sufficient to find the item set $\tilde{J}^{(i')}$, and we can do so by applying Lemma B.2 to the item set $\tilde{J}^{(i)}$ and knapsack size $\tilde{c} = c - F_{m_b}(i') - s(J^{(i'')})$.

Lemma B.3. *Let $i < i' < i''$ with $F_{m_b}(i) < F_{m_b}(i') < F_{m_b}(i'')$. We can determine the set $\tilde{J}^{(i')}$ and therefore $J^{(i')} = J^{(i'')} \cup \tilde{J}^{(i')}$ in time and space $\mathcal{O}(|\tilde{J}^{(i)}|) = \mathcal{O}(|J^{(i)} \setminus J^{(i'')}|)$. We only have to know the item set $\tilde{J}^{(i)} = J^{(i)} \setminus J^{(i'')}$ and the size $s(J^{(i'')})$ to find $\tilde{J}^{(i')}$ itself.*

We can now determine all $\phi(c - F_{m_b}(i))$ again by a divide-and-conquer strategy. Let J be the set of all small items for c with $|J| =: m \leq n$, let i_c be the largest profit i we have

to consider, and let s be the number of the current iteration of the divide-and-conquer algorithm. We start with $s = 0$. We take the profit $\tilde{i} := \lfloor \frac{i_c}{2} \rfloor + 1$, i.e. the median of $1, \ldots, i_c$, and apply Lemma B.2 to it with $\tilde{c} = c - F_{m_b}(\tilde{i})$. We find $J^{(\tilde{i})} = \tilde{J}^{(\tilde{i})}$ in time and space $\mathcal{O}(|J|) \subseteq \mathcal{O}(n)$. We save $\tilde{J}^{(\tilde{i})}$ and $s(J^{(\tilde{i})})$. The missing sets $J^{(i)}$ will now be iteratively constructed.

Let s be the current iteration and $r = r_s \in \mathbb{N}$ be a value such that $i_1 < i_3 < \ldots < i_{2r-1} < i_{2r+1}$ are the profits for which the item sets $\tilde{J}^{(i)}$ have been determined and saved so far. Moreover, we have also stored the corresponding sizes $s(J^{(i)})$. Note that $\tilde{J}^{(i_{2r+1})} = J^{(i_{2r+1})}$ and that $J^{(i)} = \tilde{J}^{(i)} \cup \ldots \cup \tilde{J}^{(i_{2r+1})}$. We take the median $i_{2r'}$ of every pair $i_{2r'-1}$ and $i_{2r'+1}$ as well as the median i_0 of 1 and i_1 and the median i_{2r+2} of i_{2r+1} and i_c. This can be done in time and space $\mathcal{O}(r)$. Thus, we have

$$1 \leq i_0 < i_1 < i_2 < \ldots < i_{2r-1} < i_{2r} < i_{2r+1} < i_{2r+2} \leq i_c \ .$$

Note that $i_{2r'-1}$ and $i_{2r'+1}$ may be neighbours so that $i_{2r'}$ does not exist. If $i_1 = 2$, we set $i_0 = 1$, and if $i_{2r+1} = i_c - 1$, we also set $i_{2r+2} = i_c$.

For every new $i_{2r'}$, we have $F_{m_b}(i_{2r'-1}) < F_{m_b}(i_{2r'}) < F_{m_b}(i_{2r'+1})$. We can determine $\tilde{J}^{(i_{2r'})}$ as the subset of $\tilde{J}^{(i_{2r'-1})}$ with Lemma B.3 in time and space $\mathcal{O}(|\tilde{J}^{(i_{2r'-1})}|) = \mathcal{O}(|J^{(i_{2r'-1})} \setminus J^{(i_{2r'+1})}|)$.
We then save $\tilde{J}^{(i_{2r'})}$ and the total size $s(J^{(i_{2r'})}) = s(\tilde{J}^{(i_{2r'})}) + s(J^{(i_{2r'+1})})$. Moreover, we replace $\tilde{J}^{(i_{2r'-1})}$ by $\tilde{J}^{(i_{2r'-1})} \setminus \tilde{J}^{(i_{2r'})}$ so that we now have $J^{(i_{2r'-1})} = \tilde{J}^{(i_{2r'-1})} \cup J^{(i_{2r'})}$. These operations additionally need time and space in $\mathcal{O}(|J^{(i_{2r'-1})} \setminus J^{(i_{2r'+1})}|)$.
One special case is i_0, where we have $J \supseteq J^{(i_0)} \supseteq J^{(i_1)}$ so that we use the set $J \setminus J^{(i_1)}$ for the algorithm of Lemma B.3, and all operations above need time and space in $\mathcal{O}(|J \setminus J^{(i_1)}|)$. Another special case is i_{2r+2}, where $\tilde{J}^{(i_{2r+1})} = J^{(i_{2r+1})} \supseteq J^{(i_{2r+2})}$ holds so that we apply the algorithm of Lemma B.3 to the set $J^{(i_{2r+1})}$: we need time and space in $\mathcal{O}(|J^{(i_{2r+1})}|)$.

