APPROXIMATION ALGORITHMS FOR 2D PACKING PROBLEMS

Dissertation

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
(Dr. rer. nat.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

Olga Gerber

Kiel 2005

1. Gutachter: Prof. Dr. Klaus Jansen

2. Gutachter: Prof. Dr. Anand Srivastav

Tag der mündlichen Prüfung: 18.10.2005

Zum Druck genehmigt: 18.10.2005

ACKNOWLEDGMENTS

The past three years at the University of Kiel have been a great experience for me, both professionally and socially. I am deeply indebted to my teachers, colleagues and friends for their help and support during this time. First of all I would like to thank my advisor Klaus Jansen for his encouragement and the supervision of my thesis. Klaus not only led me to the field and community of combinatorial optimization and approximation algorithms, but also shared with me a lot of research problems. Only with his help was I able to get my first results, but at the same time I am thankful for the independence he allowed me in my research. I also sincerely appreciate my colleague Aleksei Fishkin, who has devoted many hours of his time introducing me to the field of approximation algorithms during my first year in Kiel. He also introduced me to new problems and shared his deep insights with me. Many hours of discussions with Aleksei have turned into several joint research papers. I am grateful to him for his help, advise and friendship. I wish to thank Klaus and Aleksei for their trust and confidence in my abilities without which this thesis would never have been written.

My research was supported by the Graduiertenkolleg 357 "Effiziente Algorithmen and Mehrskalenmethoden" supported by the DFG (Deutsche Forschungsgemeinschaft). Many thanks are due to Anand Srivastav for his kind help in both my academic activity and the financial support-related matters.

I would especially like to thank Roberto Solis-Oba who not only was very kind in hosting and supporting me at the University of Western Ontario for March 2005, but also was constantly present when I needed his advice. Chapter 2 has gained a lot from his co-authorship in the corresponding papers. I want to thank Roberto for his willingness to take an assessment for this thesis.

I thank all my colleagues in the group "Theory of Parallelism" and staff of the Institute of Theoretical Computer Science and Applied Mathematics for an extraordinarily friendly atmosphere and motivating working environment. I was also happy to meet Jana Chlebiková, Guochuan Zhang, Deshi Ye, Hu Zhang, Marian Margraf, my former colleagues. I will never forget our interesting group activities. I would like to thank Brigitte Scheidemann and Ute Iaquinto for their help with the official work and advice beyond academic activities.

I thank my post-graduate advisors from the Novosibirsk State University A.I. Kozhanov and A.F. Voevodin, for their help and encouraging me to study in Novosibirsk and consider academia as a career. Special thanks are due to V.P. Chuvakov, former Vice-Dean of Mechanics and Mathematics Department for his help, counsel and support during my study in Novosibirsk and afterwards. I thank Zorjana, Natalia Malkina, Natalia Chan, my graduate advisor Elena Litvin, and all my friends back in Russia. I also thank Konstantin for his friendship and encouragement.

Finally, I thank my mother for her understanding and care, for her unconditional love and endless support, which plays an important role in my life. I thank God.

Olga Gerber

Kiel, July 2005.

Contents

INTRODUCTION

It frequently happens that in attempting to obtain a solution to an important problem we realize that this problem is difficult. This observation is especially true for many optimization problems [6, 17, 36, 43, 45, 69, 73, 74].

Solving an optimization problem we want to have an algorithm that will find an optimal solution for any instance of the problem. It is commonly held opinion that an optimization problem has not been solved efficiently until a polynomial time (deterministic) algorithm has been obtained for it. Unfortunately, most real world optimization problems seem to be too hard to be solved efficiently and, in fact, even many simply stated problems are believed to be intractable. The theory of NP-*completeness* provides a mathematical foundation for this belief [16, 36].

We can informally summarize it as follows. A decision problem is one whose solution is either "yes" or "no". There are two classes of decision problems: NP and P. It holds that $\mathrm{P} \subseteq \mathrm{NP}$. Furthermore, all problems in P can be solved efficiently, whereas all problems in $\mathrm{NP} - \mathrm{P}$ are intractable. An NP-*complete problem* $\Pi \in \mathrm{NP}$ has the property: $\Pi \in \mathrm{P}$ if and only if $\mathrm{P} = \mathrm{NP}$.

The decision versions of many combinatorial optimization problems have been shown to be NP-*complete* [54]. We might say that such combinatorial optimization problems are NP-*hard*, since they are, in a sense, at least as hard as the NP-complete problems.

It is now widely accepted that NP-complete problems cannot be solved efficiently and $\mathrm{P} \neq \mathrm{NP}$. However, the problem "P versus NP" still remains one of the most challenging problems in mathematics, operations research and theoretical computer science, and it is also included in the list of Millennium Prize Problems [14]. On the one hand this "million dollar" problem is closely related to deep theoretical questions that have been puzzling mathematicians for decades. On the other hand, NP-hard computational problems frequently arise in many application areas of Computer Science and Operations Research. One of the striking examples is a variety of NP-hard 2-dimensional packing problems, which play an important role in such areas as cutting stock, VLSI design, image processing, and multiprocessor scheduling, just to name a few.

If an optimization problem is NP-hard, then there exists no algorithm which would compute optimal solutions in polynomial time, unless P = NP. But, we can ask for less. We could relax the requirement for the running time to be polynomial or we need not require the solutions to be optimal. Indeed, we can use *heuristic* algorithms like Local Search [1] and *enumeration* algorithms like Branch-and-Bound [44]. However, in the *worst-case analysis* such algorithms are either not polynomial or produce very *sub-optimal* solutions.

In this thesis we are interested in the design and analysis of *approximation algorithms* for 2-dimensional packing problems that always compute *near-optimal* solutions in polynomial time [6, 43, 45].

Approximation Algorithms. An optimization problem can be either cost minimization or profit maximization. Informally, an optimization problem Π of cost minimization consists of a set $\mathfrak{I}$ of instances (inputs) and a cost function C. An optimization problem Π is a profit maximization problem if it consists of a set $\mathfrak{I}$ of instances (inputs) and a profit function P. A set of feasible solutions (outputs) $F(I)$ is associated with each instance $I \in \mathfrak{I}$. For each instance I and a feasible solution $S \in F(I)$, the profit (cost) associated with I and S is $P(I,S) \in \mathbb{R}^+$ (respectively $C(I,S) \in \mathbb{R}^+$). The kind of optimization problems we typically deal with are of profit maximization problems; therefore, the discussion here is primarily in terms of profit problems. It is not difficult to develop the analogous concepts for cost minimization problems.

Let ALG be any algorithm for a profit maximization problem Π. Let $\mathrm{ALG}[I]$

denote a feasible solution produced by ALG given the instance I, and let

$$\mathrm{ALG}(I) \;=\; P(I, \mathrm{ALG}[I])$$

denote the profit incurred by ALG. An *optimal algorithm* OPT is such that for each instance I,

$$\mathrm{OPT}(I) \;=\; \max_{S \in F(I)} P(I,S).$$

An algorithm ALG is a ρ-approximation algorithm for a profit maximization problem Π if for all instances I,

$$\mathrm{ALG}(I) \;\geq\; \rho \cdot \mathrm{OPT}(I).$$

The running time of ALG is polynomial in the instance size $|I|$.

(For a cost minimization problem $\mathrm{ALG}(I) \;\leq\; \rho \cdot \mathrm{OPT}(I)$, where $\mathrm{OPT}(I) \;=\; \min_{S \in F(I)} C(I,S)$.)

The value of $\rho \leq 1$ is called the *approximation ratio* or *performance ratio* or *worst-case ratio* of ALG and in general it can be a function of $|I|$ (For a cost minimization problem $\rho \geq 1$). If ρ is achieved on instances I with $\mathrm{OPT}(I)$ tending to infinity, then ALG is said to be an *asymptotic* ρ*-approximation algorithm*, where

$$\rho = \liminf_{\mathrm{OPT}(I) \to \infty} \frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)}.$$

The *size* of instance $I \in \mathfrak{I}$, denoted by $|I|$, is defined as the number of digits (possibly bits) needed to present I under the assumption that all numbers occurring in I are written in binary alphabet $\{0,1\}$.

A family of approximation algorithms, $\{A_\varepsilon\}_{\varepsilon>0}$, for a profit maximization problem Π is called a *polynomial time approximation scheme* or a PTAS, if each algorithm A_ε is a $(1-\varepsilon)$-approximation algorithm and its running time is polynomial in the size of the instance. If the running time of each A_ε is polynomial in the size of the instance and $1/\varepsilon$, then $\{A_\varepsilon\}_{\varepsilon>0}$ is called a *fully polynomial time approximation scheme* or a FPTAS. Similarly, an asymptotic PTAS (FPTAS) is defined, where each A_ε is an asymptotic $(1-\varepsilon)$-approximation algorithm.

For any given NP-hard optimization problem, we wish to determine whether it possesses a ρ-approximation algorithm, or a PTAS, or even a FPTAS. Thus, on one hand, *positive (approximability) results* in the area of approximation concern the design and analysis of good polynomial time approximation algorithms and schemes, and on the other hand, the *negative (inapproximability) results* disprove the existence of such algorithms.

Outline of the thesis

In the last three decades, approximation algorithms have become a major area of theoretical computer science, operations research and discrete mathematics, rich in its powerful techniques and methods [6, 43, 85]. Packing problems are among the most popular ones for which approximation algorithms have been analyzed. On one hand, motivated by the well-known difficulty to obtain good lower bounds for the problems, it is particularly hard to prove results on the performance of the algorithms. On the other hand, theoretically oriented studies of approximation algorithms for packing have also impacts on the development of better algorithms for real world applications.

There has recently been an increasing interest in solving a variety of 2-dimensional packing problems such as strip packing [57, 79, 84], 2-dimensional bin packing [10, 12, 13, 18, 81], storage packing (packing rectangles with weights) [7, 8, 51] and storage minimization (packing squares into a rectangle of minimum area) [59, 67, 68, 70, 71]. These problems arise in a large variety of application areas of Computer Science and Operations Research, such as cutting stock, VLSI design, image processing, multiprocessor scheduling, etc.

- The storage minimization problem, i.e. the problem of packing squares into a rectangle of minimum area, can be formulated as follows [67, 68]: Find the minimum value x such that any set of squares of total area 1 can be packed into a rectangle of area x. Regarding lower bounds for this problem, there is just one non-trivial result known [70]: the set L of four squares with side lengths $s_1 = \sqrt{\frac{1}{2}}$, $s_2 = s_3 = s_4 = \sqrt{\frac{1}{6}}$ shows that the value of x is at least $\frac{2+\sqrt{3}}{3} > 1.244$. On the other hand, there are a number of quite complicated algorithms yielding several upper bounds for this problem. As it was shown in [66], any set L of squares with side lengths at most $s_{\max}$ can be packed into a square of size $a = s_{\max} + \sqrt{1 - s_{\max}}$. Later in [65], this result was extended by showing that any set L of squares of total area V can be packed into a rectangle of size $a_1 \times a_2$, provided that $a_1 > s_{\max}$, $a_2 > s_{\max}$ and $s_{\max}^2 + (a_1 - s_{\max})(a_2 - s_{\max}) \geq V$. Hence, the value of x is upper bounded by 2. Further results in this direction were obtained in [59], where it was proven that any set L of squares of total area V can be packed into a rectangle of size $\sqrt{2V} \times 2\sqrt{V}/\sqrt{3}$. Thus, substituting $V = 1$, the value of x is upper bounded by $\sqrt{\frac{8}{3}} \approx 1.633$. Finally, the result presented in [71] shows that any set L of squares of total area 1 can be packed into a rectangle whose area is less than 1.53.

- The 2-dimensional bin packing problem is stated as follows [13]: Given a set L of rectangles of specified size (width and height), pack them into the minimum number of unit size square bins. The problem is strongly NP-hard [62] and no approximation algorithm for it has an approximation ratio smaller than 2, unless P = NP [26]. A long history of approximation results exists for this problem and its variants [10, 12, 13, 81]. Very recently a number of asymptotic results have been obtained for it (i.e. for the case when the optimum uses a large number of bins). The best approximation algorithm obtained by Caprara [12] has an asymptotic worst-case ratio $1.691\ldots$. In [10] it was proven that the general version of the problem does not admit an asymptotic PTAS, unless P = NP. However, there is an asymptotic PTAS if all rectangles are actually squares [10, 18]. Also, in [18] a polynomial algorithm was presented which packs any set L of rectangles into at most $N^{opt}(L)$ augmented bins of size $(1+\varepsilon)$ for any $\varepsilon > 0$, where $N^{opt}(L)$ denotes the minimum number of unit size bins required to pack the rectangles in L.

- The strip packing problem is formulated as follows [37]: Given a set L of rectangles, it is required to pack them into a vertical strip $[0,1] \times [0,+\infty)$ so that the height of the packing is minimized. The strip packing problem is strongly NP-hard since it includes the bin packing problem as a special case. Many strip packing ideas come from bin packing. The "Bottom-Left" heuristic has asymptotic performance ratio 2 when the rectangles are sorted by decreasing widths [9]. In [15] several simple algorithms were studied that place the rectangles on "shelves" using one-dimensional bin packing heuristics. It was shown that the First-Fit shelf algorithm has asymptotic performance ratio 1.7 when the rectangles are sorted by decreasing height. The asymptotic performance ratio was further reduced to $3/2$ [83], then to $4/3$ [38], and to $5/4$ [7]. Finally, in [57] it was shown that there exists an asymptotic FPTAS for this problem. For the case of absolute performance ratio, the two currently best algorithms have performance ratio 2 [79, 84].

- The problem of 2-dimensional storage packing (packing rectangles with weights) can be formulated as follows [8]: Given a set L of rectangles with positive weights, it is required to pack a subset of L into a rectangular region so as to maximize the total weight of the packed rectangles. For a long time the only known result has been an asymptotic $(4/3)$-approximation algorithm for packing squares with unit profits into a rectangle [8]. Only very recently this algorithm for packing unit profit squares has been improved to a PTAS [50]. For packing rectangles with weights, several approximation algorithms were presented in [51]. The best one is a $(\frac{1}{2} - \varepsilon)$-approximation algorithm, for any fixed $\varepsilon > 0$.

In this thesis we address several versions of the above mentioned 2-dimensional packing problems, and aim at the design of approximation algorithms which find solutions that are arbitrary close to the optimum. We contribute in two ways. First, we give answers to some theoretical questions in approximability. Second, we present novel techniques that lead to efficient approximation algorithms that can be used in practical applications.

The main part of this thesis is divided into five chapters. One can find some relationship between them. However, each chapter is intended to be mostly self-contained, and we hope that the reader interested in a particular topic would have no problem in reading only the corresponding part.

CHAPTER 1: In the first chapter we initiate the study of the storage packing problem. Here we address a version of the problem which naturally finds applications in real-life problems. Namely, we consider a version where a set of squares is packed into a unit size square frame. That is, given a set of weighted squares, pack a subset into a unit size square frame so that the total weight of the packed squares is maximized. We study a special case of the problem, in which the squares' areas are taken as weights, i.e. we are interested in covering the maximum area of a unit square by squares. Formally, we are given a set Q of n squares S_i ($i = 1, \dots, n$) with side lengths $s_i \in (0,1]$. For a given subset $Q' \subseteq Q$, a *packing* of Q' into a unit size square frame is a positioning of the squares from Q' within $[0,1] \times [0,1]$ such that the squares of Q' have disjoint interiors. The goal is to find a subset $Q' \subseteq Q$, and a packing of Q' within $[0,1] \times [0,1]$, of maximum area, $\sum_{S_i \in Q'} (s_i)^2$. The decision version of our problem, determining whether a set of squares can be packed into a rectangle, is NP-complete [63]. Our main result is that for any set Q of n squares and any accuracy $\varepsilon > 0$, there exists an algorithm A_ε which finds a subset of Q and its packing within a unit square frame $[0,1] \times [0,1]$ of total area $A_\varepsilon(Q) \geq (1-\varepsilon)\mathrm{OPT}(Q)$, where $\mathrm{OPT}(Q)$ is the maximum area which can be covered by packing any subset of Q. The running time of A_ε is polynomial in the number of squares n, but it is exponential in $1/\varepsilon$. We also give some ideas about how this result can be generalized for the d-dimensional version of the storage packing problem.

CHAPTER 2: In this chapter we continue the study of the storage packing problem. It would be natural to extend the above result for packing squares with areas equal to weights to the case of arbitrary weights. However even if weights are identical the problem is still strongly NP-hard [62]. Here we try a different approach. We want to investigate how restrictions on the resources can influence the approximation property of the problem.

In particular, we study the so-called case of resource augmentation, that is, we allow the length of the unit square frame to be increased by some small value. It turns out that this relaxation allows to obtain the best possible approximation results even for a more general version of the problem. Formally, we are given a set R of n rectangles, R_i ($i = 1,\ldots,n$) with widths $a_i \in (0,1]$, heights $b_i \in (0,1]$, and weights $w_i \geq 0$. For a given subset $R' \subseteq R$, a *packing* of R' into a unit size square frame $[0,1]\times[0,1]$ is a positioning of the rectangles of R' within the frame such that they have disjoint interiors. The goal is to find a subset $R' \subseteq R$, and a packing of R' within $[0,1]\times[0,1]$ of maximum weight, $\sum_{R_i\in R'} w_i$.

We derive an algorithm W_ε which, given any set R of n rectangles and any accuracy $\varepsilon > 0$, finds a subset of R and its packing within an augmented unit square frame, $[0,1+3\varepsilon]\times[0,1+3\varepsilon]$, of total weight $W_\varepsilon(R) \geq (1-\varepsilon)\mathrm{OPT}$, where OPT is the maximum weight that can be obtained by packing any subset of R into a unit size square frame $[0,1]\times[0,1]$. The running time of W_ε is polynomial in the number of rectangles, but it is exponential in $1/\varepsilon$.

To simplify the presentation of results, we first address the special case of the problem where all rectangles to be packed are squares. Presenting the algorithm for this simpler problem will help to understand the solution for the more complex problem of packing rectangles. Specifically, we present an algorithm A_ε which given a set of squares L finds a subset of L and its packing into the augmented unit square $[0,1+\varepsilon]\times[0,1+\varepsilon]$ with weight $A_\varepsilon(L) \geq (1-\varepsilon)\mathrm{OPT}$, where OPT is the maximum weight that can be achieved by packing any subset of L in the original unit square region $[0,1]\times[0,1]$. The running time of A_ε is polynomial in the number of squares. Here we also give some ideas about how this result can be extended to the case of packing d-dimensional cubes into a d-dimensional cube of size $1+\varepsilon$, for $d \geq 2$.

One can see that our problem is dual to the 2-dimensional bin packing problem [13, 10]. On the one hand, we make a significant step to close the gap between the two problems, by giving some rounding transformations which allow the usage of the known algorithm from [18]. On the other hand, we refine some known approximation techniques from knapsack problems, strip packing, and scheduling problems. Our algorithm for packing squares is based on a few simple ideas and, contrasting to the recent algorithms for packing problems [10, 18, 51, 57], it does not use linear programming. In spite of the progress made, the question of finding near-optimal $(1-\varepsilon)$-solutions for the general problem of packing a set of rectangles with weights into a square frame without augmentation remains a challenging open problem.

CHAPTER 3: In this chapter we address the general version of the storage packing problem. Inspired by the results in the previous chapter we investigate the influence of resources. Here we consider the so-called case of large resources, when the number of the packed rectangles is relatively large. Formally, we are given a dedicated rectangle R of width $a \geq 0$ and height $b \geq 0$, and a list L of n rectangles R_i $(i = 1, \ldots, n)$ with widths $a_i \in (0, a]$, heights $b_i \in (0, b]$, and positive integral weights $w_i \geq 0$. For a sublist $L' \subseteq L$ of rectangles, a *packing* of L' into the dedicated rectangle R is a positioning of the rectangles from L' within the area $[0, a] \times [0, b]$, so that all the rectangles of L' have disjoint interiors. Rectangles are not allowed to rotate. The goal is to find a sublist of rectangles $L' \subseteq L$ and its packing in R which maximizes the weight of packed rectangles, i.e., $\sum_{R_i \in L'} w_i$.

In the large resources version we assume that all rectangles R_i $(i = 1, \ldots, n)$ in the list L have widths and heights $a_i, b_i \in (0, 1]$, and the dedicated rectangle R has unit width $a = 1$ and quite a large height $b \geq 1/\varepsilon^4$, for a fixed positive $\varepsilon > 0$. We present an algorithm which finds a sublist $L' \subseteq L$ of rectangles and its packing into the dedicated rectangle R with a weight at least $(1-\varepsilon)\mathrm{OPT}$, where OPT is the optimum weight. The running time of the algorithm is polynomial in the number of rectangles n and exponential in $1/\varepsilon$.