It is of course possible that e.g. already $J^{(i_{2r'-1})} = J^{(i_{2r'+1})}$ holds so that we get $\tilde{J}^{(i_{2r'})} = \emptyset$ and a "running time" of $\mathcal{O}(|J^{(i_{2r'-1})}|) = 0$. To take into account the running time and space complexity of such cases—we still have e.g. to save the information that $\tilde{J}^{(i_{2r'})} = \emptyset$—we additionally add $\mathcal{O}(1)$ to the time and space needed for each $i_{2r'}$, which yields an additional cost of $\mathcal{O}(r)$ in time and space for iteration s.

To sum up, we need time and space in

$$\begin{aligned}&\mathcal{O}\Big(\Big|J \setminus J^{(i_1)}\Big|\Big) + \sum_{r'=1}^{r} \mathcal{O}\Big(\Big|J^{(i_{2r'-1})} \setminus J^{(i_{2r'+1})}\Big|\Big) + \mathcal{O}\Big(\Big|J^{(i_{2r+1})}\Big|\Big) + \mathcal{O}(r)\\ &\quad = \mathcal{O}(|J| + r) = \mathcal{O}(m + r_s) \subseteq \mathcal{O}(n + r_s)\end{aligned}$$

for iteration s because $J^{(i_{2r+1})}$, the $J^{(i_{2r'-1})} \setminus J^{(i_{2r'+1})}$, and $J \setminus J^{(i_1)}$ are disjoint sets. Obviously, we only need $\mathcal{O}(\log i_c)$ iterations in total to find the values for all $F_{m_b}(i)$. It is also easy to see that $r_s \in \mathcal{O}(2^s)$.

Let us assume that we have $c = c_l$ for one $c_l \in C_b$ of our 0-1 KPIP algorithm. For the overall time bound, we have $i_c = i_{c_l} \leq i_{\max} \in \mathcal{O}(\frac{1}{\varepsilon^2} c^{(b)}_{\max}/c^{(b)}_{\min})$ (see (3.23)) and therefore together with $r_s \in \mathcal{O}(2^s)$ a total running time in

$$\begin{aligned}\mathcal{O}\left(\sum_{s=0}^{\mathcal{O}(\log(i_{\max}))} (n + r_s)\right) &= \mathcal{O}\left(n \cdot \log\left(\frac{1}{\varepsilon^2}\frac{c^{(b)}_{\max}}{c^{(b)}_{\min}}\right) + \sum_{s=0}^{\mathcal{O}(\log(i_{\max}))} 2^s\right) \\ &= \mathcal{O}\left(n \cdot \log\left(\frac{1}{\varepsilon^2}\frac{c^{(b)}_{\max}}{c^{(b)}_{\min}}\right) + \frac{1}{\varepsilon^2}\frac{c^{(b)}_{\max}}{c^{(b)}_{\min}}\right) \quad .\end{aligned}$$

For the overall space requirement, note that we only save the $\tilde{J}^{(i)}$ and $s(J^{(i)})$ after iteration s is completed, which need space in $\mathcal{O}(r_s + n) \subseteq \mathcal{O}(i_{\max} + n)$ in total. During one operation s, we also need space in $\mathcal{O}(n + r_s) \subseteq \mathcal{O}(i_{\max} + n)$. As the values and sets of iteration $s - 1$ can be discarded when iteration s has been finished, the overall space complexity is in

$$\mathcal{O}(i_{\max} + n) \subseteq \mathcal{O}\left(\frac{1}{\varepsilon^2}\frac{c^{(b)}_{\max}}{c^{(b)}_{\min}} + n\right) \quad .$$

The actual values $\phi_l(c_l - F_{m_b}(i))$ can be determined by using that

$$\begin{aligned}\phi_l\left(c_l - F_{m_b}(i_{c_l})\right) &= \sum_{a \in \tilde{J}^{(i_{c_l})}} p(a), \\ \phi_l\left(c_l - F_{m_b}(i_{c_l} - 1)\right) &= \phi_l\left(c_l - F_{m_b}(i_{c_l})\right) + \sum_{a \in \tilde{J}^{(i_{c_l}-1)}} p(a) \\ &\vdots\end{aligned} \tag{B.2}$$

This needs time and space in $\mathcal{O}(i_{\max} + n)$, which is dominated by the expressions above.

We have proved Theorem 3.16. It is possible to heuristically improve the running time: let J' be the first m' most efficient small items so that $\sum_{r'=1}^{m'} s(a_{\sigma(r')}) \leq c_l$, but $\sum_{r'=1}^{m'+1} s(a_{\sigma(r')}) > c_l$. By a binary search as above, J' can be found in $\mathcal{O}(m) \subseteq \mathcal{O}(n)$ and then J' instead of J used to determine all $\phi_l(\cdot)$ and $\tilde{J}^{(i)}$.

C The Bounded KPIP: The Details

Proof of Theorem 3.29. Here, we normally use multi-sets, i.e. we do not save individual item copies, but how many copies of one item type we use. We have for example $\tilde{J}^{(i)} = \{a_{\sigma(j)} : g_j, \ldots, a_{\sigma(j')} : g_{j'}\}$, where $g_{j''}$ denotes the number of copies of item $a_{\sigma(j'')}$ in the set $\tilde{J}^{(i)}$. Note that the usual algorithm to find the median of a set can be adapted to be still linear in the number of different item types in a set. For convenience, $|J|$ denotes the number of item types in set J.