Our approach to approximation is as follows. At the beginning we take an optimal rectangle packing inside of the dedicated rectangle, considering it as a strip packing. We then perform several transformations that simplify the packing structure, without dramatically increasing the packing height and decreasing the packing weight, such that the final result is amenable to a fast enumeration. As soon as we find such a ”near-optimal” strip packing, we apply our shifting technique. This puts the packing into the dedicated rectangle by removing some less weighted piece of the packing.

Here, as an application of our algorithm, we provide a $(\frac{1}{2}-\varepsilon)$-approximation algorithm for the advertisement placement problem for newspapers and the Internet, which can be formulated as the problem of packing weighted rectangles into k identical rectangular bins so as to maximize the total weight of the packed rectangles. The algorithm proceeds as follows. First, it takes all k bins together, as a rectangle of size $(a, k\cdot b)$, and runs our algorithm for packing weighted rectangles. This outputs a packing whose profit is at least $(1-\varepsilon)\mathrm{OPT}$. Next, the algorithm draws $(k-1)$ vertical lines which cut this packing into k bins. There are two solutions: one whose rectangles lie inside the bins, and one whose rectangles are cut by the lines. So, the algorithm outputs the maximum of them whose weight is at least $(1-\varepsilon)\mathrm{OPT}/2$.

CHAPTER 4: In this chapter we continue our work on the problem addressed in Chapter 3, namely, on the storage packing problem with large resources. Here our aim is to derive a more efficient approximation algorithm. Using some novel approximation techniques, we significantly improve the running time of the algorithm. In particular we present an algorithm which finds a packing of a sublist of L into the rectangle R whose total weight is at least $(1-\varepsilon)\mathrm{OPT}(L)$, where $\mathrm{OPT}(L)$ is the optimum. The running time of the algorithm is polynomial in n and, contrasting to the previous result, is also polynomial in $1/\varepsilon$. In other words we derive a fully polynomial time approximation scheme (FPTAS) with large resources.

Our approach to approximation is as follows. At the beginning we relax the problem to *fractional packing*: any rectangle can be first cut by horizontal lines into several rectangles of the same width, and then some of them can be independently packed. The fractional relaxation formulates as a linear program (LP).

In general, the LP consists of an exponential number of variables. Hence, we cannot solve it directly. Our main idea here is to reformulate the LP as an instance of the *resource-sharing* problem and then make use of some recent approximation tools for it (see [40, 47], Section 4.2.2 and Appendix 5.4 for details). This requires a number of subsequent technical results, which, however, we obtain in quite an elegant way.

By approximating a sequence of $O(n/\varepsilon^2)$ instances of the resource-sharing problem, we are able to find an approximate fractional solution. Our next idea is to round this solution. By solving and rounding $O(1/\varepsilon^2)$ instances of the fractional knapsack problem we find a list of rectangles which is quite a good approximation for the original problem. The weight of the list is $(1-\varepsilon)$ times the optimum, and a strip packing algorithm [56] can pack it in the area $[0,a]\times[0,(1+\varepsilon)b]$. So, similar to the previous approach we can apply our shifting technique and obtain a packing within $[0,a]\times[0,b]$ with total weight at least $(1-\varepsilon)$ times the optimum.

Interestingly, by considering a weekly restricted case we are able to achieve the best possible approximation result, in terms of trade-off between approximation ratio and running time. This makes a significant step in understanding the approximation properties of the problem. Furthermore, the difference in the side lengths of the rectangles yields that the number of the packed rectangles is large, that can be met quite often in practice. In order to be able to cope with the problem we also design several new approximation techniques, some of them are nice combinations of various classical techniques used for knapsack problems, strip packing, and, surprisingly, for the resource-sharing problem. This demonstrates quite a strong relation between several variants of packing.

CHAPTER 5: In this chapter we address the strip packing problem with rotations by 90 degrees, where a set of rectangles is packed into a vertical strip of unit width so that the height, to which the strip is filled, is minimized. Formally, in the input we are given a set of n rectangles, $R=\{(a_1,b_2),\ (a_2,b_2),\ \ldots,\ (a_n,b_n)\}$, with side lengths a_j,b_j ($j=1,\ldots,n$) in the interval $[0,1]$. Rotations by 90 degrees are allowed. That is, for each rectangle (a_j,b_j) ($j=1,\ldots,n$) there is a binary variable $x_j\in\{0,1\}$: if $x_j=1$, we allocate (a_j,b_j) to a *non-rotated* rectangular frame, $R_j(x_j) = a_j \times b_j \cdot x_j$, whose width is a_j and height is $b_j \cdot x_j$; otherwise $x_j = 0$, and we allocate (a_j, b_j) to a *rotated* rectangular frame, $R'_j(x_j) = b_j \times a_j \cdot (1 - x_j)$, whose width is b_j and height is $a_j \cdot (1 - x_j)$, respectively. Then, a set of (rotated and non-rotated) frames, $R(x)$, defines an allocation of R. A strip-packing of $R(x)$ is a positioning of the frames of $R(x)$ within the vertical strip of unit width, $[0,1] \times [0,\infty)$, so that no two frames have intersecting interiors. The height of a strip-packing is defined as the height to which the strip is filled by the frames. In the strip packing problem with rotations by 90 degrees it is required to find an allocation, $R(x)$, and a strip-packing of $R(x)$ so that the packing height is minimized. Our result can be stated as follows: There is an algorithm, which given a set of n rectangles, R, with side lengths at most 1, and a positive accuracy, $\varepsilon > 0$, finds an allocation of R to a set of frames, $R(x)$, and a strip-packing of the frames of $R(x)$ whose height is at most $(1+\varepsilon)\mathrm{OPT}(R) + O(1/\varepsilon^2)$, where $\mathrm{OPT}(R)$ is the height of the optimal strip-packing of R with rotations by 90 degrees. The running time of the algorithm is polynomial in n and $1/\varepsilon$.

In other words, we present an asymptotic fully polynomial time approximation scheme (AFPTAS) (an equivalent result has been independently obtained by Jansen and van Stee in [49]). The existence of such a scheme has been an open theoretical problem for some years [19]. Besides that, we develop new techniques which allow us to use a known algorithm for the strip packing problem (without rotations) in [57]. This closes the gap between the classical statement of the strip packing problem and its extension to rotations by 90 degrees.

Applications. More generally, it should be noted that – although phrased in terms of “packing" – the most of our results really are about dynamic storage, i.e., given a set of tasks L and a resource pool R, we fix the resources R and attempt to maximize the amount of tasks from L serviced. As known, this problem is NP-hard. There are two natural questions: Which restrictions make the problem hard? How can they be relaxed to get an efficient solution? In this work we propose to look at the resource constraints. One way we follow is to augment the resource pool R to $(1+\varepsilon)R$, that is, we add a small fraction of resources to the system. We show that this relaxation allows to serve efficiently at least a fraction $(1-\varepsilon)$ of the maximum amount of the tasks in L (see Chapter 2, Sections 2.2, 2.3). Yet, we point out that the high granularity of L, i.e. the tasks of L vary little and are small comparing to the resource pool R, allows very fast near optimal solutions (see Chapter 1, Section 1.2.1 and Chapter 2, Section 2.2.1).

Another way we follow is to leave the resources of R unchanged, but to over-provision the system such that the resources of R are large. We show that if the resources of R are $\Omega(1/\varepsilon^4)$ larger than each task in L, one can efficiently serve at least a fraction $(1-\varepsilon)$ of the maximum amount of tasks in L (see Chapters 3, 4).

One can also find applications of our later results in the advertisement placement problem for newspapers and the Internet [2, 33]. In a basic version of the problem, we are given a list of n advertisements and k identical rectangular pages of fixed size (a,b), on which advertisements may be placed. Each ith advertisement appears as a small rectangle of size (a_i,b_i) and is associated with a profit p_i ($i=1,\dots,n$). Advertisements are not allowed to overlap. The goal is to maximize the total profit of the advertisements placed on all k pages.

This problem can be formulated as the problem of packing weighted rectangles into k identical rectangular bins so as to maximize the total weight of the packed rectangles. Here, as an application of our algorithm, we can simply design a $(\frac{1}{2}-\varepsilon)$-approximation algorithm in the case that the number of bins $k \geq \lceil 1/\varepsilon^4 \rceil$, for some small $\varepsilon > 0$. The running time of the algorithm is polynomial in n and $1/\varepsilon$ (see Chapter 3, Section 3.6).

As we mentioned above, our results can also find applications in *multiprocessor scheduling* [25, 32]. In the *parallel* version of the problem we are given a set of n tasks $T=\{1,\dots,n\}$ and a set of m processors $M=\{1,\dots,m\}$. Each task $j \in T$ has a unit processing time $p_j \in \mathbb{N}$, an integral due date d_j, a positive weight $w_j > 0$ and requires $size_j$ processors. The goal is to maximize the *weighted throughput* $\sum w_j \bar{U}_j$, i.e. the total weight of *early* tasks j that meet their due dates d_j ($\bar{U}_j = 0$ if task j completes after d_j, and $\bar{U}_j = 1$ otherwise). In this parallel variant the multiprocessors architecture is disregarded and for each task $j \in T$ there is given a prespecified number $size_j \in M$ which indicates that the task can be processed by any subset of processors of the cardinality equal to $size_j$. The tasks have a *common* due date if $d_j = D$ for all tasks j, where D is the largest due date $max_j d_j$. This problem can be formulated as the problem of packing weighted rectangles into a rectangular frame of total height D so as to maximize the total weight of the packed rectangles.

Manufacturing companies need to decide how to cut a piece of raw material, say wood or cloth, into the largest number of parts, say shelves or sheets, needed to produce items. This problem is called *cutting stock*. The strip packing problem, which we consider in the last chapter, is the following version of a two-dimensional *cutting stock* problem [57]: Given a supply of material consisting of one rectangular strip of fixed width 1 and large height, given a demand of n rectangles with widths and heights in the interval $[0,1]$, the problem is to cut the strip into the demand rectangles while minimizing the waste, i.e., minimizing the total height used.

Finally, a Very Large Scale Integrated (VLSI) design is a broad area where one can find applications of our results. A considerable part of optimization problems in VLSI design is based on rectangle packing problem in order either to minimize the area of rectangle (*chip*), where rectangular modules need to be packed, or to maximize the total profit of rectangles packed into a rectangular frame. For example, to minimize power consumption and energy dissipation, and to maximize the speed of chips, it is desired to pack a large number of components (rectangles) into the minimum possible area (size of the bin). Transportation and storage companies need to pack large containers (rectangles, boxes) storing goods into the smallest number of storage rooms (bins), etc. Many other problems can be formulated as 2-or 3-dimensional packing problems, indeed.

Last notes. We assume that the reader is familiar with the basic concepts of combinatorial optimization, complexity theory and approximation algorithms which can, for instance, be found in the following books [6, 17, 36, 41, 43, 45, 69, 74, 73]. There is a number of books on linear programming [11, 22, 72, 78, 80]. For the sake of convenience, we also give all main definitions from complexity theory in Appendix A on page 131. We give a description of the algorithm of C. Kenyon and E. Rémila [56, 57] for the strip packing in Appendix B on page 139 and a brief description of the algorithm by M.D. Grigoriadis et.al [40] for the resource sharing problem in Appendix C on page 149.

Parts of this thesis have been published or will be published in [27, 28, 29, 30, 31].

CHAPTER 1

ON COVERING THE MAXIMUM AREA BY SQUARES

1.1 INTRODUCTION

In this chapter we initiate the study of the storage packing problem, addressing a version of the problem, where a set of squares is packed into a unit size square frame. That is, given a set of weighted squares we wish to pack a subset of them into a unit size square frame $[0,1]\times[0,1]$ so that the total weight of the packed squares is maximized.

Here we present an algorithm for the special case of the problem, in which the squares' weights and areas coincide. In other words, in this case we wish to pack a set of squares whose weights and areas are the same, i.e. we are interested in covering the maximum area of a unit square by a subset of squares. Formally, we are given a set Q of n squares S_i $(i = 1,\ldots,n)$ with side lengths $s_i \in (0,1]$. For a given subset $Q' \subseteq Q$, a *packing* of Q' into a unit size square frame $[0,1]\times[0,1]$ is a positioning of the squares in Q' within the frame such that their interiors are disjoint. The goal is to find a subset $Q' \subseteq Q$, and a packing of Q' within $[0,1]\times[0,1]$ of maximum area, $\sum_{S_i\in Q'}(s_i)^2$. Our first main result can be stated as follows.

Theorem 1.1.1. *For any set Q of n squares and any accuracy $\varepsilon > 0$, there exists an algorithm A_ε which finds a subset of Q and its packing within the unit square frame $[0,1]\times[0,1]$, with area*

$$A_\varepsilon(Q) \geq (1-\varepsilon)\mathrm{OPT},$$

where OPT *is the maximum area that can be covered by packing any subset of Q. The running time of A_ε is polynomial in n for fixed* ε.

This result can be extended to the case of packing d-dimensional cubes into a unit d-dimensional square cube, for $d \geq 2$.

In the following sections we give our proof for Theorem 1.1.1 and describe an algorithm for nearly covering maximum area using squares.

1.2 AN ALGORITHM FOR COVERING MAXIMUM AREA USING SQUARES

Let Q be a set of n squares S_i $(i = 1, \dots, n)$ with side lengths $s_i \in (0,1]$. The goal is to find a subset $Q' \subseteq Q$, and a packing of Q' within $[0,1] \times [0,1]$, of maximum area, $\sum_{S_i \in Q'} (s_i)^2$.

Assume first, that all squares S_i in Q are small, namely, their side lengths s_i are at most ε, for some small ε. Then, we can apply the Next-Fit-Increasing-Height (NFIH) heuristic to pack the squares of Q within a unit square frame $[0,1] \times [0,1]$ (see Section 1.2.1), so that the total area covered by the packed squares is at least $min\{area(Q), 1 - 2\varepsilon + \varepsilon^2\}$ for any $\varepsilon > 0$. That is, we either pack all squares or obtain a packing which covers at least a fraction $(1 - 2\varepsilon)$ of the total area of the frame.

For the case of squares of arbitrary sizes, we partition Q into two sets formed by small and large squares, respectively. If we define these set properly, then any feasible packing of the squares in $[0,1] \times [0,1]$ will only contain $O(1)$ large squares. So, in $O(1)$ time we can enumerate all possible *tight packings* for the large squares, where a tight packing does not allow a large square to move to the left or down. For each tight packing of the large squares, we then try to fill up all empty gaps with small squares. More specifically, we take the small squares one by one in non-decreasing order of size s_i, and use the NFIH heuristic. Among all packings found we select one with the maximum area. The main problem is to define the sets of large and small squares so that the area covered is nearly optimal.

For a subset of squares $Q' \subseteq Q$, we use $area(Q')$ to denote the area, $\sum_{S_i \in Q'} s_i^2$, of Q'. In addition, we use Q^{opt} to denote an optimal subset of Q that can be packed in the unit square $[0,1] \times [0,1]$. So,

$$area(Q^{opt}) = \text{OPT and } area(Q^{opt}) \leq 1.$$

For the rest of the chapter, we assume w.l.o.g. that the value of $1/\varepsilon$ is integral.

1.2.1 The NFIH Heuristic

We consider the following simplified version of the square packing problem: given a positive value $\beta \in \mathbb{R}^+$, a set S of squares S_i with side lengths $s_i \leq \varepsilon^\beta$, and a rectangular frame $[0,a] \times [0,b]$ $(a, b \in [0,1])$, pack a subset of S into the frame such that the area covered by the squares is maximized.

First, we sort the squares $S_i \in S$ non-decreasingly by size. Then, we place the squares within $[0,a] \times [0,b]$ by using the Next-Fit-Increasing-Height (NFIH) heuristic; this packs the squares into a sequence of sublevels. The first sublevel is the bottom of the frame. Each subsequent sublevel is defined by a horizontal line drawn through the top of the largest square placed on the previous sublevel. The squares are packed one by one in a left-justified manner, until the next square cannot fit within the current sublevel. At that moment, the current sublevel is closed and a new sublevel is started. The packing procedure runs as above until there are no more squares in S or the next square in the sequence would cross the top b of the frame. For an illustration see Fig. 1.1.

The following result is a slightly tighter bound on the performance of NFIH than the one that can be derived from [18].

Lemma 1.2.1. *Let S be any set of squares S_i with sizes $s_i \leq \varepsilon^\beta$, and let $[0,a] \times [0,b]$ $(a, b \in [0,1])$ be a rectangular frame. The NFIH heuristic, which selects squares S_i in non-decreasing size, outputs a packing of a subset of S whose area is at least* $\min\{area(S), ab - \varepsilon^\beta(a+b) + \varepsilon^{2\beta}\}$.

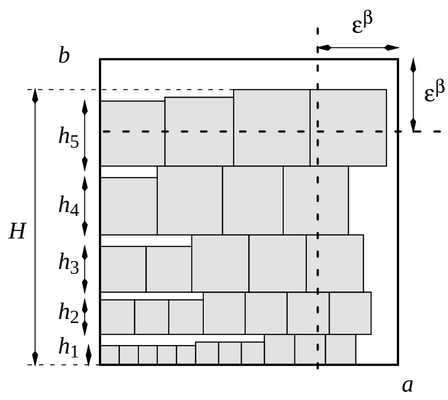


Figure 1.1: NFIH for small squares.

Proof. Let q be the number of sublevels and let h_i be the height of the first square on the ith sublevel. Let H be the height of the packing. If no square in S is left unpacked, then the area covered is $area(S)$. Hence, assume that some squares in S are left unpacked. Since all side lengths $s_i \leq \varepsilon^\beta$, then $b - H \leq \varepsilon^\beta$. Furthermore, on each sublevel i, $i = 1, \ldots, q-1$, the area covered by the squares is at least $(a - \varepsilon^\beta)h_i$. Thus, the total area covered is at least $H(a-\varepsilon^\beta) \geq (b-\varepsilon^\beta)(a-\varepsilon^\beta) \geq a \cdot b - \varepsilon^\beta(a+b) + \varepsilon^{2\beta}$. ☐

Corollary 1.2.2. *If all squares S_i in Q have sizes s_i at most $\varepsilon \leq 1$, then the NFIH heuristic packs a subset of Q within $[0,1] \times [0,1]$ of total area at least $(1-2\varepsilon)\mathrm{OPT}(Q)$. The running time of the algorithm is $O(n \log n)$.*

Proof. By using NFIH we pack a subset of Q within $[0,1] \times [0,1]$. If not all the squares in Q are packed, by Lemma 1.2.1 the covered area is at least $1 - 2\varepsilon + \varepsilon^2 \geq 1 - 2\varepsilon$. Since $\mathrm{OPT} \leq 1$, the minimum area covered is at least $(1-2\varepsilon)\mathrm{OPT}$. The running time of the algorithm is dominated by the sorting step. ☐

1.2.2 Partitioning the Squares

We define the group $Q^{(0)}$ of squares $S_i \in Q$ with side lengths s_i in $(\varepsilon^4, 1]$, and for $j \in \mathbb{Z}_+$ we define the group $Q^{(j)}$ of squares with side lengths in $(\varepsilon^{2^{j+1}+3}, \varepsilon^{2^j}]$.

Then,

$$\cup_{j=0}^{\infty} Q^{(j)} = Q \text{ and } Q^{(\ell)} \cap Q^{(j)} = \emptyset, \text{ for} |\ell - j| > 1.$$

Lemma 1.2.3. *There is a group* $Q^{(k)}$ *with* $0 \le k \le 2/\varepsilon^2 - 1$ *such that its contribution to the optimum is*

$$area(Q^{opt} \cap Q^{(k)}) \le \varepsilon^2 \mathrm{OPT},$$

where Q^{opt} *is an optimal subset of squares.*

Proof. Each square belongs to at most two consecutive groups. Therefore,

$$\cup_{k=0}^{2/\varepsilon^2-1} area(Q^{opt} \cap Q^{(k)}) \le 2\mathrm{OPT},$$

and so, there must be a group $Q^{(k)}$ as indicated in the lemma. □

Let $Q^{(k)}$ be a group such that $area(Q^{opt} \cap Q^{(k)}) \le \varepsilon^2\mathrm{OPT}$. We drop the squares $Q^{(k)}$ from consideration. Then, an optimal packing for $Q \setminus Q^{(k)}$ must cover area at least $(1-\varepsilon^2)\mathrm{OPT}$, i.e. this makes a loss of at most a factor of ε^2 in the optimum.

Next, we partition the squares in $Q \setminus Q^{(k)}$ into two groups: $L = \{S_i \mid s_i > \varepsilon^{2^k}\}$ and $S = \{S_i \mid s_i \le \varepsilon^{2^{k+1}+3}\}$. The squares in L and S are called *large* and *small*, respectively.