First, the median-based search does not change much. It is used to get the $\bar{P}_{c_l}$ and P_0 in Step 1 of Algorithm 3.4, the small-item values $\phi_l(c_l - F_{m_b}(i))$ in Step 9 and the small items of V_t in Step 12, .

Let $F_{m_b}(i)$ and $F_{m_b}(i'')$ be two values as seen in Appendix B with their sets $\tilde{J}^{(i)}, J^{(i)}$ and $\tilde{J}^{(i'')}, J^{(i'')}$, respectively. We may have e.g. $\tilde{J}^{(i)} = \{a_{\sigma(j)} : g_j, \ldots, a_{\sigma(j')} : g_{j'}\}$ and $J^{(i'')} = \{a_{\sigma(j')} : \tilde{g}_{j'}, \ldots, a_{\sigma(j'')} : \tilde{g}_{j''}\}$, i.e. both contain item type $a_{\sigma(j')}$, each with its corresponding multiple. In iteration s of the algorithm of Theorem 3.16, the sets $\tilde{J}^{(i_0)}, \tilde{J}^{(i_1)}, \ldots, \tilde{J}^{(i_{2r_s+2})}$ may therefore no longer be "disjoint" by sharing copies of the same item type. However, we have by definition of the sets $\tilde{J}^{(\tilde{i})}$ that $\tilde{J}^{(i_{2r_s+2})}$ contains the most efficient items and that the efficiencies do not increase from one item set to another: the items are added according to the order σ from the most to the least efficient items. Thus, the number of overlapping item types in iteration s is bounded by $\mathcal{O}(r_s)$. A similar reasoning shows that the time and space complexity in Lemma B.2 and Lemma B.3 do not change either. Hence, determining $\tilde{J}^{(i')}$ for $i < i' < i''$ can still be done linear in the number of item types in $\tilde{J}^{(i)}$. To sum up, the algorithm to find all $\tilde{J}^{(i)}$ works as before and its asymptotic running time and space complexity do not increase. In the same way, we can show that the computation of the values $\phi_l(c_l - F_{m_b}(i))$ as seen in (B.2) does not need more time and space.

The same properties also hold for the operations in Step 1 so that the overall running time and space complexity of this step do not change either.

For the large-item computation in Step 7, we create item copies in Step 6. Fix one C_b. Let $\tilde{a}_{1,j}, \ldots, \tilde{a}_{t,j}$ be the large item types that have the scaled profit q_j, sorted according to their size in non-decreasing order (i.e. $s(\tilde{a}_{1,j}) \leq \ldots \leq s(\tilde{a}_{t,j})$), and let $\tilde{d}_{1,j}, \ldots, \tilde{d}_{t,j}$ be their corresponding maximal possible multiple. Since a solution can contain at most $n_{L,j}$ large items for every q_j (see Bound (3.28)), it is sufficient for the large-item

computation to take copies of the items of smallest size until having $n_{L,j}$ of them. Hence, it is sufficient to find $t' \leq t$ such that

$$\tilde{d}_{1,j} + \cdots + \tilde{d}_{t'-1,j} + \underbrace{(\tilde{d}_{t',j} - h_j)}_{:=\tilde{\tilde{d}}_{t',j}} = n_{L,j}, \quad h_j \in \{0, \ldots, \tilde{d}_{t',j} - 1\} \quad ,$$

and then take $\tilde{d}_{t'',j}$ copies of $\tilde{a}_{t'',j}$, $t'' \in \{1, \ldots, t'-1\}$, and $\tilde{\tilde{d}}_{t',j}$ copies of $\tilde{a}_{t',j}$. This method already yields the bound $n_b \leq n_L \in \mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b)} / c_{\min}^{(b)})$ for the large items as seen in Lemma 3.20.

To improve the running time, Lawler's idea for the unbounded case can be slightly modified as proposed by Plotkin et al. [72, p. 296]: item copies $\tilde{a}_{t'',j}^{(r)}$ are created with

$$\begin{aligned}
& \tilde{a}_{t'',j}^{(r)} \text{ with } p(\tilde{a}_{t'',j}^{(r)}) = 2^r p(\tilde{a}_{t'',j}), s(\tilde{a}_{t'',j}^{(r)}) = 2^r s(\tilde{a}_{t'',j}) \\
& \text{for } r \in \{0, \ldots, \lfloor \log_2(\tilde{d}_{t'',j}) \rfloor - 1\} \\
\text{and} \quad & \tilde{a}_{t'',j}^{(r)} \text{ with } p(\tilde{a}_{t'',j}^{(r)}) = (\tilde{d}_{t'',j} - 2^{\lfloor \log_2(\tilde{d}_{t'',j}) \rfloor} + 1) p(\tilde{a}_{t'',j}), \\
& s(\tilde{a}_{t'',j}^{(r)}) = (\tilde{d}_{t'',j} - 2^{\lfloor \log_2(\tilde{d}_{t'',j}) \rfloor} + 1) s(\tilde{a}_{t'',j}) \\
& \text{for } r = \lfloor \log_2(\tilde{d}_{t'',j}) \rfloor \quad .
\end{aligned}$$

The only exception are the item copies $\tilde{a}_{t',j}^{(r)}$ of $\tilde{a}_{t',j}$ where $\tilde{d}_{t'',j}$ is replaced by $\tilde{\tilde{d}}_{t',j} = \tilde{d}_{t',j} - h_j$. These multiple copies are sufficient to represent all choices of item copies. Similar to the unbounded case, only $n_{L,j}$ item copies of smallest size for one profit q_j have to be kept after having created the copies. It is however not possible to keep only the item $\tilde{a}_{t'',j}^{(r)}$ of smallest size because Lemma 3.25 is not valid for BKPIP.