Corollary 1.2.4. *Let* $\alpha = 2^k$ *and* $\beta = 2^{k+1} + 3$, *where k is as defined above. The side of any large square is larger than* ε^α *and the side of any small square is at most* ε^β. *Moreover,*

$$area(Q^{opt} \cap (L \cup S)) \ge (1-\varepsilon^2)\mathrm{OPT}.$$

1.2.3 The set FEASIBLE and tight packings

We say that a subset of large squares is *feasible* if it can be packed into the unit square frame. Since the side length of any large square is at least ε^α, there are at most $1/\varepsilon^{2\alpha}$ large squares in each feasible subset. We define a set $FEASIBLE$ as a set which contains all feasible subsets. The *tight packing* of large squares is a packing, where every time that a large square is considered for packing, we put it in every position where it cannot move left or down.

1.2.4 Outline of the Algorithm

Here we give a high level description of the algorithm. The individual steps of the algorithm are analyzed in the next section.

Algorithm A_ε:

INPUT: A set of squares Q, accuracy $\varepsilon > 0$.

OUTPUT: A packing of a subset of Q within $[0,1] \times [0,1]$.

1. For each $k \in \{0, 1 \ldots, 2/\varepsilon^2 - 1\}$, form the group $Q^{(k)}$ of squares as described above.
   (a) Let $\alpha = 2^k$ and $\beta = 2^{k+1} + 3$.
   (b) Partition $Q \setminus Q^{(k)}$ into L and S, the sets of large and small squares with sides larger than ε^α and at most ε^β, respectively.
   (c) Compute the set $FEASIBLE$, containing all subsets of L with at most $1/\varepsilon^{2\alpha}$ large squares.
   (d) For every set in $FEASIBLE$, find all possible *tight packings* of its large squares. For each tight packing use the modified NFIH to pack the small squares in the empty gaps left by the large squares until no further small squares can be packed.
2. Among all packings produced, output one with the maximum area covered.

1.2.5 The Analysis of Algorithm A_ε

Large Squares. The set $FEASIBLE$ which contains all subsets of at most $1/\varepsilon^{2\alpha}$ large squares has polynomial size, $O(n^{\varepsilon^{-2\alpha}})$. We can prove the following result.

Lemma 1.2.5. *In $O(n^{O(1)})$ time we can find the set $FEASIBLE$ consisting of all subsets of at most $1/\varepsilon^{2\alpha}$ large squares from L. Any feasible set of large squares belongs to $FEASIBLE$. Moreover, the optimal set of large squares $L \cap Q^{opt}$ is feasible and, hence, it also belongs to $FEASIBLE$.*

Proof. By definition, any feasible set of large squares can be packed into the unit square, i.e. into a square area of size 1. The area of any large square is at least $\varepsilon^{2\alpha}$, hence, there are at most $1/\varepsilon^{2\alpha}$ large squares in any feasible set. There are at most n squares in L, so, there are $O(n^{1/\varepsilon^{2\alpha}})$ sets in $FEASIBLE$. Notice that the optimal set $L \cap Q^{opt}$ of large squares is also feasible, hence, it must belong to $FEASIBLE$. □

Lemma 1.2.6. *For any set $L' \in FEASIBLE$ of large squares, we can find in $O(1)$ time all possible tight packings of its large squares.*

Proof. Consider all possible permutations of the squares in L'. For each permutation we take the squares one by one and pack them in the square frame starting at the left bottom corner. Every time that a square is considered for packing, we put it in each position where it cannot move left or down, generating all possible packings.

This procedure works as follows. First square is placed in the left bottom corner. This gives just one packing. The second square can potentially generate two different packings, being placed on the top of the first square with its left side aligned with the left side of the large square $[0,1]\times[0,1]$, and on the top of the $[0,1]\times[0,1]$ square with its left side aligned to the right side of the first square. In step ℓ ($\ell = 3,\ldots,|L'|$), we consider the ℓth square. Let $N(\ell-1)$ be the number of all already generated packings by $1,2,\ldots,\ell-1$ squares. For each of these $N(\ell-1)$ packings, we place the ℓth square inside it so that it is aligned with its left or bottom sides either to two previously packed squares or to a previously packed square and the $[0,1]\times[0,1]$ square. This can generate at most $\ell \cdot N(\ell-1)$ new packings. By induction, $N(\ell) = \ell \cdot N(\ell-1) = \ell \cdot (\ell-1) \cdot N(\ell-2) = \ldots = \ell!$.

For each of $|L'|!$ permutations, we generate $|L'|!$ packings. Since $|L'| = O(1)$, we get $O(1)$ packings in overall.

□

Small Squares. We sort the small squares non-decreasingly by size. Assume that we have a tight packing of some set $L' \in FEASIBLE$. We define a *sliced* *structure* for this packing as follows. We draw a vertical line at each position where a large square starts or ends (see Fig 1.2). The space between any two consecutive vertical lines is called a *slice*. Looking into each slice we can see that the horizontal boundaries of the large squares cut some slices out. We work with the empty rectangular gaps inside the slices.

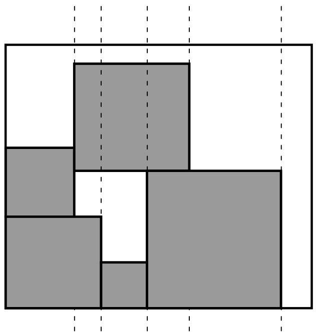


Figure 1.2: A sliced structure in a tight packing.

We add the small squares from S to the gaps by using the NFIH heuristic: We consider slices one by one, filling the gaps in a bottom-up manner using small squares. To fill a gap, we take small squares $S_i \in S$ one by one in order of non-decreasing size, and apply the NFIH heuristic, see Fig. 1.3. We can prove the following result.

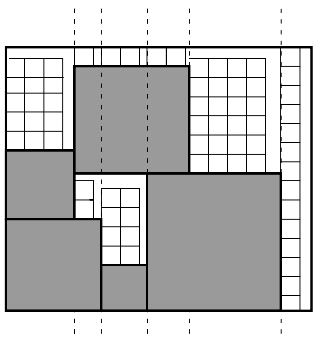


Figure 1.3: Packing the small squares.

Lemma 1.2.7. *For any feasible set $L' \in FEASIBLE$ which has a tight packing within the frame $[0,1] \times [0,1]$, the modified NFIH heuristic adds small squares to the packing in such a way that the area covered is at least $\min\{area(L') + area(S), 1-\varepsilon^2\}$, for any $0 < \varepsilon \leq 1/5$.*

Proof. Recall that $\alpha = 2^k$, $\beta = 2^{k+1} + 3$, and $|L'| \leq 1/\varepsilon^{2\alpha}$. The number of slices in a packing of L' is at most $2|L'|$. The widths of all slices add up to 1. The heights of all empty gaps in each slice add up to at most 1.

Assume that some small squares are left unpacked. Let q be the number of gaps, and let $x_1 * y_1, x_2 * y_2, \dots, x_q * y_q$ be their areas. Then,

$$q \leq (2|L'|)^2,$$

$$\sum_{j=1}^{q} x_j * y_j = 1 - area(L'),$$

and

$$\sum_{j=1}^{q} y_j \leq 2|L'| \text{ and } \sum_{j=1}^{q} x_j \leq 2|L'|.$$

To see that $\sum_{j=1}^{q} y_j \leq 2|L'|$, note that all rectangular gaps are inside the slices, so the sum of the lengths of their vertical boundaries is at most $2|L'|$, the total length of all the slices. The last inequality follows from a symmetry argument, i.e., if we draw horizontal slices instead of vertical ones, we obtain a similar figure but with respect to the widths x_j.

Remember that each small square in S has side length at most ε^β. Thus, using Lemma 1.2.1, we can bound the area covered by the small squares as follows

$$\begin{aligned}
AREA &= \sum_{j=1}^{q} (x_j * y_j - \varepsilon^\beta (x_j + y_j) + \varepsilon^{2\beta}) \\
&\geq (1 - area(L')) - \varepsilon^\beta (4|L'|) + \varepsilon^{2\beta} q \\
&\geq (1 - area(L')) - \varepsilon^\beta (4/\varepsilon^{2\alpha}) + \varepsilon^{2\beta}(4/\varepsilon^{4\alpha}) \\
&\geq (1 - area(L')) - 4\varepsilon^{\beta-2\alpha} + 4\varepsilon^{2\beta-4\alpha}, \text{ for } k = 0, \text{ we get} \\
&\geq (1 - area(L')) - 4\varepsilon^3 + 4\varepsilon^6 \geq (1 - area(L')) - \varepsilon^2, \text{ since } 4\varepsilon^3 - 4\varepsilon^6 \leq \varepsilon^2 \\
&\quad \text{for } \varepsilon \in (0, 1/5].
\end{aligned}$$

$\square$

1.2.6 Proof of Theorem 1.1.1

The algorithm A_ε considers all values $k \in \{0, 1 \ldots, 2/\varepsilon^2 - 1\}$ and groups $Q^{(k)}$. By Lemma 1.2.3 at least for one of these groups $Q^{(k)}$,

$$area(Q^{opt} \setminus Q^{(k)}) \geq (1-\varepsilon^2)\text{OPT}.$$

Consider one such group $Q^{(k)}$ and let $\alpha = 2^k$ and $\beta = 2^{k+1}+3$. Partition $Q \setminus Q^{(k)}$ into the sets of large and small squares, L and S, where the side length of each large square is larger than ε^α and the side length of each small square is at most ε^β.

We know that $Q^{opt} \cap L$ belongs to the set $FEASIBLE$, which consists of all sets with at most $1/\varepsilon^{2\alpha}$ large squares. Since Q^{opt} can be packed within the frame $[0,1] \times [0,1]$, there exists a tight packing for $Q^{opt} \cap L$ as well. For each such a tight packing, the NFIH heuristic adds small squares to the packing such that the total area covered by the squares is at least

$$\min\{area(Q^{opt} \cap L) + area(S), 1-\varepsilon^2\}.$$

Since $\text{OPT} \leq 1$,

$$1-\varepsilon^2 \geq (1-\varepsilon^2)\text{OPT}.$$

On the other hand, since $area(Q^{(k)}) \leq \varepsilon^2\text{OPT}$, then

$$area(Q^{opt} \cap L) + area(S) \geq area(Q^{opt} \setminus Q^{(k)}) \geq (1-\varepsilon^2)\text{OPT}.$$

We also know that the set $FEASIBLE$ and all possible tight packings of large squares can be found in $O(n^{O(1)})$ time. The NFIH heuristic runs in time polynomial in the number of squares, n. Hence, the overall running time of the algorithm is polynomial in n for fixed ε.

1.2.7 Remark on packing d-Dimensional Cubes

Our algorithm can be easily extended to the problem of packing d-dimensional cubes into a unit d-dimensional cubic frame so as to maximize the total volume of the cubes packed. As in the 2-dimensional case, we partition the set of cubes into two sets, L and S, containing large and small cubes, respectively. Since only a constant number of large cubes can be packed into the frame, we can enumerate all feasible subsets of L that can be packed in the frame in polynomial time. The following generalization of Lemma 1.2.1 can be proved (see also [18]).

Lemma 1.2.8. *Let S be any set of d-dimensional cubes S_i with sizes $s_i \leq \varepsilon^{\beta}$, and let $[0,a_1] \times [0,a_2] \times \cdots \times [0,a_d]$ ($a_i \in [0,1]$) be a parallelepiped. The generalization of the NFIH heuristic to d dimensions outputs a packing of a subset of S whose volume is at least* $\min\{volume(S), (a_1 - \varepsilon^{\beta})(a_2 - \varepsilon^{\beta}) \cdots (a_d - \varepsilon^{\beta})\}$.

This lemma shows that the generalization of NFIH to d dimensions can be used to pack the small cubes in the empty space left by a tight packing of the large cubes, so that the total empty space left is only an ε fraction of the total volume of the frame.

1.3 CONCLUDING REMARKS

In this chapter we consider the version of the storage packing problem, where we pack the squares with weights into a unit size square frame. We present an algorithm for the special case of the problem, in which the squares' weights are equal to their areas, i.e. we are interested in covering the maximum area of a unit square by a subset of squares. The algorithm we present finds a subset of squares and it's packing into the unit size square frame with area at least $(1-\varepsilon)$OPT. The first natural question is whether it is possible to extend this result to the more general case of packing rectangles. We think that this can be done. The second natural and not less interesting question is to try to extend our result to the case of packing squares with arbitrary weights. In this case the problem becomes not trivial, that is why we would like to investigate how the restrictions on resources can influence the complexity of the problem. As a result, our next step is to address the resource augmentation version of the storage packing problem.

CHAPTER 2

ON PACKING RECTANGLES WITH RESOURCE AUGMENTATION: MAXIMIZING THE TOTAL WEIGHT

2.1 INTRODUCTION

In this chapter we continue to study the storage packing problem. It would be natural to extend the result from Chapter 1 for packing squares with areas equal to weights to the more general case of packing rectangles with arbitrary weights. Here we address a version of the storage packing problem, in which rectangles with weights are packed into a unit size square region so as to maximize the total weight of the packed rectangles. More precisely, we are given a set R of n rectangles, R_i ($i = 1, \ldots, n$) with widths $a_i \in (0,1]$, heights $b_i \in (0,1]$, and weights $w_i \geq 0$. For a given subset $R' \subseteq R$, a *packing* of R' into a unit size square frame $[0,1] \times [0,1]$ is a positioning of the rectangles of R' within the frame such that they have disjoint interiors. The goal is to find a subset $R' \subseteq R$, and a packing of R' within $[0,1] \times [0,1]$ of maximum weight, $\sum_{R_i \in R'} w_i$.

This problem is known to be strongly NP-hard even for the restricted case of packing squares with identical weights [62]. Hence, it is very unlikely that any polynomial time algorithm for the problem exists, and so, we look for efficient heuristics with good performance guarantees. Now we try a different approach: We want to investigate how the restrictions on resources can influence the approximation property of the problem. In particular, we consider the so-called resource augmentation version of the storage packing problem, that is, we allow the length of the unit square region where the rectangles are to be packed to be increased by some small value. Our main result is this:

Theorem 2.1.1. *For any set R of n rectangles and any accuracy $\varepsilon > 0$, there is an algorithm W_ε which finds a subset of R and its packing within an augmented unit square frame, $[0,1+3\varepsilon]\times[0,1+3\varepsilon]$, with weight*

$$W_\varepsilon(R) \geq (1-\varepsilon)\mathrm{OPT},$$

where OPT *is the maximum weight that can be obtained by packing any subset of R into a unit size square frame $[0,1]\times[0,1]$. The running time of W_ε is polynomial in n for fixed ε.*

We note that the algorithm of Correa and Kenyon [18] for packing a set of rectangles into the minimum number of square bins of size $1+\varepsilon$ can not be directly used to prove Theorem 2.1.1 because (i) the algorithm in [18] does not consider rectangles with weights, and (ii) in the storage packing problem not all rectangles need to be packed. If we can find a set of rectangles of nearly maximum weight and which can be packed into a unit square frame, then we could use the algorithm in [18] to find such a packing. The problem of finding this set of rectangles is not a simple one, though. We show how to find in polynomial time a set of rectangles of nearly optimum weight that can be packed into a square frame of size $1+\varepsilon$. This is enough to prove the theorem.

To simplify the presentation of results, we first address the special case of the problem when all rectangles to be packed are squares. Presenting the algorithm for this simpler problem will help to understand the solution for the more complex problem of packing rectangles. Specifically, we present an algorithm A_ε which given a set of squares L finds a subset of L and its packing into the augmented unit square $[0,1+\varepsilon]\times[0,1+\varepsilon]$ with weight

$$A_\varepsilon(L) \geq (1-\varepsilon)\mathrm{OPT},$$

where OPT is the maximum weight that can be achieved by packing any subset of L in the original unit square region $[0,1]\times[0,1]$. The running time of A_ε is polynomial in n for fixed ε. This result can be extended to the case of packing d-dimensional cubes into a d-dimensional cube of size $1+\varepsilon$, for $d \geq 2$.

Our algorithms combine several known approximation techniques used for knapsack problems, strip packing, and scheduling problems. Our algorithm for packing squares is based on a few simple ideas and, contrasting to recent algorithms for packing problems [10, 18, 51, 57], it does not use linear programming. Since the problem for packing squares is a special case of that of packing rectangles, our algorithm is simpler and more efficient that the algorithm in [18]. The algorithm deals separately with squares of different sizes. This idea has been used before to solve other problems [42, 82]. We partition the squares into two sets formed by large and small squares, respectively. The sets are chosen so that only $O(1)$ large squares can be packed in the unit square frame. We augment the size of the frame to $1+\varepsilon$, and discretize the set of possible positions for the large squares in a packing. This allows us to enumerate all possible packings of the large squares. For each one of these packing we try to fill with small squares the empty spaces left by the large squares. To do this we solve a knapsack problem to select the small squares to be packed, and use a variation of the Next-Fit-Decreasing-Height heuristic to place them (see Section 2.2.1). Among all packings found we select one with the maximum weight, which must be at least $(1-\varepsilon)\mathrm{OPT}$.

For the problem of packing rectangles we need to make a more complex partition, separating the rectangles into four groups: $\mathcal{L}$,$\mathcal{H}$,$\mathcal{V}$, and $\mathcal{S}$. Sets $\mathcal{L}$ and $\mathcal{S}$ contain rectangles with, respectively, large and small widths and heights. These are treated in a similar way as above. The other two sets, $\mathcal{H}$ and $\mathcal{V}$, contain wide and short (i.e. horizontal), and narrow and tall (i.e. vertical) rectangles, respectively. To pack these rectangles we first round their sizes and group them, so they form larger rectangles. These grouped rectangles are then packed by solving a fractional strip packing problem.

Even though, the running times of both algorithms A_ε and W_ε are polynomial in n for fixed ε, they are exponential in $1/\varepsilon$. Therefore, our results are primarily of theoretical importance.

In Section 2.2 we describe our algorithm for packing squares. In Section 2.3 we describe an algorithm for packing a set of rectangles into an augmented square frame and we give a proof for Theorem 2.1.1. Finally, in the last section we give some concluding remarks.

2.2 ALGORITHM FOR PACKING SQUARES

In this section we present an algorithm for packing squares into a unit size square frame so as to maximize the total weight of the packed squares. More precisely, we are given a set Q of n squares S_i $(i = 1, \ldots, n)$ with side lengths $s_i \in (0, 1]$ and positive weights $w_i \in \mathbb{Z}_+$. For a subset $Q' \subseteq Q$, a *packing* of Q' into the unit square is a positioning of the squares Q' within the frame $[0, 1] \times [0, 1]$ such that they have disjoint interiors. The goal is to find a subset $Q' \subseteq Q$ and its packing into the unit square, of maximum weight, $\sum_{S_i \in Q'} w_i$.

For a subset of squares $Q' \subseteq Q$, we use $weight(Q')$ and $area(Q')$ to denote the weight, $\sum_{S_i \in Q'} w_i$, and area, $\sum_{S_i \in Q'} s_i \cdot s_i$, of Q'. In addition, we use Q^{opt} to denote an optimal subset of Q that can be packed in the unit square $[0, 1] \times [0, 1]$. So,

$$weight(Q^{opt}) = \text{OPT and } area(Q^{opt}) \leq 1.$$

Throughout the chapter we also assume that $\varepsilon \in (0, 1/4)$ and the value of $1/\varepsilon$ is integral.

Naive approach. There is a natural two-step approach that could be used for our problem: first, use a knapsack FPTAS with accuracy $\delta \in (0, \varepsilon]$ to find a set Q' of squares of total area at most 1 and maximum weight, and then apply one of the known algorithms to produce a packing of those squares inside a square region of minimum area.

This approach approximates the optimum weight quite well. However, the approach fails in the sense that the augmented square cannot be of size arbitrarily close to the unit one. Consider the following example. Let $\varepsilon \in (0, 1]$, and L be a set consisting of two large squares S_1, S_2 with side lengths $s_1, s_2 = 1/\sqrt{2}$ and weights $p_1 = p - \varepsilon p$, $p_2 = \varepsilon p$, and n^2 small squares S_i ($i = 3, \ldots, n^2+2$) with side lengths $s_i = 1/(\sqrt{2}n)$ and weights $w_i = \varepsilon p/n^2$, for some positive value p. For all small squares, their total area is

$$\sum_{i=3}^{n^2+2} (s_i)^2 = n^2/(2n^2) = \frac{1}{2}$$

and their total weight is

$$\sum_{i=3}^{n^2+2} w_i = n^2 \cdot (\varepsilon p/n^2) = \varepsilon p.$$

The corresponding knapsack problem for this set of squares can be formulated as:

$$\begin{array}{lrcll} \text{Maximize} & \sum_{i=1}^{n^2+2} w_i x_i & & & \\ & & & & \\ \text{subject to} & \sum_{i=1}^{n^2+2} (s_i)^2 x_i & \leq & 1, & \\ & x_i & \in & \{0,1\} & \text{for all } i = 1, \ldots, n^2+2. \end{array}$$

There are two optimum solutions for this knapsack problem:

(a) the two large squares S_1, S_2 are chosen; their area is $(s_1)^2 + (s_2)^2 = 1$ and their weight is $(p_1 + p_2) = p - \varepsilon p + \varepsilon p = p$, and

(b) the large square S_1 and all the small squares S_i ($i = 3, \ldots, n^2+2$) are chosen; their area is $\sum_{i=3}^{n^2+2} (s_i)^2 + (s_1)^2 = \frac{1}{2} + \frac{1}{(\sqrt{2})^2} = 1$ and their weight is $\sum_{i=3}^{n^2+2} w_i + p_1 = p - \varepsilon p + \varepsilon p = p$.

If we use an FPTAS for the knapsack problem with accuracy $\delta \leq \varepsilon/2$, there is no guarantee that a solution of the form (b) is produced. If solution (a) is obtained, then its two large squares can only be packed into a square of side length $\sqrt{2}$ (since $\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} = \frac{2}{\sqrt{2}} = \sqrt{2}$). This is a large augmentation of the unit square, see Fig. 2.1. Hence, by using this naive approach we cannot guarantee that the augmented square has size arbitrarily close to 1. Contrasting to this approach our algorithm, for any set Q of n squares and any fixed value $\varepsilon > 0$, finds a subset of

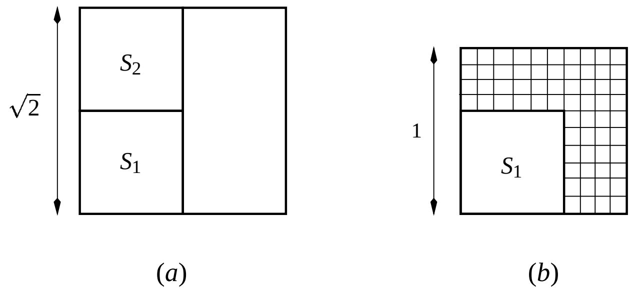


Figure 2.1: Example.

Q and its packing into the augmented unit square $[0,1+\varepsilon]\times[0,1+\varepsilon]$ with weight at least $(1-\varepsilon)\mathrm{OPT}$, where OPT is the maximum weight that can be achieved by packing any subset of Q in the original unit square region $[0,1]\times[0,1]$.

2.2.1 The NFDH Heuristic

We consider first the following special case of the square packing problem: given a subset $Q'\subseteq Q$ of squares with side lengths at most ε^2, and a rectangle $[0,a]\times[0,b]$ ($a,b\in[0,1]$) such that $area(Q')\leq ab$, pack the squares of Q' into the augmented rectangle $[0,a+\varepsilon^2]\times[0,b+\varepsilon^2]$.

To solve this problem, we sort the squares of Q' non-increasingly by side lengths. Then, we put the squares into the rectangle $[0,a]\times[0,b]$ by using the Next-Fit-Decreasing-Height (NFDH) heuristic; this packs the squares into a sequence of sublevels. The first sublevel is the bottom of the rectangle. Each subsequent sublevel is defined by a horizontal line drawn at the top of the largest square placed on the previous sublevel. In each sublevel, squares are packed in a left-justified manner until their total width is at least a. At that moment, the current sublevel is closed, a new sublevel is started and the packing proceeds as above. For an illustration see Fig. 2.2.

We will use the following simple result, which can be directly derived from results in [15, 65], but for completeness we include a proof.

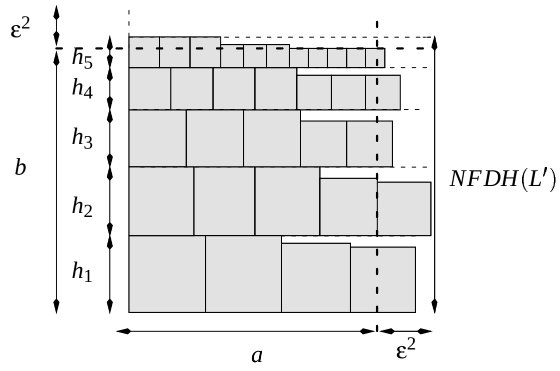


Figure 2.2: NFDH for small squares.

Lemma 2.2.1. *Let $Q' \subseteq Q$ be any subset of squares with side lengths at most ε^2, ordered non-increasingly by side lengths, and let $[0,a] \times [0,b]$ $(a,b \in [0,1])$ be a rectangle such that $area(Q') \leq ab$. Then, the NFDH heuristic outputs a packing of Q' in the augmented rectangle $[0,a+\varepsilon^2) \times [0,b+\varepsilon^2]$.*

Proof. Let q be the number of sublevels. Let h_i be the height of the first square on the ith sublevel. Since NFDH packs the squares of Q' on sublevels in order of non-increasing side lengths, the height of the packing is

$$H = \sum_{i=1}^{q} h_i.$$

Since the side of any square is at most ε^2, then $\varepsilon^2 \geq h_1 \geq h_2 \geq \ldots \geq h_q > 0$. Furthermore, the total width of the squares on each sublevel (except, maybe, the last) is at least a and at most $a+\varepsilon^2$. Then, the total area of the squares on the ith sublevel ($i = 1,\ldots,q-1$) is at least $h_{i+1} \cdot a$. Assume that the value of H is larger than $b+\varepsilon^2$. Then, the area covered by squares would be at least

$$\begin{aligned}
&\sum_{i=1}^{q-1} h_{i+1} \cdot a = a \cdot \sum_{i=2}^{q} h_i \\
&= a[H-h_1] > a[(b+\varepsilon^2)-h_1] \quad \text{by assumption } H > b+\varepsilon^2 \\
&= a[b+(\varepsilon^2-h_1)] \geq ab = area(Q') \text{ since } \quad h_1 \leq \varepsilon^2,
\end{aligned}$$

which gives a contradiction.

□

Corollary 2.2.2. *If all squares in Q have side length at most ε^2, then there is an algorithm which finds a subset of Q and its packing in the augmented square* $[0,1+\varepsilon^2]\times[0,1+\varepsilon^2]$ *with weight at least* $(1-\varepsilon)\mathrm{OPT}$. *The running time of the algorithm is polynomial in n and $1/\varepsilon$.*

Proof. By solving a knapsack problem we can find a subset of Q, whose total area is at most 1 and whose weight is at least $(1-\varepsilon)\mathrm{OPT}$. By using NFDH we can pack these squares into the augmented frame $[0,1+\varepsilon^2]\times[0,1+\varepsilon^2]$. □

2.2.2 Partitioning the Squares

Now we consider the case of squares with arbitrary sizes. We define the group $L^{(0)}$ of squares with side lengths in $(\varepsilon^4,1]$, and for $j\in\mathbb{Z}_+$ we define the group $L^{(j)}$ of squares with side lengths in $(\varepsilon^{4^{j+1}},\varepsilon^{4^j}]$. Then,

$$\cup_{j=0}^{\infty}L^{(j)}=Q \;\text{ and }\; L^{(\ell)}\cap L^{(j)}=\emptyset,\text{ for } \ell\neq j.$$

We will use the following simple observation, which also has been made by other researchers in different contexts [10, 18, 42, 82].

Lemma 2.2.3. *There is a group $L^{(k)}$ with $0\le k\le 1/\varepsilon^2-1$ such that its contribution to the optimum is*

$$weight(Q^{opt}\cap L^{(k)})\le\varepsilon^2\mathrm{OPT},$$

where Q^{opt} is an optimal subset of squares.

Proof. Since $L^{(\ell)}\cap L^{(j)}=\emptyset$ for all $\ell\neq j$, then

$$\mathrm{OPT}=weight(Q^{opt})\ge\sum_{j=0}^{1/\varepsilon^2-1}weight(Q^{opt}\cap L^{(j)}).$$

There must exist at least one group $L^{(k)}$ with $0 \leq k \leq 1/\varepsilon^2 - 1$ whose contribution to the weight of the optimal solution is at most the average contribution of the $1/\varepsilon^2$ groups:

$$weight(L^{(k)} \cap Q^{opt}) \leq [\sum_{j=0}^{1/\varepsilon^2-1} weight(Q^{opt} \cap L^{(j)})]/(1/\varepsilon^2) \leq \varepsilon^2 \mathrm{OPT}.$$

□

We drop the squares in this group $L^{(k)}$ of low weight from consideration. Then, an optimal packing for $Q \setminus L^{(k)}$ has weight at least $(1-\varepsilon^2)$OPT, i.e. this makes a loss of at most a factor of ε^2 in the optimum. We partition the squares in $Q \setminus L^{(k)}$ into two groups: $\mathcal{L} = \cup_{j \leq k-1} L^{(j)}$ and $\mathcal{S} = \cup_{j \geq k+1} L^{(j)}$. The squares in $\mathcal{L}$ and $\mathcal{S}$ are called large and small, respectively.

Corollary 2.2.4. *Let $\Delta = \varepsilon^{4^k}$, where k is as defined above. The side length of any large square is larger than Δ and the side length of any small square is at most $\varepsilon^4\Delta$. Moreover,*

$$weight(Q^{opt} \cap [\mathcal{L} \cup \mathcal{S}]) \geq (1-\varepsilon^2)\mathrm{OPT}.$$

2.2.3 Large Squares

We say that a subset of large squares is *feasible* if it can be packed into the unit square frame. We can prove the following result.

Lemma 2.2.5. *In $O(n^{O(1)})$ time we can find the set $FEASIBLE$ consisting of all subsets of at most $1/\Delta^2$ large squares from $\mathcal{L}$. Any feasible set of large squares belongs to $FEASIBLE$. Moreover, the optimal set of large squares $\mathcal{L} \cap Q^{opt}$ is feasible and, hence, it also belongs to $FEASIBLE$.*

Proof. By definition, any feasible set of large squares can be packed into the unit square, i.e. into a square area of size 1. The area of any large square is at least Δ^2, hence, there are at most $1/\Delta^2$ large squares in any feasible set. There are at most n squares in $\mathcal{L}$, so, there are $O(n^{1/\Delta^2})$ sets in $FEASIBLE$. Notice that the optimal set $\mathcal{L} \cap Q^{opt}$ of large squares is also feasible, hence, it must belong to $FEASIBLE$. □

Packing large squares. Even if we could find the optimal set of large squares, we would still need to determine how to pack them in the square frame. We enlarge the size of the unit square so that there is a packing for the large squares such that the positions of their lower left corners belong to a finite set of discrete points.

Consider a packing of a subset of large squares in the frame $[0,1] \times [0,1]$. In this packing, increase the size of each large square by a factor $1+\varepsilon^2$. This increases the size of the enclosing frame by the same factor. Then, without reducing the size of the frame, reduce the size of every large square back to its original value. See Fig. 2.3 for an illustration of this process.

The side length of any large square is at least Δ. So, for each large square we now have an "induced space" where we can move the square up to a distance $\varepsilon^2\Delta$ vertically or horizontally, without increasing the area of the packing. Since $\varepsilon^2\Delta > \varepsilon^3\Delta$, we can move all large squares such that each one of them has its lower left corner in the following set

$$CORNER = \{(x,y) | x = \ell \cdot (\varepsilon^3\Delta), y = p \cdot (\varepsilon^3\Delta) \text{ and } \ell, p = 1, 2, \ldots, \frac{1+\varepsilon^2-\Delta}{\varepsilon^3\Delta}\}.$$

By discretizing the positions of the large squares we reduce to a constant the number of different packings for the large squares in a feasible set.

2.2.4 Small Squares

Let $\mathcal{L}' \subseteq \mathcal{L}$ be any feasible set of large squares. The *complement* of $\mathcal{L}'$, denoted $COM(\mathcal{L}')$, is the set of small squares which is selected by an FPTAS [55] for the knapsack problem with accuracy ε^2, knapsack capacity $1 - area(\mathcal{L}')$, and set of

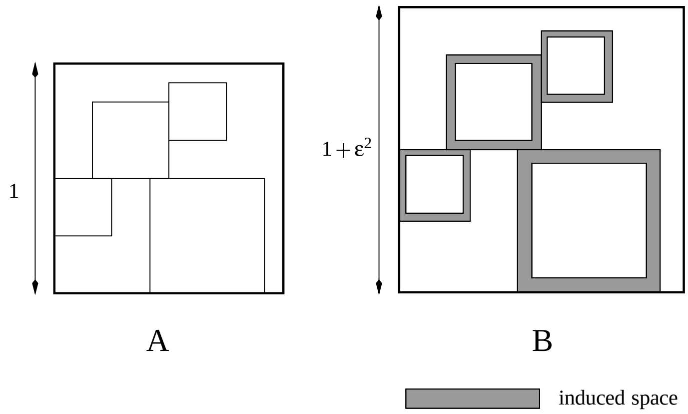


Figure 2.3: Increasing and decreasing the sizes of the large squares.

items $\mathcal{S}$; each item $S_i \in \mathcal{S}$ has size $(s_i)^2$ and weight w_i. We can prove the following simple result.

Lemma 2.2.6. *For the optimal set $Q^{opt} \cap \mathcal{L}$ of large squares, its complement $COM(Q^{opt} \cap \mathcal{L})$ has total area at most*

$$1 - area(L^{opt} \cap \mathcal{L})$$

and weight at least

$$(1-\varepsilon^2) weight(Q^{opt} \cap \mathcal{S}).$$

Proof. The area of Q^{opt} is at most 1, hence, $Q^{opt} \cap \mathcal{S}$ is a feasible solution for the instance of the knapsack problem with knapsack capacity $1 - area(Q^{opt} \cap \mathcal{L})$ and set of items $\mathcal{S}$. So, the optimum weight of this instance is at least $weight(Q^{opt} \cap \mathcal{S})$ and the FPTAS finds a solution of weight at least $(1-\varepsilon^2) weight(Q^{opt} \cap \mathcal{S})$. $\square$

Placing small squares: The modified NFDH. Assume that we have a packing of some feasible set $\mathcal{L}' \subseteq \mathcal{L}$ of large squares in the augmented frame $[0, 1+\varepsilon^2] \times [0, 1+\varepsilon^2]$. By solving a knapsack problem, we can find its complement $COM(\mathcal{L}')$. Our next task is to place the small squares from $COM(\mathcal{L}')$ in the slightly larger frame $[0, 1+\varepsilon] \times [0, 1+\varepsilon]$.

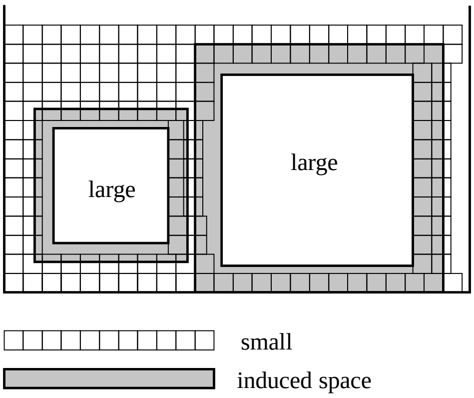


Figure 2.4: Packing the small squares.

We pack the small squares in the empty space left by the large squares using the modified NFDH heuristic from [15]: Pack the squares on sublevels, creating sublevels in a bottom up manner and filling each one of them from left to right. On each sublevel, if the next small square overlaps with a large square, we place it immediately after the right boundary of the large square. For an illustration see Fig. 2.4. We cannot pack small squares within the space occupied by the large squares, but we can pack them inside the "induced space" around the large squares. We can prove the following result.

Lemma 2.2.7. *For any feasible set $\mathcal{L}' \subseteq \mathcal{L}$ of large squares packed in the augmented frame $[0,1+\varepsilon^2]\times[0,1+\varepsilon^2]$, the modified NFDH heuristic outputs a packing of $\mathcal{L}'$ and the small squares from its complement $COM(\mathcal{L}')$ in the augmented frame $[0,1+\varepsilon]\times[0,1+\varepsilon]$.*

Proof. Since we use the modified NFDH heuristic, in each sublevel at most one small square can cross the right border of the square $[0,1+\varepsilon^2]\times[0,1+\varepsilon^2]$. Any small square has side at most $\varepsilon^4\Delta < \varepsilon^2$, hence, the total width of the packing is at most $(1+\varepsilon^2)+\varepsilon^2 < 1+\varepsilon$, for $\varepsilon < 1/4$.

Now we show that the height of the packing cannot be larger than $1+\varepsilon$. We follow the ideas of Lemma 2.2.1. Let H be the height of the packing. Let h_i ($i=1,\ldots,q$) be the height of the first square on the ith sublevel. We assume that H is larger than $1+\varepsilon$ and derive a contradiction. Consider one large square of side length s_i and all sublevels ℓ that intersect it. The maximum distance from the large square's boundary to the closest small square on a sublevel ℓ cannot be larger than $\varepsilon^4\Delta$ (otherwise, a small square could be added on that sublevel). Hence, the maximum area not covered by small squares around, and including this large square, is at most $(s_i+2\varepsilon^4\Delta)^2$.

Summing, over all large squares, we get that the area not covered by small squares is at most

$$\sum_{s_i\in\mathcal{L}'}(s_i+2\varepsilon^4\Delta)^2.$$

Notice that our packing for small squares goes further than point $1+\varepsilon^2$ in width, and $H=\sum_{i=1}^{q}h_i$. Then, as in Lemma 2.2.1, the area covered by the squares from $COM(\mathcal{L}')$ is

$$\begin{aligned}
AREA &\geq \sum_{i=1}^{q-1} h_{i+1}\cdot(1+\varepsilon^2)-\sum_{s_i\in\mathcal{L}'}(s_i+2\varepsilon^4\Delta)^2\\
&=(H-h_1)\cdot(1+\varepsilon^2)-\sum_{s_i\in\mathcal{L}'}(s_i+2\varepsilon^4\Delta)^2\\
&>(1+\varepsilon^2)^2-\sum_{s_i\in\mathcal{L}'}[(s_i^2+4s_i\varepsilon^4\Delta+(2\varepsilon^4\Delta)^2)] \quad \text{since } H>1+\varepsilon \text{ and } h_1<\varepsilon^4\\
&\geq [1-\sum_{s_i\in\mathcal{L}'}s_i^2]+2\varepsilon^2[1-2\varepsilon^2\Delta\sum_{s_i\in\mathcal{L}'}s_i]+\varepsilon^4[1-4\Delta^2\varepsilon^4|\mathcal{L}'|].
\end{aligned} \tag{2.1}$$

Since $s_i\geq\Delta$ and $\varepsilon<1/4$, then

$$1-2\varepsilon^2\Delta\sum_{s_i\in\mathcal{L}'}s_i>1-\sum_{s_i\in\mathcal{L}'}s_i^2\geq 0.$$

From $|\mathcal{L}'|\leq 1/\Delta^2$ we also get

$$1-4\Delta^2\varepsilon^4|\mathcal{L}'|\geq 1-4\varepsilon^4\geq 0.$$

Combining the above inequalities, we get

$$AREA>1-\sum_{s_i\in\mathcal{L}'}s_i^2=area(COM(\mathcal{L}')).$$

This gives a contradiction. Hence, the value of H is at most $1+\varepsilon$. $\square$

2.2.5 The Algorithm

ALGORITHM A_ε:

Input: A set of squares Q, accuracy $\varepsilon > 0$.

Output: A packing of a subset of Q in $[0,1+\varepsilon]\times[0,1+\varepsilon]$.

1. For each $k \in \{0,1\ldots,1/\varepsilon^2\}$, form the group $L^{(k)}$ as described above.
   (a) Let $\Delta := \varepsilon^{4^k}$.
   (b) Split $Q\setminus L^{(k)}$ into $\mathcal{L}$ and $\mathcal{S}$, the sets of large and small squares with side lengths larger than Δ and at most $\varepsilon^4\Delta$, respectively.
   (c) Compute the set $FEASIBLE$ containing all subsets of $\mathcal{L}$ with at most $1/\Delta^2$ large squares.
   (d) For every set $\mathcal{L}' \in FEASIBLE$ find its complement $\mathcal{S}' := COM(\mathcal{L}')$ by solving a knapsack problem. For each packing of $\mathcal{L}'$ in the augmented square $[0,1+\varepsilon^2]\times[0,1+\varepsilon^2]$ such that every large square in $\mathcal{L}'$ has its lower left corner in a point of $CORNER$:
      - Use the modified NFDH to pack the small squares $\mathcal{S}'$ in the augmented unit square $[0,1+\varepsilon]\times[0,1+\varepsilon]$.
2. Among all packings produced, output one with the largest weight.