Note that keeping the $n_{L,j}$ smallest items for every profit q_j can still be done in time linear in the number of item types: the median finding algorithm used for it (see Lemma 3.20) still runs in linear time.

For the entire BKPIP algorithm, the running time and space bound can be determined like for the improved 0-1 KPIP algorithm in Lemma 3.22. The additional time to take the item copies in Step 6 and to keep only the smallest $n_{L,j}$ item copies for every q_j is dominated by the large-item computation in Step 7. In Step 12, the (normal) set of large items found by backtracking and the (multi-)set $J^{(i)}$ of small items additionally have to be combined into the multi-set V_t of the solution. This can be done in time $\mathcal{O}(\frac{1}{\varepsilon^2} c_{\max}^{(b_0)} / c_{\min}^{(b_0)} + n)$ and space $\mathcal{O}(n)$, which does not change the running time of the inner for-loop (Steps 8–12; see also Lemma 3.17, which is still valid for the improved FPTAS as seen in the proof of Lemma 3.22). □

Acronyms

PTAS	Polynomial Time Approximation Scheme
APTAS	Asymptotic Polynomial Time Approximation Scheme
EPTAS	Efficient Polynomial Time Approximation Scheme
FPTAS	Fully Polynomial Time Approximation Scheme
AFPTAS	Asymptotic Fully Polynomial Time Approximation Scheme
KP	Knapsack Problem
UKP	Unbounded Knapsack Problem
BKP	Bounded Knapsack Problem
KPIP	Knapsack Problem with Inversely Proportional Profits
UKPIP	Unbounded Knapsack Problem with Inversely Proportional Profits
BKPIP	Bounded Knapsack Problem with Inversely Proportional Profits
BP	Bin Packing Problem
VBP	Variable-sized Bin Packing Problem
MLCSP	Multiple-Length Cutting Stock Problem
CSP	Cutting Stock Problem
SP	Strip Packing
LP	linear program
ILP	integer linear program
$P\,\vert\vert\,C_{max}$	Scheduling on Identical Machines
$Pm\,\vert\vert\,C_{max}$	Scheduling on m Identical Machines (m is constant)
$Q\,\vert\vert\,C_{max}$	Scheduling on Uniform Machines

$\mathrm{R}\,\|\|\,C_{\max}$	Scheduling on Unrelated Machines
$\mathrm{R}m\,\|\|\,C_{\max}$	Scheduling on m Unrelated Machines (m is constant)
$(\mathrm{P}m_1,\ldots,\mathrm{P}m_K)\|\|\,C_{\max}$	Scheduling on Unrelated Machines of Few Different Types (K is constant)
$\mathrm{OPT}(I)$	The optimal value for instance I
$\mathrm{OPT}(I,C)$	For VBP: the optimal value for the instance (I,C) with items I and bin sizes C
$\mathrm{OPT}_{c_l}(I)$	For KPIP: the optimal value for instance I, knapsack size c_l, and profits $\frac{p_j}{c_l}$
$\overline{\mathrm{OPT}}_{c_l}(I)$	For KPIP: the optimal value for instance I, knapsack size c_l, and basic profits p_j
$\mathrm{OPT}(I,v)$	For UKP: the optimal value for instance I, knapsack volume v, and profits p_j. For KPIP: the optimal value for instance I, knapsack volume v, and profits $\frac{p_j}{c_l}$
$\overline{\mathrm{OPT}}(I,v)$	For KPIP: the optimal value for instance I, knapsack volume v, and profits p_j
$\mathrm{OPT}_{\leq k_0}(I,v)$	For UKP: the value of the optimal structured solution to instance I for $k=k_0$, knapsack volume v, and profits p_j (see Definition 4.11)
$\mathrm{OPT}_{\mathrm{St}}(I,v)$	For UKP: the value of the optimal structured solution to instance I with a lower bound for knapsack volume v and profits p_j (see Definition 4.16)
$\overline{\mathrm{OPT}}_{\leq k_0}(I,v)$	For UKPIP: the value of the optimal structured solution to instance I for $k=k_0$, knapsack volume v, and basic profits p_j (see Definition 5.19)
$\overline{\mathrm{OPT}}_{\mathrm{St}}(I,v)$	For UKPIP: the value of an optimal structured solution to instance I with a lower bound for knapsack volume v and profits p_j (see Definition 5.28)
$\mathrm{LIN}(I)$	The optimal value for an LP, which is sometimes the relaxation of an ILP in this thesis
$\mathrm{Area}(I)$	For VBP: The total size of the items in I

Bibliography

[1] M. Aizatulin, F. Diedrich, and K. Jansen. "Experimental Results in Approximation of Max-Min Resource Sharing". Unpublished manuscript.