Theorem 2.2.8. *For any set Q of n squares and any fixed value $\varepsilon > 0$, there exists an algorithm A_ε which finds a subset of Q and its packing into the augmented unit square $[0,1+\varepsilon]\times[0,1+\varepsilon]$ with weight*

$$A_\varepsilon(Q) \geq (1-\varepsilon)\mathrm{OPT},$$

where OPT *is the maximum weight that can be achieved by packing any subset of Q in the original unit square region $[0,1]\times[0,1]$. The running time of A_ε is*

$$O\left(\frac{n^2}{\varepsilon^3}\left(\frac{n}{\varepsilon^8\Delta^2}\right)^{1/\Delta^2}\right),$$

where $\Delta = \varepsilon^{4^{1/\varepsilon^2}}$.

Proof. By Lemma 2.2.7 algorithm A_ε produces a packing in the augmented square $[0,1+\varepsilon]\times[0,1+\varepsilon]$. Hence, we only need to compute the weight of the packing chosen in Step 2. The optimal set of large squares $Q^{opt}\cap\mathcal{L}$ belongs to $FEASIBLE$, and hence, there exists a packing of these squares in the augmented square $[0,1+\varepsilon^2]\times[0,1+\varepsilon^2]$ such that each large square has its lower left corner in a point of $CORNER$.

Since algorithm A_ε checks all possible packings, it will find one for $Q^{opt}\cap\mathcal{L}$. Next, A_ε finds the complement $COM(Q^{opt}\cap\mathcal{L})$ and packs it using the modified NFDH. The weight of the packing output by the algorithm is

$$\begin{aligned}
A_\varepsilon(Q) &\geq weight(Q^{opt}\cap\mathcal{L}) + weight(COM(Q^{opt}\cap\mathcal{L}))\\
&\geq weight(Q^{opt}\cap\mathcal{L}) + (1-\varepsilon^2)weight(Q^{opt}\cap\mathcal{S}) \quad \text{by Lemma 2.2.6}\\
&\geq (1-\varepsilon^2)weight(Q^{opt}\cap[\mathcal{L}\cup\mathcal{S}])\\
&\geq (1-\varepsilon^2)[(1-\varepsilon^2)weight(Q^{opt})] \quad \text{from Corollary 2.2.4}\\
&\geq (1-\varepsilon)\mathrm{OPT}.
\end{aligned}$$

We know that any set of large squares from $FEASIBLE$ consists of at most $(1/\Delta^2)$ squares. Hence, $FEASIBLE$ can be computed in $O(n^{1/\Delta^2})$ time, and we need to do this $1/\varepsilon^2$ times (once for each value of k, see Step 1 of the algorithm). Since $|CORNER| = (\frac{1+\varepsilon^2-\Delta}{\varepsilon^3\Delta})^2 \leq \frac{1}{\varepsilon^8\Delta^2}$, the algorithm computes at most $\left(\frac{1}{\varepsilon^8\Delta^2}\right)^{1/\Delta^2}$ packings of large squares in the augmented square $[0,1+\varepsilon^2]\times[0,1+\varepsilon^2]$. The running time of the basic-FPTAS in [55] for the knapsack problem is $O(n^2\cdot 1/\varepsilon)$ (the different versions of FPTAS can be found in [55]). The modified NFDH algorithm runs in $O(n\log n)$ time. Combining all together, we get that the running time of the algorithm is

$$O\left(\left[\frac{(n^{1/\Delta^2})}{\varepsilon^2}\right]\cdot\left[\left(\frac{1}{\varepsilon^8\Delta^2}\right)^{1/\Delta^2}\right]\cdot\left[(n^2\cdot 1/\varepsilon))+(n\log n)\right]\right).$$

Simplifying, we find that the running time of the overall algorithm is bounded by

$$O\left(\frac{n^2}{\varepsilon^3}\left(\frac{n}{\varepsilon^8\Delta^2}\right)^{1/\Delta^2}\right),$$

where $\Delta = \varepsilon^{4^{1/\varepsilon^2}}$. $\square$

2.2.6 Remark on packing d-Dimensional Cubes

Our algorithm can be easily extended to the problem of packing d-dimensional cubes into a unit d-dimensional cubic frame so as to maximize the total weight of the cubes packed. As in the 2-dimensional case, we partition the set of cubes into two sets $\mathcal{L}$ and $\mathcal{S}$ containing large and small cubes, respectively. Since only a constant number of large cubes can be packed into the frame, we can enumerate all feasible subsets of $\mathcal{L}$ that can be packed in the augmented cubic frame of size $1+\varepsilon^2$ in polynomial time. The following generalization of Lemma 2.2.1 can be proved (see also [18]).

Lemma 2.2.9. *Let $Q' \subseteq Q$ be any subset of d-dimensional cubes with side lengths at most ε^2, ordered by non-increasing side lengths, and let $[0,a_1]\times[0,a_2]\times\cdots\times[0,a_d]$ ($a_i \in [0,1]$) be a parallelepiped, such that $area(Q') \le a_1 \times a_2 \ldots \times a_d$. Then, the generalization of the NFDH heuristic to d dimensions outputs a packing of Q' in the augmented parallelepiped $[0,a_1+\varepsilon^2]\times[0,a_2+\varepsilon^2]\times\cdots\times[0,a_d+\varepsilon^2]$.*

This lemma shows that the generalization of NFDH to d dimensions can be used to pack the small cubes in the empty spaces left by a packing of the large cubes into the augmented cubic frame. Then, we can prove that the generalization of the modified NFDH heuristic to d dimensions outputs a packing of $\mathcal{L}'$ and the small cubes from its complement $COM(\mathcal{L}')$ in the augmented cubic frame of size $1+\varepsilon$. Among all packings found we select one with the maximum weight, which must be at least $(1-\varepsilon)$OPT.

2.3 Algorithm for Packing Rectangles

Let R be a set of n rectangles, R_i $(i=1,\ldots,n)$ with widths $a_i \in (0,1]$, heights $b_i \in (0,1]$, and weights $w_i \ge 0$. The goal is to find a subset $R' \subseteq R$, and a packing of R' within the frame $[0,1]\times[0,1]$ of maximum weight, $\sum_{R_i \in R'} w_i$.

We partition the rectangles R into four sets: $\mathcal{L}, \mathcal{H}, \mathcal{V}$, and $\mathcal{S}$. The rectangles in $\mathcal{L}$ have large widths and heights, so only $O(1)$ of them can be packed in the unit square frame. The rectangles in $\mathcal{H}$ ($\mathcal{V}$) have large width (height). We round the sizes of these rectangles in order to reduce the number of distinct widths and heights. Then, we use enumeration and a fractional strip-packing algorithm to select the best subsets of $\mathcal{H}$ and $\mathcal{V}$ to include in our solution. The rectangles in $\mathcal{S}$ have very small width and height, so as soon as we have selected near-optimal subsets of rectangles from $\mathcal{L} \cup \mathcal{H} \cup \mathcal{V}$ we add rectangles from $\mathcal{S}$ to the set of rectangles to be packed in a greedy way. Once we have selected the set of rectangles to be packed into the frame, we use a modification of the algorithm of Correa and Kenyon [18] to pack them.

For a subset of rectangles $R' \subseteq R$, we use $weight(R')$ to denote its weight, $\sum_{R_i \in R'} w_i$, and $area(R')$ to denote its area, $\sum_{R_i \in R'} a_i b_i$. In addition, we use R^{opt} to denote an optimal subset of R that can be packed into the unit square frame $[0,1] \times [0,1]$. So,

$$weight(R^{opt}) = \text{OPT and } area(R^{opt}) \leq 1.$$

2.3.1 Partitioning the Rectangles

We slightly modify the definition of the groups $L^{(j)}$ given above to account for the fact that now the width and height of a rectangle might be different. We define the group $L^{(0)}$ of rectangles $R_i \in R$ with widths $a_i \in (\varepsilon^4, 1]$ and/or heights $b_i \in (\varepsilon^4, 1]$. For $j \in Z_+$ we define the group $L^{(j)}$ of rectangles R_i with either widths $a_i \in (\varepsilon^{4^{j+1}}, \varepsilon^{4^j}]$ or heights $b_i \in (\varepsilon^{4^{j+1}}, \varepsilon^{4^j}]$. One can see that each rectangle belongs to at most 2 groups.

Lemma 2.3.1. *There is a group $L^{(k)}$ with $0 \leq k \leq 2/\varepsilon^2 - 1$ such that*

$$weight(L^{(k)} \cap R^{opt}) \leq \varepsilon^2 \cdot \text{OPT},$$

where R^{opt} is the subset of rectangles selected by an optimum solution.

Proof. The proof is very similar to the proof of Lemma 2.2.3 □

We again drop the rectangles in group $L^{(k)}$, as described in Lemma 2.3.1, from consideration. Then, an optimal packing for $R^{opt} \setminus L^{(k)}$ must have weight at least

$(1-\varepsilon^2)$OPT. However, now we partition the rectangles of R into four groups according to their side lengths, as follows. Let $\Delta = \varepsilon^{4^k}$.

$$\begin{aligned}
\mathcal{L} &= \{R_i \mid a_i > \Delta \text{ and } b_i > \Delta\} \\
\mathcal{S} &= \{R_i \mid a_i \leq \varepsilon^4\Delta \text{ and } b_i \leq \varepsilon^4\Delta\} \\
\mathcal{H} &= \{R_i \mid a_i > \Delta \text{ and } b_i \leq \varepsilon^4\Delta\} \\
\mathcal{V} &= \{R_i \mid a_i \leq \varepsilon^4\Delta \text{ and } b_i > \Delta\}
\end{aligned}$$

The rectangles in $\mathcal{L}$, $\mathcal{S}$, $\mathcal{H}$ and $\mathcal{V}$ are called large, small, horizontal and vertical, respectively.

Lemma 2.3.2. *For $0 < \varepsilon < 1/2$ the subset $R^{opt} \setminus L^{(k)}$ of rectangles can be packed within the frame $[0, 1+\varepsilon] \times [0, 1+\varepsilon]$ in such a way that*

- *each rectangle $R_i \in \mathcal{H} \cup \mathcal{L}$ is positioned so that its lower left corner is at an x-coordinate that is a multiple of $\varepsilon^2\Delta$,*
- *each rectangle $R_i \in \mathcal{V} \cup \mathcal{L}$ is positioned so that its lower left corner is at a y-coordinate that is a multiple of $\varepsilon^2\Delta$,*

Furthermore, any width $a_i > \Delta$ or height $b_i > \Delta$ can be rounded up to the nearest multiple of $\varepsilon^2\Delta$ without affecting the feasibility of the packing, i.e. (i) for each $R_i \in \mathcal{L}$, both, a_i and b_i can be rounded up, (ii) for each $R_i \in \mathcal{H}$, only a_i can be rounded, and (iii) for each $R_i \in \mathcal{V}$, only b_i can be rounded.

Proof. Increase the size of every rectangle in $\mathcal{L} \cup \mathcal{H} \cup \mathcal{V}$ by a factor $1+\varepsilon$. These enlarged rectangles can be packed in a frame of size $1+\varepsilon$. Now shrink the rectangles back to their original sizes to create the "induced spaces" as before. Shift each rectangle inside its induced space so that it is positioned as indicated in the lemma. Note that each rectangle needs to be shifted vertically and/or horizontally at most a distance $\varepsilon^2\Delta$. Finally, round each side length larger than Δ to the nearest multiple of $\varepsilon^2\Delta$. Since each rectangle can be shifted inside its induced space vertically or horizontally by a distance $\varepsilon\Delta$, and since $2\varepsilon^2\Delta < \varepsilon\Delta$ for all $0 < \varepsilon < 1/2$, then the enlarged rectangles fit in a frame of size $1+\varepsilon$. □

Selecting the large rectangles. As before, we say that a subset of large rectangles is feasible if they can be packed in the unit frame. We define the set $FEASIBLE$ consisting of all subsets of at most $1/\Delta^2$ large rectangles. Observe that the optimal set of large rectangles $\mathcal{L} \cap R^{opt} \in FEASIBLE$. As we showed above $FEASIBLE$ can be computed in $O(n^{1/\Delta^2})$ time.

Selecting the horizontal rectangles. Recall that for each rectangle $R_i \in \mathcal{H}$, its width, $a_i \in (\Delta, 1]$ was rounded up to a multiple of $\varepsilon^2\Delta$. Hence, there are at most $\alpha = 1/(\varepsilon^2\Delta)$ distinct widths, $\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_\alpha$, in $\mathcal{H}$. We use $\mathcal{H}(\bar{a}_q)$ to denote the subset of $\mathcal{H}$ consisting of all rectangles with width $\bar{a}_q$. Let $\mathcal{H}' \subseteq \mathcal{H}$. We define the *profile* of $\mathcal{H}'$ as an α-tuple $(h'_1, h'_2, \ldots, h'_\alpha)$ such that each entry $h'_q \in (0, 1]$ $(q = 1, \ldots, \alpha)$ is the total height of the rectangles in $\mathcal{H}' \cap \mathcal{H}(\bar{a}_q)$.

Consider the profile $(h^*_1, h^*_2, \ldots, h^*_\alpha)$ of $\mathcal{H} \cap R^{opt}$. Note that if each value h^*_i is rounded up to the nearest multiple of ε/α, this might increase the height of the frame where the rectangles are packed by at most $\alpha(\varepsilon/\alpha) = \varepsilon$. The advantage of doing this, is that the number of possible values for each entry of the profile of $\mathcal{H} \cap R^{opt}$ is only constant, i.e. α/ε, and, the total number of profiles is also constant, $\alpha^{\alpha/\varepsilon}$.

By trying all possible profiles with entries that are multiples of ε/α we ensure to find one that is identical to the rounded profile for $\mathcal{H} \cap R^{opt}$. However, the profile itself does not yield the set of rectangles in $\mathcal{H} \cap R^{opt}$. Fortunately, we do not need to find this set, since (from the algorithms in [18] it can be shown that) any set $\mathcal{H}''$ of rectangles with the same rounded profile as $\mathcal{H} \cap R^{opt}$ can be packed along with $\mathcal{L} \cap R^{opt}$ in a frame of height $1 + \varepsilon$ by solving a fractional strip-packing problem:

- Fix an optimum solution and consider the packing of $\mathcal{L} \cap R^{opt}$ in that solution.

- Trace a grid of size ε/α over the entire square frame and mark those squares of the grid which are (partially) occupied by rectangles from $\mathcal{H} \cap R^{opt}$ in the optimum packing.

- Group marked grid squares that are horizontally adjacent into a strip.

- Let $(h''_1, h''_2, \ldots, h''_\alpha)$ be the profile of $\mathcal{H}''$. The fractional strip packing problem is to fractionally pack rectangles of width $\bar{q}$ and height h''_i into these strips. In this fractional packing problem a rectangle can only be split into rectangles of smaller height and the same width as the original rectangle.

The rectangles from $\mathcal{H}''$ are packed according to the solution of the fractional strip packing problem, but since a rectangle of $\mathcal{H}''$ might not completely fit in a strip, the height of the strips might need to be slightly increased. The total increase in the height of the packing is at most $(\alpha/\varepsilon)\varepsilon^4\Delta = \varepsilon$. (For a more detailed explanation, the reader is referred to [18].)

Thus, we just need to find a set of rectangles from $\mathcal{H}$ with nearly-maximum weight and with the same rounded profile as $\mathcal{H} \cap R^{opt}$. We say that a subset $\mathcal{H}' \subseteq \mathcal{H}$ is *feasible* if

- each entry $h'_q \in (0,1]$ $(q = 1, \ldots, \alpha)$ in the profile of $\mathcal{H}'$ is a multiple of ε/α, and

- each subset $\mathcal{H}' \cap \mathcal{H}(\bar{q})$ $(q = 1, \ldots, \alpha)$ is a $(1-\varepsilon)$-approximate solution of an instance of the knapsack problem where h'_q is the knapsack's capacity and each rectangle $R_i \in \mathcal{H}(\bar{q})$ is an item of size b_i and weight w_i.

Lemma 2.3.3. *In $O(n^2 \cdot 1/\varepsilon)$ time we can find the set $FEASIBLE_{\mathcal{H}}$ consisting of all feasible subsets of $\mathcal{H}$.*

Proof. There are $O(1)$ possible profiles. For each entry in a profile, in order to find a $(1-\varepsilon)$-solution for the corresponding knapsack problem, we can use the FPTAS of [55] with $O(n^2 \cdot 1/\varepsilon)$ running time. □

Selecting the vertical rectangles. We use similar ideas as above to define *profiles* and to find the set $FEASIBLE_{\mathcal{V}}$ consisting of all *feasible* subsets of $\mathcal{V}$. Note that a set $\mathcal{V}'' \subseteq \mathcal{V}$ of rectangles with the same rounded profile as $\mathcal{V} \cap R^{opt}$ can be packed, along with $\mathcal{L} \cap R^{opt}$ and a set $\mathcal{H}'' \subseteq \mathcal{H}$ as described above, in a square frame of size $1+\varepsilon$. To see this, consider a grid as described above and mark in this grid the squares occupied by rectangles from $\mathcal{V} \cap R^{opt}$ in an optimum solution. The rectangles in $\mathcal{V}''$ can be placed in these marked grid squares by solving a fractional strip packing problem as described above. This time the width of the frame needs to be increased to $1+\varepsilon$.

Selecting the small rectangles. Assume that we are given feasible subsets $\mathcal{L}' \in FEASIBLE$, $\mathcal{H}' \in FEASIBLE_{\mathcal{H}}$, $\mathcal{V}' \in FEASIBLE_{\mathcal{V}}$ such that $area(\mathcal{L}' \cup \mathcal{H}' \cup \mathcal{V}') \leq (1+2\varepsilon)^2$ (Recall that the rounding involved in packing the rectangles in $\mathcal{H} \cup \mathcal{V}$ increases the size of the frame of Lemma 2.3.2 to $1+2\varepsilon$). A subset $\mathcal{S}' \subseteq \mathcal{S}$ is feasible for the selection $\mathcal{L}'$,$\mathcal{H}'$, $\mathcal{V}'$, if $\mathcal{S}'$ is a $(1-\varepsilon)$-approximate solution for the instance of the knapsack problem where $(1+2\varepsilon)^2 - area(\mathcal{L}' \cup \mathcal{H}' \cup \mathcal{V}')$ is the knapsack's capacity, and each rectangle $R_i \in S$ is an item of size $a_i b_i$ and weight w_i.

Proposition 2.3.4. *Given sets $\mathcal{L}' \subseteq FEASIBLE, \mathcal{H}' \subseteq FEASIBLE_{\mathcal{H}}$, and $\mathcal{V}' \subseteq FIASIBLE_{\mathcal{V}}$, a feasible subset $\mathcal{S}'$ of $\mathcal{S}$ can be found in $O(n^2 \cdot 1/\varepsilon)$ time.*

2.3.2 The Algorithm

Algorithm W_ε:

INPUT: A set of rectangles R, accuracy $\varepsilon > 0$.

OUTPUT: A packing of a subset of R within $[0, 1+3\varepsilon] \times [0, 1+3\varepsilon]$.

1. **For each** $k \in \{0, 1 \ldots, 2/\varepsilon^2 - 1\}$ form the group $L^{(k)}$ of rectangles $R_i \in R$ as described above and perform Steps 2 and 3.

2. Let $\alpha = 1/(\varepsilon^3 \Delta)$.

   (a) Partition $R \setminus L^{(k)}$ into sets $\mathcal{L}, \mathcal{S}, \mathcal{H}$, and $\mathcal{V}$ as described above.

   (b) Round the sizes of the rectangles $\mathcal{L} \cup \mathcal{H} \cup \mathcal{V}$ as indicated in Lemma 2.3.2.

(c) Compute the set $FEASIBLE$ containing all subsets of $\mathcal{L}$ with at most $1/\Delta^2$ rectangles.

(d) Compute the set $FEASIBLE_{\mathcal{H}}$ containing all *feasible* subsets of $\mathcal{H}$ with *profiles* $(h_1, h_2, \ldots, h_\alpha)$ where each entry $h_q \leq 1$ ($q = 1, \ldots, \alpha$) is a multiple of ε/α.

(e) Compute the set $FEASIBLE_{\mathcal{V}}$ containing all *feasible* subsets of $\mathcal{V}$ with *profiles* $(v_1, v_2, \ldots, v_\alpha)$ where each entry $v_q \leq 1$ ($q = 1, \ldots, \alpha$) is a multiple of ε/α.

3. For each set $\mathcal{L}' \in FEASIBLE$, $\mathcal{H}' \in FEASIBLE_{\mathcal{H}}$, and $\mathcal{V}' \in FEASIBLE_{\mathcal{V}}$ do:

   (a) Try all possible packings for $\mathcal{L}'$ in the frame $[0, 1+\varepsilon] \times [0, 1+\varepsilon]$, positioning the rectangles as indicated in Lemma 2.3.2.

   (b) For each packing of $\mathcal{L}'$ in the frame of size $1+2\varepsilon$, split the empty space with a grid of size ε/α. Try all possible labellings for the grid's squares in which a square is labelled either $\ell_{\mathcal{H}}$ or $\ell_{\mathcal{V}}$. For each labelling, try to pack the rectangles from $\mathcal{H}'$ into the grid squares labelled $\ell_{\mathcal{H}}$, and try to pack $\mathcal{V}'$ into the squares labelled $\ell_{\mathcal{V}}$ by solving a fractional strip-packing problem as described above.

   (c) If there is a packing for $\mathcal{L}' \cup \mathcal{H}' \cup \mathcal{V}'$ in the frame of size $1+2\varepsilon$, find a subset $\mathcal{S}' \subseteq \mathcal{S}$ which is *feasible* for $\mathcal{L}', \mathcal{H}'$ and $\mathcal{V}'$.

   (d) Increase the size of the frame to $[1+3\varepsilon] \times [1+3\varepsilon]$ and use the NFDH algorithm to pack the rectangles $\mathcal{S}'$ within the empty gaps left by $\mathcal{L}' \cup \mathcal{H}' \cup \mathcal{V}'$.

4. Among all packings computed in Step 3, output one having the maximum weight.

2.3.3 Proof of Theorem 2.1.1

Lemma 2.3.5. *There exists a selection of feasible subsets $\mathcal{L}' \in FEASIBLE, \mathcal{H}' \in FEASIBLE_{\mathcal{H}}$, $\mathcal{V}' \in FEASIBLE_{\mathcal{V}}$, and $\mathcal{S}' \subseteq \mathcal{S}$, such that*

- *weight*$(\mathcal{L}' \cup \mathcal{H}' \cup \mathcal{V}' \cup \mathcal{S}') \geq (1-\varepsilon)\mathrm{OPT}$,
- *algorithm W_ε outputs a packing of $\mathcal{L}' \cup \mathcal{H}' \cup \mathcal{V}' \cup \mathcal{S}'$ within the augmented square frame* $[0, 1+3\varepsilon] \times [0, 1+3\varepsilon]$.

Proof. Choose $\mathcal{L}' = \mathcal{L} \cap R^{opt}$. Let $\mathcal{H}' \subseteq \mathcal{H}$ and $\mathcal{V}' \subseteq \mathcal{V}$ be sets with the same rounded profiles as $\mathcal{H} \cap R^{opt}$ and $\mathcal{V} \cap R^{opt}$ and weights at least $(1-\varepsilon)weight(\mathcal{H} \cap R^{opt})$ and $(1-\varepsilon)weight(\mathcal{V} \cap R^{opt})$ respectively. Let $\mathcal{S}' \subseteq \mathcal{S}$ be a $(1-\varepsilon)$-approximate solution of the knapsack problem with knapsack capacity $(1+2\varepsilon)^2 - area(\mathcal{L}' \cup \mathcal{H}' \cup \mathcal{V}')$ and items $R_i \in \mathcal{S}$ of size $a_i b_i$ and weight w_i. Note that $weight(\mathcal{S}') \geq (1-\varepsilon)weight(\mathcal{S} \cap R^{opt})$ and, therefore, $weight(\mathcal{L}' \cup \mathcal{H}' \cup \mathcal{V}' \cup \mathcal{S}') \geq (1-\varepsilon)weight(R^{opt})$. Since R^{opt} can be packed into a unit size square frame and the sets $\mathcal{L}', \mathcal{H}'$, and $\mathcal{V}'$ are rounded-up sets with weights at least the weights of $R^{opt} \cap \mathcal{L}$, $R^{opt} \cap \mathcal{H}$, and $R^{opt} \cap \mathcal{V}$, then, by Lemma 2.3.2 and the discussion in Section 2.3.1 about the selection of $FEASIBLE_{\mathcal{H}}$ and $FEASIBLE_{\mathcal{V}}$, they can be packed into a square frame of size $[0, 1+2\varepsilon] \times [0, 1+2\varepsilon]$. The small rectangles in $\mathcal{S}'$ have total area $(1+2\varepsilon)^2 - area(\mathcal{L}' \cup \mathcal{H}' \cup \mathcal{V}')$ and, thus, the NFDH algorithm can pack them in the empty gaps left by the other rectangles if we increase the size of the frame to $[0, 1+3\varepsilon] \times [0, 1+3\varepsilon]$. This follows from a straightforward extension of Lemma 2.2.1 to rectangles. □

Algorithm W_ε considers all values $k \in \{0, 1 \ldots, 2/\varepsilon^2 - 1\}$. For at least one of these values it must find a group $L^{(k)}$ such that

$$weight(R^{opt} \setminus L^{(k)}) \geq (1-\varepsilon^2)\mathrm{OPT}.$$

For this group, the rest of the rectangles $R \setminus L^{(k)}$ is partitioned into sets $\mathcal{L}, \mathcal{S}, \mathcal{H}$, and $\mathcal{V}$.

By Lemma 2.3.5 there exist a selection of feasible subsets $\mathcal{L}' \in FEASIBLE, \mathcal{H}' \in FEASIBLE_{\mathcal{H}}$, $\mathcal{V}' \in FEASIBLE_{\mathcal{V}}$, and $\mathcal{S}' \subseteq \mathcal{S}$, such that

$$weight(\mathcal{L}' \cup \mathcal{H}' \cup \mathcal{V}' \cup \mathcal{S}') \geq (1-\varepsilon)\mathrm{OPT},$$

and such that algorithm W_ε outputs a packing of $\mathcal{L}' \cup \mathcal{H}' \cup \mathcal{V}' \cup \mathcal{S}'$ within an augmented square frame $[0, 1+3\varepsilon] \times [0, 1+3\varepsilon]$. Since algorithm W_ε tries all feasible sets in $FEASIBLE$, $FEASIBLE_{\mathcal{H}}$, and $FEASIBLE_{\mathcal{V}}$, and all packings for them, W_ε must find the required solution.

All feasible subsets $FEASIBLE$, $FEASIBLE_{\mathcal{H}}$ and $FEASIBLE_{\mathcal{V}}$, can be found in $O(n^2 \cdot 1/\varepsilon)$ time. Step 3(b) of algorithm W_ε can be performed by using the algorithm for strip-packing described in [18]. This algorithm also runs in time polynomial in n. Furthermore, there is only a constant number of possible packings for any set of large rectangles from $FEASIBLE$. Hence, the overall running time of algorithm W_ε is polynomial in n for fixed ε.

2.4 Concluding Remarks

Following the same line of ideas, our result for packing squares can be extended to the packing of squares into a square $[0,1] \times [0, 1+\varepsilon]$, which is augmented only in one direction, as well as to the packing of squares into a square $[0,1] \times [0,1]$ without augmentation. An interesting open problem, however, is that of finding a set $R' \subseteq R$ of rectangles with weight at least $(1-\varepsilon)\mathrm{OPT}$ and a packing for them in the unit square region $[0,1] \times [0,1]$ without augmentation. Natural extensions of our algorithm (like removing one of the large rectangles to accommodate those rectangles that in our algorithm would overflow the boundaries of the unit square region, thus, requiring the ε extension in the size of the region) do not work. We conjecture that this more complex problem can be solved in polynomial time, but new techniques seem to be needed.

CHAPTER 3

ON WEIGHTED RECTANGLE PACKING WITH LARGE RESOURCES

3.1 INTRODUCTION

In this chapter we address the following general version of the storage packing problem: We are given a dedicated rectangle R of width $a \geq 0$ and height $b \geq 0$, and a list L of n rectangles R_i $(i = 1, \ldots, n)$ with widths $a_i \in (0, a]$, heights $b_i \in (0, b]$, and positive integral weights w_i. For a sublist $L' \subseteq L$ of rectangles, a *packing* of L' into the dedicated rectangle R is a positioning of the rectangles from L' within the area $[0, a] \times [0, b]$, so that all the rectangles of L' have disjoint interiors. Rectangles are not allowed to rotate. The goal is to find a sublist of rectangles $L' \subseteq L$ and its packing in R which maximizes the weight of packed rectangles, i.e., $\sum_{R_i \in L'} w_i$.

The above problem is a natural generalization of the knapsack problem to the two-dimensional version. The knapsack problem is known to be NP-hard [36]. Hence it is very unlikely that any polynomial time algorithm exists. So, then one looks for efficient heuristics with good performance guarantees.

Related results. As we mentioned, one can find a clear relation to the knapsack problem. It is well-known that the knapsack problem is just weakly NP-hard [36], and admits an FPTAS [55, 60]. In contrast, already the problem of packing squares with unit weights into a rectangle is strongly NP-hard [8]. So, the problem of packing rectangles with weights into a rectangle admits no FPTAS, unless $\mathrm{P} = \mathrm{NP}$.

From another side, one can also find a relation to strip packing: Given a list L of rectangles R_i $(i = 1, \ldots, n)$ with widths $a_i \in (0,1]$ and positive heights $b_i \geq 0$ it is required to pack the rectangles of L into the vertical strip $[0,1] \times [0,+\infty)$ so that the packing height is minimized. In particular, this also defines the problem of packing rectangles into a rectangle of fixed width and minimum height, or the well-known two-dimensional cutting stock problem [37].

Of course, the strip packing problem is strongly NP-hard since it includes the bin packing problem as a special case. In fact many known simple strip packing ideas come from bin packing. The "Bottom-Left" heuristic has asymptotic performance ratio equal to 2 when the rectangles are sorted by decreasing widths [9]. In [15] several simple algorithms were studied where the rectangles are placed on "shelves" using one-dimensional bin-packing heuristics. It was shown that the First-Fit shelf algorithm has asymptotic performance ratio of 1.7 when the rectangles are sorted by decreasing height (this defines the First-Fit-Decreasing-Height algorithm). The asymptotic performance ratio was further reduced to $3/2$ [83], then to $4/3$ [38] and to $5/4$ [7]. Finally, in [56] it was shown that there exists an asymptotic FPTAS in the case when the side lengths of all rectangles in the list are at most 1. (In the above definition $a_i, b_i \in (0,1]$ for all R_i.) For the absolute performance, the two best current algorithms have the same performance ratio 2 [79, 84].

In contrast to knapsack and strip packing there are just few results known for packing rectangles into a rectangle. For a long time the only known result has been an asymptotic $(4/3)$-approximation algorithm for packing unweighted squares into a rectangle [8]. Only very recently in [51], several first approximability results have been presented for the packing rectangles with weights into a rectangle. The best one is a $(\frac{1}{2} - \varepsilon)$-approximation algorithm.

Our results. Inspired by the results in the previous chapter we investigate the influence of resources. In this chapter we consider the so-called case of large resources, when the number of the packed rectangles is relatively large. Formally, in the above formulation it is assumed that all rectangles R_i $(i = 1, \ldots, n)$ in the list L have widths and heights $a_i, b_i \in (0,1]$, and the dedicated rectangle R has unit width $a = 1$ and quite a large height $b \geq 1/\varepsilon^4$, for a fixed positive $\varepsilon > 0$. We present an algorithm which finds a sublist $L' \subseteq L$ of rectangles and its packing into the dedicated rectangle R with weight at least $(1-\varepsilon)\mathrm{OPT}$, where OPT is the optimum weight. The running time of the algorithm is polynomial in the number of rectangles n.

Our approach to approximation is as follows. At the beginning we take an optimal rectangle packing inside of the dedicated rectangle, considering it as a strip packing. We then perform several transformations that simplify the packing structure, without dramatically increasing the packing height and decreasing the packing weight, such that the final result is amenable to a fast enumeration. As soon as such a ”near-optimal” strip packing is found, we apply our shifting technique. This puts the packing into the dedicated rectangle by removing some less weighted piece of the packing.

Applications. There has recently been increasing interest in the advertisement placement problem for newspapers and the Internet [2, 33]. In a basic version of the problem, we are given a list of n advertisements and k identical rectangular pages of fixed size (a,b), on which advertisements may be placed. Each ith advertisement appears as a small rectangle of size (a_i,b_i), and is associated with a profit p_i ($i = 1,\ldots,n$). Advertisements may not overlap. The goal is to maximize the total profit of the advertisements placed on all k pages.

This problem is also known as the problem of packing n weighted rectangles into k identical rectangular bins. Here, as an application of our algorithm, we provide a $(\frac{1}{2}-\varepsilon)$-approximation algorithm. The running time of the algorithm is polynomial in the number of rectangles n for any fixed $\varepsilon > 0$.

Last notes. The chapter is organized as follows. In section 3.2 we introduce notations and give some preliminary results. In Section 3.3, we present our shifting technique. In Section 3.4 we perform packing transformations. In Section 3.5 we outline the algorithm. In Section 3.6 we give an approximation algorithm to pack rectangles into k rectangular bins of size (a,b). Finally, in the last section we give some concluding remarks.

3.2 Preliminaries

We are given a dedicated rectangle R of unit width $a = 1$ and height $b \geq 0$, and a list L of rectangles R_i $(i = 1,\dots,n)$ with widths $a_i \in (0,1]$, heights $b_i \in (0,1]$, and positive integral weights w_i. The goal is to find a sublist of rectangles $L' \subseteq L$ and its packing in R which maximizes the weight of the packed rectangles, i.e., $\sum_{R_i \in L'} w_i$.

We will use the following notations. For a sublist of rectangles $L' \subseteq L$, we will write $weight(L')$, $height(L')$, and $size(L')$ to denote the values of $\sum_{R_i \in L'} w_i$; $\sum_{R_i \in L'} b_i$, and $\sum_{R_i \in L'} a_i \cdot b_i$, respectively. Also, we will write $L^{opt} \subseteq L$ to denote an optimal sublist of rectangles, and OPT to denote the optimal objective value. Thus, $weight(L^{opt}) = \text{OPT}$ and $size(L^{opt}) \leq a \cdot b = b$. Throughout of the chapter we assume that $0 < \varepsilon < 1/50$, $1/\varepsilon' = (2+\varepsilon)/\varepsilon$ is integral ($\varepsilon' = \varepsilon/(2+\varepsilon)$), $m = 1/(\varepsilon')^2$, and the height value $b \geq 1/\varepsilon^4$.

3.2.1 *Separating rectangles*

Given a positive $\varepsilon' > 0$, we partition the list L of rectangles into two sublists: L_{narrow}, containing all the rectangles of width at most ε', and L_{wide}, containing all the rectangles of width larger than ε'.

3.2.2 *Knapsack*

In the knapsack problem we are given a knapsack capacity B and a set of items $I = \{1,2,\dots,n\}$, where each item $i \in I$ is associated with its size s_i and profit p_i. It is required to find a subset $I' \subseteq I$ which maximizes the profit of $\sum_{i \in I'} p_i$ subject to $\sum_{i\in I'} s_i \le B$, i.e., it fits in a knapsack of size B.

The knapsack problem is NP-hard, but it admits an FPTAS [36]. In particular, we can use any FPTAS version from [55, 60]. Given a precision $\delta > 0$, the algorithm outputs a subset $I(B) \subseteq I$ such that

$$\sum_{i\in I(B)} s_i \le B \text{ and } \sum_{i\in I(B)} p_i \ge (1-\delta)\mathrm{OPT}(I,B), \tag{3.1}$$

where $\mathrm{OPT}(I,B)$ is the maximum profit of I with respect to capacity B. For simplicity, we will write $KS(n,\delta)$ to denote the running time of the algorithm, which is polynomial in the number of items n and $1/\delta$.

3.2.3 *Solving knapsacks with wide and narrow rectangles*

Here we work with rectangles as items. However, we treat narrow and wide rectangles differently.

Knapsacks with wide rectangles. We handle wide rectangles as follows. We order all the wide rectangles in L_{wide} by non-increasing widths. W.l.o.g. we assume that there are n' wide rectangles

$$R_1 = (a_1,b_1), R_2 = (a_2,b_2), \ldots, R_{n'} = (a_{n'},b_{n'})$$

with widths

$$a_1 \ge a_2 \ge \ldots \ge a_{n'} \ge \varepsilon'.$$

So, for any two $1 \le k < \ell \le n'$, let $L_{wide}(k,\ell)$ denote the list of all wide rectangles R_i in L_{wide} with $\ell \ge i \ge k$. Next, we only pay attention to the height values.

Let H be some positive variable. Let $L_{wide}(k,\ell)$ be the list of wide rectangles between R_k and R_ℓ as defined above. We associate each wide rectangle $R_i = (a_i,b_i)$ of weight w_i in $L_{wide}(k,\ell)$ with item i in $I \subseteq \{1,2,\ldots,n\}$ of size $s_i := b_i$ and profit $p_i := w_i$. We also define knapsack capacity $B := H$. So, given precision $\delta := \varepsilon^2/4$, knapsack capacity B and item set I we apply the FPTAS. The solution defines some sublist $L_{wide}(k,\ell,H) \subseteq L_{wide}(k,\ell)$ of wide rectangles with precision $\varepsilon^2/4$.

Lemma 3.2.1. *The height of* $L_{wide}(k,\ell,H)$ *is at most* H*. Furthermore,*

$$weight(L_{wide}(k,\ell,H)) \geq (1-\varepsilon^2/4)\mathrm{OPT}(L_{wide}(k,\ell),H),$$

where $\mathrm{OPT}(L_{wide}(k,\ell),H)$ *is the maximum profit of a subset of* $L_{wide}(k,\ell)$ *with respect to capacity (height)* H*.*

Knapsacks with narrow rectangles. Similarly, we deal with narrow rectangles. However, we only pay attention to the size values.

Let S be some positive variable. Let L_{narrow} be the list of all narrow rectangles. We associate each narrow rectangle $R_i = (a_i, b_i)$ of weight w_i in $L_{narrow}(k,\ell)$ with item i in $I \subseteq \{1,2,\ldots,n\}$ of size $s_i = a_i \cdot b_i$ and profit $p_i = w_i$. We also define knapsack capacity $B := S$. So, given precision $\delta := \varepsilon^2/4$, knapsack capacity B and items I we apply the FPTAS. The solution defines some sublist $L_{narrow}(S) \subseteq L_{narrow}$ of narrow rectangles with precision $\varepsilon^2/4$.

Lemma 3.2.2. *The size of* $L_{narrow}(S)$ *is at most* S*. Furthermore,*

$$weight(L_{narrow}(S)) \geq (1-\varepsilon^2/4)\mathrm{OPT}(L_{narrow},S),$$

where $\mathrm{OPT}(L_{narrow},S)$ *is the maximum profit of a subset of* L_{narrow} *with respect to capacity (area)* S*.*

3.2.4 Packing narrow rectangles: NFDH

We consider the following strip-packing problem: Given a sublist $L' \subseteq L_{narrow}$ of narrow rectangles and a strip with fixed width $1-c$ ($c \in [0,1]$) and unbounded height, pack the rectangles of L' into the the strip such that the height to which the strip is filled is as small as possible.

First, we order the rectangles of L' by decreasing heights. Then, we put the narrow rectangles into the strip-packing by using Next-Fit-Decreasing-Height (NFDH): The rectangles are packed so as to form a sequence of sublevels. The first sublevel is just the bottom line of the strip. Each subsequent sublevel is defined by a horizontal line drawn through the top of the rectangle placed on the previous sublevel. Rectangles are packed in a left-justified greedy manner, until there is insufficient space to the right to place the next rectangle, at that point, the current sublevel is discontinued, the next sublevel is defined and packing proceeds on the new sublevel. For an illustration see Fig. 3.1.

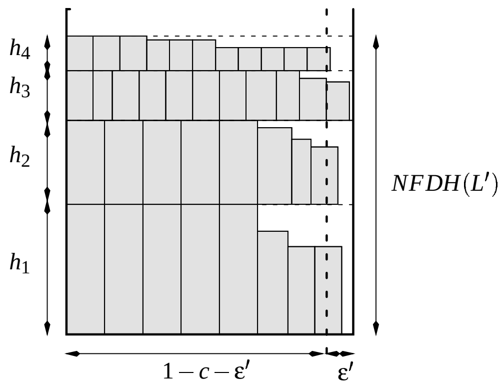


Figure 3.1: NFDH for narrow rectangles

We will use the following simple result.