[2] D. Arad, Y. Mordechai, and H. Shachnai. *Tighter Bounds for Makespan Minimization on Unrelated Machines*. 2014. arXiv: 1405.2530.

[3] S. K. Baruah. "Task assignment on two unrelated types of processors". In: *Proceedings of the 19th International Conference on Real-Time and Network Systems, RTNS '11*. Ed. by S. Faucou, A. Burns, and L. George. 2011, pp. 205–214.

[4] P. A. Beling and N. Megiddo. "Using fast matrix multiplication to find basic solutions". In: *Theoretical Computer Science* 205.1–2 (1998), pp. 307–316.

[5] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[6] A. Bhaskara, R. Krishnaswamy, K. Talwar, and U. Wieder. "Minimum Makespan Scheduling with Low Rank Processing Times". In: *Proceedings of the Twenty-Forth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*. Ed. by S. Khanna. SIAM, 2013, pp. 937–947.

[7] R. Bleuse, S. Kedad-Sidhoum, F. Monna, G. Mounié, and D. Trystram. "Scheduling independent tasks on multi-cores with GPU accelerators". In: *Concurrency and Computation: Practice and Experience* 27.6 (2015), pp. 1625–1638. DOI: 10.1002/cpe.3359.

[8] V. Bonifaci and A. Wiese. *Scheduling Unrelated Machines of Few Different Types*. 2012. arXiv: 1205.0974.

[9] M. Bougeret, P.-F. Dutot, K. Jansen, C. Robenek, and D. Trystram. "Approximation Algorithms for Multiple Strip Packing and Scheduling Parallel Jobs in Platforms". In: *Discrete Mathematics, Algorithms and Applications* 3.4 (2011), pp. 553–586.

[10] D. Chakrabarty, S. Khanna, and S. Li. "On $(1, \varepsilon)$-Restricted Assignment Makespan Minimization". In: *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*. Ed. by P. Indyk. SIAM, July 2015, pp. 1087–1101. arXiv: 1410.7506.

[11] L. Chen, D. Ye, and G. Zhang. "An improved lower bound for rank four scheduling". In: *Operations Research Letters* 42.5 (2014), pp. 348–350.

[12] E. G. Coffman Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan. "Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms". In: *SIAM Journal on Computing* 9.4 (1980), pp. 808–826.

[13] G. Desaulniers, J. Desrosiers, and M. M. Solomon, eds. *Column Generation*. Springer Science & Business Media, 2005.

[14] F. Diedrich. "Approximation Algorithms for Linear Programs and Geometrically Constrained Packing Problems". PhD thesis. Christian-Albrechts-Universität zu Kiel, 2009.

[15] G. Dósa. "The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is $FFD(I) \leq \frac{11}{9}OPT(I) + \frac{6}{9}$". In: *Proceedings of the First International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, ESCAPE 2007*. Ed. by B. Chen, M. Paterson, and G. Zhang. Vol. 4614. LNCS. Springer, Heidelberg, 2007, pp. 1–11.

[16] G. Dósa, R. Li, X. Han, and Z. Tuza. "Tight absolute bound for First Fit Decreasing bin-packing: $FFD(L) \leq \frac{11}{9}OPT(L) + \frac{6}{9}$". In: *Theoretical Computer Science* 510 (2013), pp. 13–61.

[17] K. Eisemann. "The Trim Problem". In: *Management Science* 3.3 (1957), pp. 279–284.

[18] M. Elberfeld, A. Jakoby, and T. Tantau. *Logspace Versions of the Theorems of Bodlaender and Courcelle*. Tech. rep. 62. Electronic Colloquium on Computational Complexity (ECCC), 2010. First published in: *Proceedings of the 51th Annual Symposium on Foundations of Computer Science, FOCS 2010*. IEEE Computer Society, 2010, pp. 143–152.

[19] L. Epstein, C. Imreh, and A. Levin. "Class constrained bin packing revisited". In: *Theoretical Computer Science* 411.34–36 (July 2010), pp. 3073–3089.

[20] W. Fernandez de la Vega and G. S. Lueker. "Bin packing can be solved within $1 + \varepsilon$ in linear time". In: *Combinatorica* 1.4 (1981), pp. 349–355.

[21] A. V. Fishkin, K. Jansen, and M. Mastrolilli. "Grouping Techniques for Scheduling Problems: Simpler and Faster". In: *Algorithmica* 51.2 (2008), pp. 183–199.

[22] D. K. Friesen and M. A. Langston. "Variable Sized Bin Packing". In: *SIAM Journal on Computing* 15.1 (Feb. 1986), pp. 222–230.

[23] M. Gairing, B. Monien, and A. Woclaw. "A faster combinatorial approximation algorithm for scheduling unrelated parallel machines". In: *Theoretical Computer Science* 380.1–2 (2007), pp. 87–99.