Lemma 3.2.3. *Let $L' \subseteq L_{narrow}$ be any sublist of narrow rectangles ordered by non-increasing heights. If the Next-Fit-Decreasing-Height (NFDH) heuristic outputs a packing of height $NFDH(L')$, then the area covered by the narrow rectangles*

$$AREA \geq (1-c-\varepsilon')(NFDH(L')-1). \tag{3.2}$$

Proof. Let q be the number of sublevels. Let h_i be the height of the first rectangle on the ith sublevel. Recall that NFDH packs the rectangles of L' on sublevels in order of non-increasing heights. Hence,

$$NFDH(L') = \sum_{i=1}^{q} h_i,$$

and

$$1 \geq h_1 \geq h_2 \geq \ldots \geq h_q > 0.$$

(All rectangle heights are in $(0,1]$.) Since no rectangle in L' has width exceeding $\varepsilon' \geq 0$, the total width on each sublevel is at least $(1-c)-\varepsilon' = 1-c-\varepsilon'$. Recall that the rectangles of L' are packed in order of non-increasing heights. Thus, the size of rectangles on each ith ($i = 1,\ldots,q-1$) sublevel is at least

$$h_{i+1}((1-c)-\varepsilon').$$

So, the covered area is

$$\begin{aligned} AREA &\geq \sum_{i=1}^{q-1} h_{i+1}((1-c)-\varepsilon') \\ &= (1-c-\varepsilon')\sum_{i=2}^{q} h_i \\ &= (1-c-\varepsilon')(NFDH(L')-h_1) \\ &\geq (1-c-\varepsilon')(NFDH(L')-1). \end{aligned}$$

The result of lemma follows. □

3.2.5 Strip packing by KR-algorithm

We consider the following strip-packing problem: Given a sublist $L' \subseteq L$ of rectangles and a strip with unit width and unbounded height, pack the rectangles of L' into the the strip such that the height to which the strip is filled is as small as possible.

As we mentioned before the strip packing problem admits an asymptotic FPTAS. We will use the following result.(See also Appendix 5.4)

Theorem 3.2.4 (C. Kenyon, E. Rémila [56]). *There is an algorithm A which, given an accuracy $\varepsilon > 0$, a sublist $L' \subseteq L$ of rectangles and a strip with unit width* 1 *and unbounded height, packs the rectangles of L' into the the strip such that the height to which the strip is filled*

$$A(L') \leq (1+\varepsilon)strip(L') + O(1/\varepsilon^2), \tag{3.3}$$

where $strip(L')$ denotes the height of the optimal strip packing of L'. The running time of A is polynomial in n and $1/\varepsilon$.

For simplicity, we name such an algorithm in the theorem by the KR-algorithm (description of the KR-algorithm is in Appendix B on page 139). Also, we will write $KR(n,\varepsilon)$ to denote its running time. In Section 3.4 we will give more details on packing by the KR-algorithm.

3.3 SHIFTING

Assume that we are given a strip packing of height $(1+O(\varepsilon))b$ for a list of rectangles whose weight is at least $(1-O(\varepsilon))\mathrm{OPT}$. The idea of our shifting technique is to remove some less weighted piece of height $O(\varepsilon)b$. Then, the weight value remains $(1-O(\varepsilon))\mathrm{OPT}$, but the height value reduces to b, giving a packing in the area $[0,1]\times[0,b]$ of the dedicated rectangle $R=(1,b)$.

Lemma 3.3.1. *Suppose we are given a strip packing of height $(1+\delta_2\cdot\varepsilon))b$ for a sublist $L'\subseteq L$ with weight at least $(1-\delta_1\cdot\varepsilon)OPT$, for some $\delta_1,\delta_2=O(1)$. Then in $O(n+1/\varepsilon)$ time one can obtain a rectangle packing of a sublist of L' into the area $[0,1]\times[0,b]$ whose weight is at least $(1-(\delta_1+2\delta_2+2)\varepsilon)\mathrm{OPT}$, provided $1/\varepsilon\geq\delta_2+1$.*

Proof. Recall that $\delta_2=O(1)$ and $b\geq 1/\varepsilon^4$. W.l.o.g. it can be assumed that $weight(L')\leq 2\mathrm{OPT}$, i.e. the weight of L' is not larger than 2OPT. If it is larger than 2OPT, we could proceed as follows. Take the current strip packing of L' of height $(1+\delta_2\cdot\varepsilon)b$. Cut it by a horizontal line at height point b. This gives the two strip packing of height b and at most $(\delta_2\cdot\varepsilon)b+1$, respectively. So, either of the strip packings is a feasible rectangle packing in the area of the dedicated rectangle $R=(1,b)$. Furthermore, one of them must have the weight value larger than OPT. This gives a contradiction.

Now we define

$$k=\left\lfloor\frac{(1+\delta_2\cdot\varepsilon)b+2}{(\delta_2\cdot\varepsilon)b+2}\right\rfloor.$$

Since $b \geq 1/\varepsilon^4$ and $\varepsilon \in (0, 1/4]$ we also have that

$$\begin{aligned} k = \left\lfloor \frac{b}{(\delta_2 \cdot \varepsilon)b + 2} + 1 \right\rfloor &\geq \left\lfloor \frac{1}{(\delta_2 \cdot \varepsilon) + (2/b)} + 1 \right\rfloor \\ \geq \left\lfloor \frac{1}{(\delta_2 \cdot \varepsilon) + 2\varepsilon^3} + 1 \right\rfloor &\geq \left\lfloor \frac{1}{\varepsilon(\delta_2 + 1)} + 1 \right\rfloor. \end{aligned}$$

Assume now that

$$1/\varepsilon \ \geq \ \delta_2 + 1. \tag{3.4}$$

Then, $k \geq 2$. Next, we proceed as follows. We take the current strip packing of length $(1 + \delta_2 \cdot \varepsilon)b$. We draw $k - 1$ horizontal lines which divide the packing into k cuts, as shown in Fig. 3.2. Each of the cuts has the inner part of height $(\delta_2 \cdot \varepsilon)b$ and the outer part of height 2. Then, the height of the k cuts is

$$((\delta_2 \cdot \varepsilon)b + 2)k - 2 \leq (1 + \delta_2 \cdot \varepsilon)b.$$

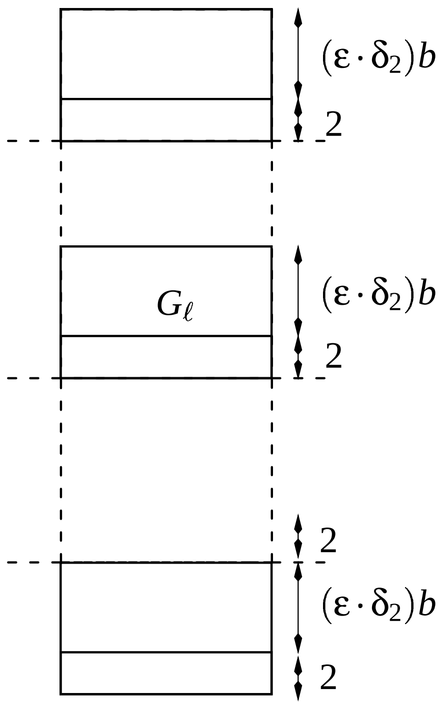


Figure 3.2: Shifting

Let G_i be the list of rectangles which intersect the inner part of the ith cut. Each outer part has height 2, but no rectangle in the list L can be higher than 1. Hence, we have that $G_i \cap G_j = \emptyset$ for $i \neq j$. Furthermore,

$$\sum_{i=1}^{k} weight(G_i) \leq weight(L') \leq 2\mathrm{OPT}.$$

Since

$$k \geq \left\lfloor \frac{1}{\varepsilon(\delta_2+1)} + 1 \right\rfloor \geq \frac{1}{\varepsilon(\delta_2+1)},$$

there must exist at least one list G_ℓ such that

$$weight(G_\ell) \leq [2\mathrm{OPT}](1/k) \leq 2\varepsilon(\delta_2+1)\mathrm{OPT}.$$

So, we break the strip packing into two ones from both sides of the inner part of the ℓth cut. Next, we throw away the rectangles of G_ℓ, and put these two strip packing together. This gives a strip packing of height b. Its weight is bounded below by

$$\begin{aligned}(1-\delta_1\cdot\varepsilon)\mathrm{OPT} - weight(G_\ell) &\geq (1-\delta_1\cdot\varepsilon)\mathrm{OPT} - 2\varepsilon(\delta_2+1)\mathrm{OPT} \\ &= (1-(\delta_1+2\delta_2+2)\varepsilon)\mathrm{OPT}.\end{aligned}$$

The construction requires at most $O(n+k)$ time. From $\delta_1, \delta_2 = O(1)$, this turns to $O(n+1/\varepsilon)$, and the result of lemma follows. □

Corollary 3.3.2. *Let* $\beta \geq 4$, $b \geq \alpha/\varepsilon^4$ *and* $\varepsilon \in (0, 1/\beta]$. *Then, given a packing of* L' *in the area* $[0,1] \times [0,(1+\delta_2\cdot\varepsilon)b]$ *whose weight is at least* $(1-\delta_1\cdot\varepsilon)\mathrm{OPT}$, *in time* $O(n+1/\varepsilon)$ *one can obtain a packing in the area* $[0,1]\times[0,b]$ *whose weight is at least* $(1-35\varepsilon)\mathrm{OPT}$ *if* $\beta = 50$, $\delta_1 = 1/3$, $\delta_2 = 16$.

Proof. Let $b \geq 1/\varepsilon^4$, $\varepsilon \in (0,1/\beta]$, $\delta_1 = 1/3$, and $\delta_2 = 16$. Then, for $\beta = 50$ we have that

$$\frac{1}{\varepsilon} \;\geq\; \beta = 50 \;\geq\; 1+\delta_2 = 17.$$

Hence, by Lemma 3.3.1, the shifting procedure outputs a packing whose weight is at least

$$(1-(\delta_1+2\delta_2+2)\varepsilon)\mathrm{OPT} \geq (1-35\varepsilon)\mathrm{OPT}.$$

□

3.4 Transformations of optimal solution

Here we discuss some transformations which simplify the structure of the optimal solution L^{opt}. We start with transforming a packing of L^{opt} into a well structured packing. This introduces the lists L^{opt}_{wide} of wide rectangles, L^{opt}_{narrow} of narrow rectangles, and m optimal threshold rectangles. Next, assuming the m threshold rectangles and the m height capacity values are known, we perform a transformation of the optimal lists L^{opt}_{wide} and L^{opt}_{narrow} to some lists found by solving a series of knapsacks. Then, we perform a rounding transformation which turns all the m height capacity values to some discrete points. Each of these transformations may increases the height value by $O(\varepsilon b)$, and may decrease the weight value by $O(\varepsilon \mathrm{OPT})$. However, in the next section we show that L^{opt} can be still approximated with quite a good precision.

3.4.1 *Well-structured packing*

Here we describe a well structured packing of the optimal solution.

Separation. Let L^{opt} be the optimal solution. We define the lists of narrow and wide rectangles: $L^{opt}_{narrow} = L^{opt} \cap L_{narrow}$ and $L^{opt}_{wide} = L^{opt} \cap L_{wide}$. Clearly,

$$weight(L^{opt}_{wide}) + weight(L^{opt}_{narrow}) = \mathrm{OPT}. \tag{3.5}$$

Threshold rectangles. Let $R_{k_1} = (a_{k_1}, b_{k_1}), R_{k_2} = (a_{k_2}, b_{k_2}), \ldots, R_{k_m} = (a_{k_m}, b_{k_m})$ be a sequence of optimal wide rectangles in L^{opt}_{wide} such that $1 \le k_1 < k_2 < \ldots < k_m \le n'$. Then, we call such rectangles as the threshold rectangles. For an illustration see Fig. 3.3. As it is defined, widths

$$a_{k_1} \ge a_{k_2} \ge \ldots \ge a_{k_m} \ge \varepsilon'.$$

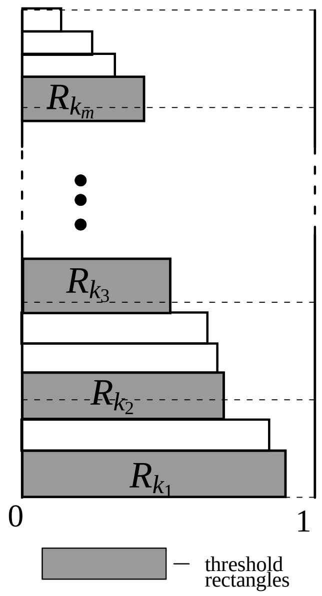


Figure 3.3: Threshold rectangles

Configurations. Now we can define configurations. A configuration is defined as a multi-set of widths chosen among the m threshold widths in $\{a_{k_i}|i=1,\ldots,m\}$ which sum to at most 1, i.e. they may occur at the same level. Their sum is called the width of the configuration.

Layers. Let q be some positive integer. Let $C_1, C_2, \ldots, C_q$ be some distinct configurations, numbered by non-increasing widths, and let C_{q+1} be an empty configuration. Let α_{ij} denote the number of occurrences of width a_{k_i} in C_j. Then, the value of $c_j = \sum_{i=1}^{m} a_{k(ij)}\alpha_{ij}$ is called the width of C_j. Therefore,

$$c_1 \geq c_2 \geq \ldots \geq c_q \geq c_{q+1} = 0.$$

Let $0 = \ell_0 \leq \ell_1 \leq \ldots \leq \ell_q \leq \ell_{q+1} = h$ be some $q+1$ non-negative values. We define $q+1$ layers as follows. The layer $[0,1]\times[\ell_j,\ell_{j+1}]$ $(j=0,\ldots,q+1)$ corresponds to configuration C_j. It is divided into two rectangles: $Q_j = [c_j,1]\times[\ell_j,\ell_{j+1}]$ and $Q'_j = [0,c_j]\times[\ell_j,\ell_{j+1}]$. (Notice that the last layer is $Q_{q+1} = [0,1]\times[\ell_q,\ell_{q+1}]$, as shown in Fig. 3.4)

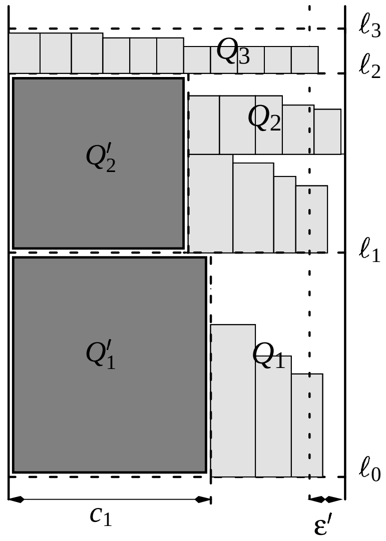


Figure 3.4: A well structured packing with 3 layers

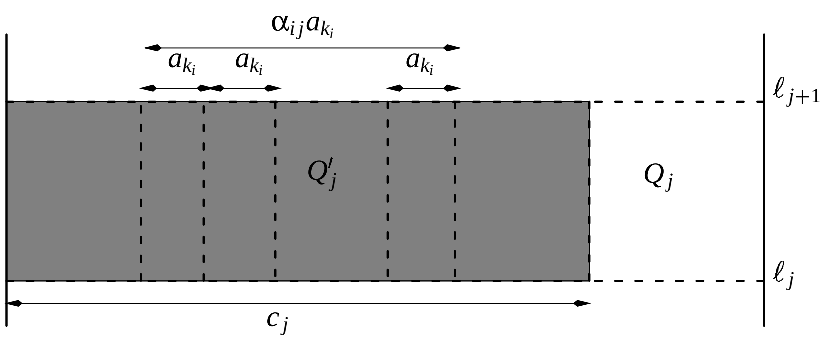


Figure 3.5: Structure of layer $[0,1] \times [\ell_j, \ell_{j+1}]$

From one side, all Q_j ($j = 1,\dots,q+1$) are empty. From another side, each Q'_j ($j = 1,\dots,q$) consists of m vertical multi-slices, each ith of those with exactly α_{ij} identical slices of width a_{k_i}, as shown in Fig. 3.5. The value of $(\ell_{j+1} - \ell_j)$ defines the height of configuration C_j, and the value of $h = \ell_{q+1}$ defines the packing height. The value of

$$H_i = \sum_{j=1}^{q} \alpha_{ij}(\ell_{j+1} - \ell_j)$$

defines the total height of all slices of width a_{k_i}, and it is called the ith threshold capacity.

Well-structured packing. A strip packing of the optimal solution L^{opt} is called a well-structured strip packing with $q+1$ layers if all Q_j ($j = 1,\dots,q+1$) are filled by narrow rectangles, and all the slices of width a_{k_i} ($i = 1,\dots,m$) are greedily filled by the wide rectangles from $L^{opt} \cap L_{wide}(k_i, k_{i+1} - 1)$. (Here and further we assume w.l.o.g. that $k_{m+1} - 1 = n'$.) Now we are ready to give the following result.

Theorem 3.4.1 (C. Kenyon, E. Rémila [56]). *There exist a well-structured packing of L^{opt} with $2m+1$ layers such that its height*

$$\begin{aligned} h \le \max\{&strip(L^{opt}_{wide})(1 + 1/(m\varepsilon')) + 2m + 1, \\ &size(L^{opt})(1 + 1/(m\varepsilon'))/(1 - \varepsilon') + 4m + 1\}, \end{aligned}$$

where $strip(L^{opt}_{wide})$ is the height of the optimal strip packing of L^{opt}_{wide}.

3.4.2 Augmentation

Now we can give the following simple result.

Lemma 3.4.2. *If $\varepsilon' = \varepsilon/(2+\varepsilon)$, $m = (1/\varepsilon')^2$, $\varepsilon < 1/2^{10}$ and $b \ge 1/\varepsilon^4$, then there exists a well-structured packing with $2m+1$ layers of the optimal solution L^{opt} of height*

$$h \le (1 + 2\varepsilon)b. \tag{3.6}$$

Proof. Recall that $strip(L^{opt}_{wide})$ is the height of the optimal strip packing of the wide rectangles of L^{opt}_{wide}, and $size(L^{opt})$ is the area of the optimal strip packing of L^{opt}. As we know $L^{opt}_{wide} \subseteq L^{opt}$. Since L^{opt} is an optimal solution, the rectangles of L^{opt} can be packed into the dedicated rectangle $R = (a,b)$. Hence $strip(L^{opt}_{wide}) \leq strip(L^{opt}) \leq b$. Since $a = 1$, the value of $size(L^{opt})$ must be at most $1 \cdot b$. Recall also that $m = 1/(\varepsilon')^2$. Substituting, we have that

$$\begin{aligned}
h &\leq b(1+\varepsilon')/(1-\varepsilon') + 4/(\varepsilon')^2 + 1 \quad \text{from } \varepsilon < 1 \text{ and } \varepsilon' = \varepsilon/(2+\varepsilon)\\
&\leq b(2+2\varepsilon)/2 + 4(2+\varepsilon)^2/(\varepsilon^2) + 1\\
&\leq b(1+\varepsilon) + 4\cdot 3^2/(\varepsilon^2) + 1/\varepsilon\\
&\leq b(1+\varepsilon) + (36+1)/\varepsilon^2\\
&\leq (1+\varepsilon)b + 37/\varepsilon^2\\
&\leq (1+\varepsilon)b + \varepsilon b = (1+2\varepsilon)b \quad \text{from } b \geq 1/\varepsilon^4 \text{ and } \varepsilon < 1/50.
\end{aligned}$$

The result of lemma follows. □

3.4.3 Approximating wide rectangles

Our idea is to guess most profitable rectangles, knowing the optimal threshold rectangles and capacity values. Let R_{k_i} and H_i ($i = 1,\dots,m$) be the optimal ith threshold rectangle and capacity, respectively. Then, by solving a series of knapsacks we can find the lists $L_{wide}(k_i, k_{i+1}-1, H_i)$ of wide rectangles. These are quite good approximations for lists $L_{wide}(k_i, k_{i+1}-1) \cap L^{opt}$, and hence all together they give a good approximation of the optimal list L^{opt}_{wide} of wide rectangles.

Lemma 3.4.3. *The value of*

$$\sum_{i=1}^{m} weight(L_{wide}(k_i, k_{i+1}-1, H_i)) \geq (1-\varepsilon^2/4)\, weight(L^{opt}_{wide}). \qquad (3.7)$$

If the wide rectangles of L^{opt}_{wide} are replaced by the rectangles of all $L_{wide}(k_i, k_{i+1}-1, H_i)$ ($i = 1,\dots,m$), then the height h of the well-structured packing increases by at most $\Delta_{wide} \leq \varepsilon b$.

Proof. As it was defined,

$$L_{wide}(k_i, k_{i+1}-1) \cap L^{opt} \subseteq L_{wide}(k_i, k_{i+1}-1).$$

In the well structured packing, the rectangles of $L_{wide}(k_i, k_{i+1}-1) \cap L^{opt}$ are placed in the slices of width a_{k_i}. The total height of all these slices is exactly the value of H_i. So,

$$height(L_{wide}(k_i, k_{i+1}-1) \cap L^{opt}) \leq H_i.$$

Hence, by Lemma 3.2.1 solving the knapsack problem we can decrease the weight by at most some factor of $(1-\varepsilon^2/4)$. Combining, the value of

$$\sum_{i=1}^{m} weight(L_{wide}(k_i, k_{i+1}-1, H_i)) \geq \sum_{i=1}^{m} (1-\varepsilon^2/4) weight(L_{wide}(k_i, k_{i+1}-1) \cap L^{opt})$$
$$= (1-\varepsilon^2/4) weight(L^{opt}_{wide}).$$

Notice that both $L_{wide}(k_i, k_{i+1}-1, H_i)$ and $L_{wide}(k_i, k_{i+1}-1) \cap L^{opt}$ have quite similar characteristics. We use it as follows. We take the well-structured packing of L^{opt} and go over all the rectangles $Q'_1, Q'_2, \ldots, Q'_{2m}$ in the $2m$ layers. Inside all the slices of widths a_{k_i} $(i = 1, \ldots, m)$ we replace the rectangles of $L_{wide}(k_i, k_{i+1}-1) \cap L^{opt}$ by the rectangles of $L_{wide}(k_i, k_{i+1}-1, H_i)$ in a greedy manner.