[24] A. Gál, J. Jang, N. Limaye, M. Mahajan, and K. Sreenivasaiah. *Space-Efficient Approximations for Subset Sum*. Tech. rep. 180. Electronic Colloquium on Computational Complexity (ECCC), 2014.

[25] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[26] J. C. Gehrke, K. Jansen, S. E. J. Kraft, and J. Schikowski. *A PTAS for Scheduling Unrelated Machines of Few Different Types*. Tech. rep. 1506. Christian-Albrechts-Universität zu Kiel, 2015. ISSN: 2192-6247.

[27] J. C. Gehrke, K. Jansen, S. E. J. Kraft, and J. Schikowski. "A PTAS for Scheduling Unrelated Machines of Few Different Types". In: *Proceedings of 42nd International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2016*. Ed. by R. Freivalds, G. Engels, and B. Catania. Vol. 9587. LNCS: Advanced Research in Computing and Software Science. Springer, 2016. In press.

[28] J. C. Gehrke and J. Schikowski. *Scheduling auf unabhängigen Maschinen einiger weniger Typen*. Report. Christian-Albrechts-Universität zu Kiel, 2014.

[29] P. C. Gilmore and R. E. Gomory. "A Linear Programming Approach to the Cutting-Stock Problem". In: *Operations Research* 9.6 (1961), pp. 849–859.

[30] P. C. Gilmore and R. E. Gomory. "A Linear Programming Approach to the Cutting Stock Problem—Part II". In: *Operations Research* 11.6 (1963), pp. 863–888.

[31] M. X. Goemans and T. Rothvoß. "Polynomiality for Bin Packing with a Constant Number of Item Types". In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*. Ed. by C. Chekuri. 2014, pp. 830–839. arXiv: 1307.5108.

[32] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey". In: *Discrete Optimization II. Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium*. Ed. by P. L. Hammer, E. L. Johnson, and B. H. Korte. Vol. 5. Annals of Discrete Mathematics. 1979, pp. 287–326. DOI: 10.1016/S0167-5060(08)70356-X.

[33] M. D. Grigoriadis, L. G. Khachiyan, L. Porkolab, and J. Villavicencio. "Approximate Max-Min Resource Sharing for Structured Concave Optimization". In: *SIAM Journal on Optimization* 11.4 (2001), pp. 1081–1091.

[34] R. Harren, K. Jansen, L. Prädel, and R. van Stee. "A $(\frac{5}{3} + \varepsilon)$-approximation for strip packing". In: *Computational Geometry: Theory and Applications* 47.2 (2014), pp. 248–267.

[35] R. Hoberg and T. Rothvoß. *A Logarithmic Additive Integrality Gap for Bin Packing*. 2015. arXiv: 1503.08796.

[36] D. S. Hochbaum and D. B. Shmoys. "Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results". In: *Journal of the ACM* 34.1 (1987), pp. 144–162.

[37] D. S. Hochbaum and D. B. Shmoys. "A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach". In: *SIAM Journal on Computing* 17.3 (June 1988), pp. 539–551.

[38] E. Horowitz and S. Sahni. "Exact and Approximate Algorithms for Scheduling Nonidentical Processors". In: *Journal of the ACM* 23.2 (1976), pp. 317–327. DOI: 10.1145/321941.321951.

[39] O. H. Ibarra and C. E. Kim. "Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems". In: *Journal of the ACM* 22.4 (1975), pp. 463–468.

[40] C. Imreh. "Scheduling Problems on Two Sets of Identical Machines". In: *Computing* 70.4 (2003), pp. 277–294.

[41] K. Jansen. "Approximation Algorithms for Min-Max and Max-Min Resource Sharing Problems, and Applications". In: *Efficient approximation and online algorithms*. Ed. by E. Bampis, K. Jansen, and C. Kenyon. Vol. 3484. LNCS. Springer, 2006, pp. 156–202.

[42] K. Jansen and S. Kraft. "An Improved Approximation Scheme for Variable-Sized Bin Packing". In: *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science, MFCS 2012*. Ed. by B. Rovan, V. Sassone, and P. Widmayer. Vol. 7464. LNCS. Springer, 2012, pp. 529–541. DOI: 10.1007/978-3-642-32589-2_47.

[43] K. Jansen and S. Kraft. "An Improved Knapsack Solver for Column Generation". In: *Proceedings of the 8th International Computer Science Symposium in Russia, CSR 2013*. Ed. by A. Bulatov and A. Shur. Vol. 7913. LNCS. Springer, 2013, pp. 12–23. DOI: 10.1007/978-3-642-38536-0_2.

[44] K. Jansen and S. Kraft. "An Improved Approximation Scheme for Variable-Sized Bin Packing". In: *Theory of Computing Systems* (2015), pp. 1–61. DOI: 10.1007/s00224-015-9644-2. Pre-published.

[45] K. Jansen and S. E. J. Kraft. *A Faster FPTAS for the Unbounded Knapsack Problem*. 2014. arXiv: 1504.04650.

[46] K. Jansen and S. E. J. Kraft. "A Faster FPTAS for the Unbounded Knapsack Problem with Inversely Proportional Profits". 2015. Unpublished manuscript.

[47] K. Jansen and S. E. J. Kraft. "A Faster FPTAS for the Unbounded Knapsack Problem". In: *Proceedings of the 26th International Workshop on Combinatorial Algorithms, IWOCA 2015*. Ed. by Z. Lipták and B. Smyth. LNCS. Springer, 2015/2016. Forthcoming.

[48] K. Jansen and M. Margraf. *Approximative Algorithmen und Nichtapproximierbarkeit*. De Gruyter, 2008.

[49] K. Jansen and M. Mastrolilli. *Scheduling unrelated parallel machines: linear programming strikes back*. Tech. rep. 1004. Christian-Albrechts-Universität zu Kiel, Mar. 2010. ISSN: 2192-6247.

[50] K. Jansen and L. Porkolab. "Improved Approximation Schemes for Scheduling Unrelated Parallel Machines". In: *Mathematics of Operations Research* 26.2 (2001), pp. 324–338.

[51] K. Jansen and C. Robenek. "Scheduling Jobs on Identical and Uniform Processors Revisited". In: *Proceedings of the 9th Workshop on Approximation and Online Algorithms, WAOA 2011*. LNCS 7164. 2012, pp. 109–122.

[52] K. Jansen and R. Solis-Oba. "An asymptotic fully polynomial time approximation scheme for bin covering". In: *Theoretical Computer Science* 306.1–3 (2003), pp. 543–551.

[53] K. Jansen and R. Solis-Oba. "Rectangle packing with one-dimensional resource augmentation". In: *Discrete Optimization* 6.3 (2009), pp. 310–323.

[54] D. M. Kane. *Unary Subset-Sum is in Logspace*. 2010. arXiv: 1012.1336.

[55] L. V. Kantorovich. "Mathematical Methods of Organizing and Planning Production". In: *Management Science* 6.4 (July 1960), pp. 366–422. Significantly enlarged and translated record of a report given in 1939.

[56] N. Karmarkar and R. M. Karp. "An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem". In: *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, FOCS 1982*. IEEE Computer Society, 1982, pp. 312–320.

[57] S. Ke, B. Zeng, W. Han, and V. Y. Pan. "Fast rectangular matrix multiplication and some applications". In: *Science in China Series A: Mathematics* 51.3 (2008), pp. 389–406.

[58] H. Kellerer, R. Mansini, U. Pferschy, and M. G. Speranza. "An efficient fully polynomial approximation scheme for the Subset-Sum Problem". In: *Journal of Computer and System Sciences* 66.2 (2003), pp. 349–370.

[59] H. Kellerer and U. Pferschy. "A New Fully Polynomial Time Approximation Scheme for the Knapsack Problem". In: *Journal of Combinatorial Optimization* 3.1 (1999), pp. 59–71.

[60] H. Kellerer and U. Pferschy. "Improved Dynamic Programming in Connection with an FTPAS for the Knapsack Problem". In: *Journal of Combinatorial Optimization* 8.1 (2004), pp. 5–11.

[61] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[62] C. Kenyon and E. Rémila. "A Near-Optimal Solution to a Two-Dimensional Cutting Stock Problem". In: *Mathematics of Operations Research* 25.4 (2000), pp. 645–656. First published as "Approximate Strip Packing". In: *Proceedings of the 37th Annual Symposium on Foundations of Computer Science, FOCS 1996*. IEEE Computer Society, 1996, pp. 31–36.

[63] E. L. Lawler. "Fast Approximation Algorithms for Knapsack Problems". In: *Mathematics of Operations Research* 4.4 (1979), pp. 339–356.

[64] J. K. Lenstra, D. B. Shmoys, and É. Tardos. "Approximation Algorithms for Scheduling Unrelated Parallel Machines". In: *Mathematical Programming* 46 (1990), pp. 259–271.

[65] D. Lokshtanov and J. Nederlof. "Saving space by algebraization". In: *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*. Ed. by L. J. Schulman. ACM, 2010, pp. 321–330.

[66] M. E. Lübbecke and J. Desrosiers. "Selected Topics in Column Generation". In: *Operations Research* 53.6 (2005), pp. 1007–1023. DOI: 10.1287/opre.1050.0234.

[67] M. J. Magazine and O. Oguz. "A fully polynomial approximation algorithm for the 0-1 knapsack problem". In: *European Journal of Operational Research* 8.3 (Nov. 1981), pp. 270–273.

[68] S. Martello and P. Toth. *Knapsack Problems. Algorithms and Computer Implementations*. John Wiley & Sons, 1990. Freely available on Martello's website http://www.or.deis.unibo.it/staff_pages/martello/cvitae.html.

[69] F. D. Murgolo. "An Efficient Approximation Scheme for Variable-Sized Bin Packing". In: *SIAM Journal on Computing* 16.1 (1987), pp. 149–161.

[70] G. L. Nemhauser. "Column generation for linear and integer programming". In: *Documenta Mathematica* (2012): *Optimization Stories*, pp. 65–73. URL: https://www.math.uni-bielefeld.de/documenta/vol-ismp/vol-ismp.html.

[71] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[72] S. A. Plotkin, D. B. Shmoys, and É. Tardos. "Fast Approximation Algorithms for Fractional Packing and Covering Problems". In: *Mathematics of Operations Research* 20.2 (1995), pp. 257–301. First published in: *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, FOCS 1991*. IEEE Computer Society, 1991, pp. 495–504.

[73] L. Prädel. "Approximation Algorithms for Geometric Packing Problems". PhD thesis. Christian-Albrechts-Universität zu Kiel, 2012.

[74] G. Raravi and V. Nélis. "A PTAS for Assigning Sporadic Tasks on Two-type Heterogeneous Multiprocessors". In: *Proceedings of the 33rd IEEE Real-Time Systems Symposium, RTSS 2012*. IEEE Computer Society, 2012, pp. 117–126.

[75] T. Rothvoß. "Approximating Bin Packing within $O(\log OPT \cdot \log \log OPT)$ bins". In: *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*. IEEE Computer Society, Chicago, 2013, pp. 20–29. arXiv: 1301.4010.

[76] I. Schiermeyer. "Reverse-Fit: A 2-Optimal Algorithm for Packing Rectangles". In: *Proceedings of the Second Annual European Symposium on Algorithms, ESA 1994*. Ed. by J. van Leeuwen. Vol. 855. LNCS. Springer, 1994, pp. 290–299.

[77] H. Shachnai, T. Tamir, and O. Yehezkely. "Approximation Schemes for Packing with Item Fragmentation". In: *Theory of Computing Systems* 43.1 (2008), pp. 81–98.

[78] H. Shachnai and O. Yehezkely. "Fast Asymptotic FPTAS for Packing Fragmentable Items with Costs". In: *Proceedings of the 16th International Symposium on Fundamentals of Computation Theory, FCT 2007*. Ed. by E. Csuhaj-Varjú and Z. Ésik. Vol. 4639. LNCS. Springer, 2007, pp. 482–493.

[79] E. V. Shchepin and N. Vakhania. "An optimal rounding gives a better approximation for scheduling unrelated machines". In: *Operations Research Letters* 33.2 (2005), pp. 127–133.

[80] G. Shmonin. "Parameterised Integer Programming, Integer Cones, and Related Problems". PhD thesis. Universität Paderborn, June 2007.

[81] D. B. Shmoys and É. Tardos. "An approximation algorithm for the generalized assignment problem". In: *Mathematical Programming* 62.1 (1993), pp. 461–474. DOI: 10.1007/BF01585178.

[82] D. Simchi-Levi. "New worst-case results for the bin-packing problem". In: *Naval Research Logistics* 41.4 (1994), pp. 579–585.

[83] D. D. Sleator. "A 2.5 Times Optimal Algorithm for Packing in Two Dimensions". In: *Information Processing Letters* 10.1 (1980), pp. 37–40.

[84] A. Steinberg. "A Strip-Packing Algorithm with Absolute Performance Bound 2". In: *SIAM Journal on Computing* 26.2 (1997), pp. 401–409.

[85] O. Svensson. "Santa Claus Schedules Jobs on Unrelated Machines". In: *SIAM Journal on Computing* 41.5 (2012), pp. 1318–1341. arXiv: 1011.1168.

[86] M. Sviridenko. "A note on the Kenyon-Remila strip-packing algorithm". In: *Information Processing Letters* 112.1–2 (2012), pp. 10–12.

[87] A. Wiese, V. Bonifaci, and S. K. Baruah. "Partitioned EDF scheduling on a few types of unrelated multiprocessors". In: *Real-Time Systems* 49.2 (2013), pp. 219–238.

Colophon

The typographical look-and-feel of this thesis is based in parts on `classicthesis` developed by André Miede. His style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and mLyX:

`http://code.google.com/p/classicthesis/`

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

`http://postcards.miede.de/`

Since `classicthesis` is released under the GNU General Public License (see `http://www.gnu.org/copyleft/gpl.html`), the look-and-feel of this thesis is also licensed under the GNU General Public License. Please contact the author to receive a copy of the LaTeX code.

Erklärung

Diese Abhandlung ist nach Inhalt und Form meine eigene Arbeit. Ausnahmen sind die Beratung durch meinen Betreuer Prof. Dr. Klaus Jansen sowie teilweise Kapitel 6. Das Ergebnis dieses Kapitels wurde in der Vorlesung „Effiziente Algorithmen“ im Sommersemester 2014 an der Christian-Albrechts-Universität zu Kiel von Jan Clemens Gehrke, Klaus Jansen, Stefan Kraft und Jakob Schikowski entdeckt. Das Kapitel basiert auf dem von Stefan Kraft erstellten technischen Bericht [26], der eine Erweiterung und Überarbeitung des deutschsprachigen Berichts [28] von Jan Clemens Gehrke und Jakob Schikowski ist. Deswegen sind Teile von Kapitel 6 auch eine direkte Übersetzung von [28].

Diese Dissertation wurde weder ganz noch in Teilen an anderer Stelle im Rahmen eines Prüfungsverfahrens vorgelegt. Die Veröffentlichungen und Einreichungen zur Veröffentlichung, auf denen die Dissertation basiert, sind in Abschnitt 1.5 ab Seite 9 aufgeführt.

Diese Arbeit ist unter Einhaltung der Regeln guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft entstanden.

Kiel, im Januar 2016

Stefan Kraft