Since we greedily place rectangles, it may happen that some rectangles do not fit completely into the slices. We then increase the height of each layer by 1, that must create enough space for all rectangles. Since there are $2m$ layers, we increase the height h of the well-structured packing by at most

$$\Delta_{wide} = 2m = 2/(\varepsilon')^2 = 2(2+\varepsilon)^2/\varepsilon^2 \leq 2 \cdot 3^2/\varepsilon^2 \leq \varepsilon b,$$

for $\varepsilon < 1/50$, $\varepsilon' = \varepsilon/(2+\varepsilon)$ and $b \geq 1/\varepsilon^4$. The result of lemma follows. $\square$

3.4.4 *Approximating narrow rectangles*

We use a similar idea to guess most profitable narrow rectangles, knowing the optimal configurations with heights and widths. Let c_j and ℓ_j $(i = 1, \ldots, 2m+1)$ be the width and height of configuration C_j, respectively. Recall that the optimal narrow rectangles of L^{opt}_{narrow} are placed in rectangles $Q_1, Q_2, \ldots, Q_{2m}, Q_{2m+1}$. Hence we can bound the size value

$$size(L^{opt}_{narrow}) \leq \sum_{j=1}^{2m+1} (1 - c_j)(\ell_{i+1} - \ell_i). \tag{3.8}$$

So, by solving the knapsack problem we can find the list $L_{narrow}(S)$ of narrow rectangles, where the value of knapsack capacity

$$S = \sum_{j=1}^{2m+1} (1 - c_j)(\ell_{j+1} - \ell_j). \tag{3.9}$$

This is a good approximation of the optimal list L^{opt}_{narrow} of narrow rectangles.

Lemma 3.4.4. *The value of*

$$weight(L_{narrow}(S)) \geq (1 - \varepsilon^2/4) weight(L^{opt}_{narrow}). \tag{3.10}$$

If the narrow rectangles of L^{opt}_{narrow} are replaced by the narrow rectangles $L_{narrow}(S)$, then the height h of the well-structured packing increases by at most $\Delta_{narrow} \leq 2\varepsilon b$.

Proof. Clearly, the rectangles of L^{opt}_{narrow} must be in L_{narrow}. By (3.8), the area of L^{opt}_{narrow} is at most S. Hence, by Lemma 3.2.2 solving the knapsack problem can only decrease the weight by some factor of $(1 - \varepsilon^2/4)$. So, we get

$$weight(L_{narrow}(S)) \geq (1 - \varepsilon^2/4) weight(L^{opt}_{narrow}).$$

Notice that both L^{opt}_{narrow} and $L_{narrow}(S)$ have quite similar characteristics. We use it as follows. We go over the rectangles $Q_1, Q_2, \ldots, Q_{2m}, Q_{2m+1}$ in the $2m + 1$ layers, and place the rectangles of $L_{narrow}(S)$ by using NFDH. If not all rectangles are placed, then we work with a new layer of width 1 and height Δ_{narrow}.

The new rectangle has width 1 and height Δ_{narrow}. Similar to Lemma 3.2.3, the area covered by narrow rectangles in additional layer is at least

$$(1 - \varepsilon')(\Delta_{narrow} - 1).$$

Similarly, consider the narrow rectangles packed in rectangle Q_j ($j = 1,\ldots,2m+1$). The height of this packing is at least $\ell_{j+1} - \ell_j - 1$. The width of Q_j is $1 - c_j$. Hence, the area covered by the narrow rectangles is at least

$$(1 - c_j - \varepsilon')(\ell_{j+1} - \ell_j - 2).$$

Combining over all layers, the area covered is at least

$$\sum_{j=1}^{2m+1} (1 - c_j - \varepsilon')(\ell_{j+1} - \ell_j - 2) + (1 - \varepsilon')(\Delta_{narrow} - 1).$$

Recall that the area of $L^{opt}_{narrow}(S)$ is at most

$$S = \sum_{j=1}^{2m+1} (1 - c_j)(\ell_{j+1} - \ell_j).$$

We need an upper bound on the value of Δ_{narrow}. So, it is enough to require that this size value is equal to the above bound. So,

$$\sum_{j=1}^{2m+1} (1 - c_j - \varepsilon')(\ell_{j+1} - \ell_j - 2) + (1 - \varepsilon')(\Delta_{narrow} - 1) \leq \sum_{j=1}^{2m+1} (1 - c_j)(\ell_{j+1} - \ell_j).$$

Hence,

$$(1 - \varepsilon')(\Delta_{narrow} - 1) \leq 2 \sum_{j=1}^{2m+1} (1 - c_j - \varepsilon') + \varepsilon' \sum_{j=1}^{2m+1} (\ell_{j+1} - \ell_j).$$

and from $\sum_{j=1}^{2m+1} (\ell_{j+1} - \ell_j) = h$

$$\begin{aligned}
\Delta_{narrow} &\leq 1 + [2 \sum_{j=1}^{2m+1} (1 - c_j - \varepsilon') + \varepsilon' \cdot h]/(1 - \varepsilon') \\
&\leq 1 + [2(2m+1) + \varepsilon'(1 + 2\varepsilon)b]/(1 - \varepsilon') \\
&\quad \text{from } h \leq (1 + 2\varepsilon)b \text{ and } 1 - c_j - \varepsilon' \leq 1 \\
&\leq O(1/\varepsilon^2) + (\varepsilon/2)(2)b \leq \varepsilon b + \varepsilon b = 2\varepsilon b \\
&\quad \text{from } m = 1/(\varepsilon')^2, \varepsilon' = \varepsilon/(2 + \varepsilon),
\end{aligned}$$

for $\varepsilon < 1/50$ and $b \geq 1/\varepsilon^4$. The result of lemma follows. □

3.4.5 Rounding

Finally, we round all values to some discrete points.

Lemma 3.4.5. *If we round up each threshold capacity H_i ($i = 1,\ldots,m$) in $L_{wide}(k_i, k_{i+1}-1, H_i)$ to the the closest value in*

$$CAPACITY = \{t \cdot (\varepsilon')^4 \cdot b | t = 1,2,\ldots,1/(\varepsilon')^6\},$$

and the value of S in $L_{narrow}(S)$ to the closest value in

$$SIZE = \{t \cdot (\varepsilon')^4 \cdot b | t = 1,2,\ldots,1/(\varepsilon')^5\},$$

then the height h of the well-structured packing increases by at most $\Delta_{rounding} \leq \varepsilon b$.

Proof. Consider a well structured packing of all $L_{wide}(k_i, k_{i+1}-1, H_i)$ and $L_{narrow}(S)$ with $2m+1$ layers. Each layer is cut into slices which correspond to a configuration. The wide rectangles of $L_{wide}(k_i, k_{i+1}-1, H_i)$ are packed in the slices of width a_{k_i} in a greedy manner. The rectangles of $L_{narrow}(S)$ are packed by the NFDH heuristic. The height of the packing is

$$h + \Delta_{wide} + \Delta_{narrow} \leq (1+5\varepsilon)b.$$

By rounding, we increase the value of each H_i and S by at most $(\varepsilon')^4 b$. Hence, in solving knapsacks the height of $L_{wide}(k_i, k_{i+1}-1, H_i)$ increases by at most $(\varepsilon')^4 b$, and the area of $L_{narrow}(S)$ increases by at most $(\varepsilon')^4 b$. Next, we proceed as in approximating wide and narrow rectangles. We go over all slices of width a_{k_i} and replace all old wide rectangles by the new wide rectangles in $L_{wide}(k_i, k_{i+1}-1, H_i)$. Also, we go over all layers and replace all old narrow rectangles by the new narrow rectangles in $L_{narrow}(S)$.

In order to accommodate all of wide and narrow rectangles we need to increase the heights of some layers (configurations). We can estimate the total increase as follows. First, we increase the height value of each layer (configuration) by $(\varepsilon')^4 b$.

Then, similar to approximating wide and narrow rectangles, we can pack all the rectangles, but cutting them if they do not fit into slices or layers. Since the height value of any rectangle is at most 1, we simply increase the height of each layer by 1. This eliminates cuts. In overall, we can estimate the total increase as

$$\Delta_{rounding} \leq (2m+1)[(\varepsilon')^4 b+1] = O(\varepsilon^2 b) \leq \varepsilon b,$$

for $m = 1/(\varepsilon')^2$, $\varepsilon' = \varepsilon/(2+\varepsilon)$, $\varepsilon \leq 1/50$ and $b \geq 1/\varepsilon^4$.

The height of the final packing is at most $(1+5\varepsilon)b + \Delta_{rounding} = (1+6\varepsilon)b$. This means that the size of all $L_{wide}(k_i, k_{i+1}-1, H_i)$ and $L_{narrow}(S)$ is at most $(1+6\varepsilon)b$. Hence, after rounding the value of S is at most $(1+6\varepsilon)b \leq b/\varepsilon'$. Since the width value of the rectangles in $L_{wide}(k_i, k_{i+1}-1, H_i)$ is at least ε', after rounding the value of H_i can be at most $(1+6\varepsilon)b/\varepsilon' \leq b/(\varepsilon')^2$. Thus, the value of t in $CAPACITY$ and $SIZE$ can be at most $1/(\varepsilon')^5$ and $1/(\varepsilon')^6$, respectively. The result of lemma follows. $\square$

3.5 OVERALL ALGORITHM

Here we outline our algorithm and summarize all above results. We simply enumerate all possible sequences of threshold rectangles and their capacity values. Then, we solve a series of knapsack problems to get several lists of wide and narrow rectangles, and find a packing for them by using the KR-algorithm. At the end, we select the most profitable packing and apply the shifting technique to it. The final packing fits into the dedicated rectangle and its weight is near-optimal.

RECTANGLE PACKING (RP):
Input: List L, accuracy $\varepsilon > 0$, and $\varepsilon' = \varepsilon/(2+\varepsilon)$, $m = 1/(\varepsilon')^2$.

1. Split L into L_{narrow} and L_{wide} of narrow and wide rectangles, whose widths are at most ε' and larger than ε';
2. Sort the wide rectangles of L_{wide} according to their widths;
3. For each sequence of $m = (1/\varepsilon')$ wide threshold rectangles $R_{k_1}, R_{k_2}, \ldots, R_{k_m}$ from L_{wide}:
   (a) select m capacity values of $H_i \in CAPACITY$ and a value of $S \in SIZE$;
   (b) find m lists $L_{wide}(k_i, k_{i+1}-1, H_i)$ and list $L_{narrow}(S)$;
   (c) run the KR-algorithm and keep the solution (if it's height is at most $(1+16\varepsilon)b$).
4. Select a packing whose weight is maximum;
5. Apply the shifting technique.

We conclude with the following final result.

Theorem 3.5.1. *The* RP*-algorithm outputs a rectangle packing of a sublist $L' \subseteq L$ in the area $[0,a] \times [0,b]$ of the dedicated rectangle R. The weight of the packing*

$$weight(L') \geq (1-\varepsilon)\mathrm{OPT},$$

where OPT *is the optimal weight. The running time of the* RP*-algorithm is bounded by*

$$O(n^{1/\varepsilon^2}(1/\varepsilon^6)^{1/\varepsilon^2+1}[KS(n,\varepsilon) \cdot KR(n,\varepsilon)]),$$

where $KS(n,\varepsilon)$ is the running time of a FPTAS for solving the knapsack problem, and $KR(n,\varepsilon)$ is the running time of the KR-algorithm.

Proof. In the algorithm, for each guess of a sequences of m threshold rectangles we have to solve

$$|CAPACITY|^m \cdot |SIZE| = O((1/\varepsilon^6)^{1/\varepsilon^2+1})$$

knapsack problems, and run the KR-algorithm. Since there are at most n wide rectangles, we have to try at most $n^m = O(n^{\frac{1}{\varepsilon^2}})$ distinct sequences. So, this running time is bounded by

$$\sum_{threshold} (1/\varepsilon^6)^{1/\varepsilon^2+1} KS(n,\varepsilon)KR(n,\varepsilon) = O(n^{1/\varepsilon^2}(1/\varepsilon^6)^{1/\varepsilon^2+1}[KS(n,\varepsilon)\cdot KR(n,\varepsilon)]).$$

Since we enumerate all possible threshold rectangles and capacity values, we also consider the ones which correspond to the optimal solution L^{opt}. Their knapsack solutions have weight at least

$$(1-\varepsilon^2/4)weight(L^{opt}_{wide}) + (1-\varepsilon^2/4)weight(L^{opt}_{narrow}) \leq (1-\varepsilon/3)\mathrm{OPT}.$$

As we have shown in the previous section, the well-structured packing of the knapsack solutions has height at most

$$h + \Delta_{wide} + \Delta_{narrow} + \Delta_{rounding} \leq (1+6\varepsilon)b.$$

So, after applying the KR-algorithm, we get a packing of height

$$(1+\varepsilon)[(1+6\varepsilon)b] + O(1/\varepsilon^2) \leq (1+16\varepsilon)b,$$

for $b \geq 1/\varepsilon^4$.

Finally, by Lemma 3.3.1, in Step 5 the shifting technique must output a packing in the area $[0,1]\times[0,b]$ whose weight is at least $(1-O(\varepsilon))\mathrm{OPT}$. Scaling ε in an appropriate way we can obtain a desired packing with total weight at least $(1-\varepsilon)\mathrm{OPT}$. This completes the proof of the theorem. □

Remark on scaling. In order to obtain a required algorithm as defined in Theorem 3.5.1, we first need to define bound on ε, using the above described algorithm together with Lemma 3.3.1, and then scale ε in an appropriate way. If $b \geq 1/\varepsilon^4$ and $\varepsilon \in (0,1/50]$, then the algorithm outputs a packing whose weight is at least $(1-35\varepsilon)\mathrm{OPT}$ (see Corollary 3.3.2). Hence, we can obtain a required algorithm for $b \geq 1/\varepsilon^4$ and $\varepsilon \in (0,1/1750]$.

3.6 PACKING INTO k RECTANGULAR BINS

Here we consider the problem of packing weighted rectangles into k bins. Given k identical bins of size (a,b) and a list L of n rectangles R_i $(i = 1,\dots,n)$ with widths $a_i \in (0,a]$, heights $b_i \in (0,b]$, and positive integral weights w_i. The goal is to find a sublist $L' \subseteq L$ of rectangles and its packing into k bins such that the total weight of packed rectangles is maximized. We present the following algorithm:

ALGORITHM k-BINS:
Input: List L, accuracy $\varepsilon > 0$, k bins of size (a,b).

Case 1. $k \leq O(1/\varepsilon^4)$. Use a $(\frac{1}{2} - \varepsilon)$-approximation algorithm, that generalizes an approximation algorithm for one bin [51] to a constant number of bins [24].

Case 2. $k > O(1/\varepsilon^4)$.

1. Take all k bins together to get the rectangle (a, kb).
2. Apply our algorithm with the PTAS to pack a subset of rectangles into a larger rectangle (a, kb), that gives us a packing with the total profit $\geq (1-\varepsilon)\mathrm{OPT}$.
3. Take the current rectangle packing. Draw $(k-1)$ vertical lines which divide the packing into k bins.
4. Split this packing into 2 solutions (see Fig. 3.6):
   (a) solution, which contains all rectangles which lie inside of each bin.
   (b) solution, which contains all rectangles which intersect any dividing line between two bins.
5. Take the solution which has the highest profit.

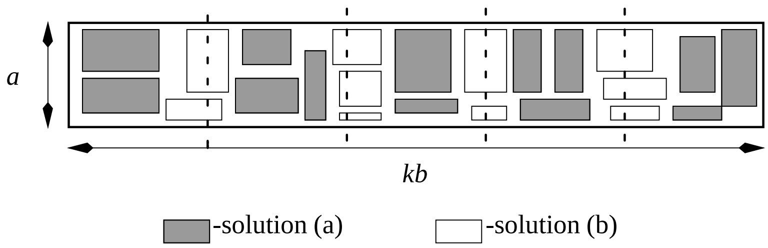


Figure 3.6: Packing into k bins

We can conclude with the following result.

Theorem 3.6.1. *The algorithm k-Bins is a* $(\frac{1}{2} - \varepsilon)$*-approximation algorithm. Its running time is polynomial in the number of rectangles n for any fixed* $\varepsilon > 0$.

Remark. If in the Step 2 of the algorithm k-Bins we replace a PTAS to the FPTAS from Chapter 4, we will automatically get, that the running time of the algorithm k-Bins is polynomial in the number of rectangles n and in $1/\varepsilon$.

3.7 CONCLUDING REMARKS

In this chapter we continue to investigate the influence of the resources. We address the general version of the storage packing problem, where we pack weighted rectangles into a rectangular frame, in the case of large resources, i.e the number of packed rectangles is relatively large. The algorithm we present finds a subset of rectangles and its packing into the dedicated rectangle with weight at least $(1-\varepsilon)$OPT. The running time of the algorithm is polynomial in the number of rectangles. In other words we present a PTAS with large resources. Of course, the challenging question is whether for this version of the storage packing problem we can obtain a more efficient algorithm with a better running time, namely, whether we can obtain an FPTAS. In the next chapter we give a positive answer to this question.

CHAPTER 4

EFFICIENT WEIGHTED RECTANGLE PACKING WITH LARGE RESOURCES

4.1 INTRODUCTION

In this chapter we continue our work on the problem addressed in Chapter 3, namely, on the storage packing problem, where a list of weighted rectangles needs to be packed into a dedicated rectangle so that the total weight of the packed rectangles is maximized. More precisely, we are given again a dedicated rectangle R of width $a > 0$ and height $b > 0$, and a list L of n rectangles R_i ($i = 1, \ldots, n$) of widths $a_i \in (0, a]$ and heights $b_i \in (0, b]$. Each rectangle R_i has a positive weight $w_i > 0$. For any sublist of rectangles $L' \subseteq L$, a *packing* of L' into R is a positioning of the rectangles from L' within the area $[0, a] \times [0, b]$ of R, so that all the rectangles of L' have disjoint interiors. Rectangles are not allowed to rotate. The goal is to find a sublist $L' \subseteq L$, and its packing into R, of maximum total weight, $\sum_{R_i \in L'} w_i$.

Here we again consider the case of large resources, that is, the dedicated rectangle R has width $a > 0$ and height $b > 0$, whereas each rectangle R_i in the list L has width $a_i \in (0, a]$ and height $b_i \in (0, \varepsilon^3 \cdot b]$, for $\varepsilon > 0$. Our aim now is to derive a more efficient approximation algorithm. Using some novel approximation techniques, we significantly improve on the running time of the algorithm. In particular we present an algorithm which finds a packing of a sublist of L into the rectangle R whose total weight is at least $(1 - \varepsilon)\mathrm{OPT}(L)$, where $\mathrm{OPT}(L)$ is the optimum. The running time of the algorithm is polynomial in n and, contrasting to the previous result, is also polynomial in $1/\varepsilon$. In other words we derive a fully polynomial time approximation scheme (FPTAS) with large resources.

Our approach is as follows. At the beginning we relax the problem to *fractional packing*: any rectangle can be first cut by horizontal lines into several rectangles of the same width, and then some of them can be independently packed. The fractional relaxation formulates as a linear program (LP).

In general, the LP consists of an exponential number of variables. Hence, we cannot solve it directly. Our main idea here is to reformulate the LP as an instance of the *resource-sharing* problem and then make use of some recent approximation tools for it (see [40, 47], Section 4.2.2 and Appendix 5.4 for details). This requires a number of subsequent technical results, which, however, we obtain in quite an elegant way.

By approximating a sequence of $O(n/\varepsilon^2)$ instances of the resource-sharing problem, we are able to find an approximate fractional solution. Our next idea is to round this solution. By solving and rounding $O(1/\varepsilon^2)$ instances of the fractional knapsack problem we find a list of rectangles which is quite a good approximation for the original problem. The weight of the list is $(1-O(\varepsilon))$ times the optimum, and a strip packing algorithm [56] can pack it in the area $[0,a]\times[0,(1+O(\varepsilon))b]$.

As soon as such a “near-optimal" packing is found, we apply our shifting technique. This puts the packing into the dedicated rectangle by removing some less weighted part of the packing.

By combining all above ideas and careful analysis of the algorithm we provide here the following result.

Theorem 4.1.1. *There exists some constant* $\beta \geq 4$ *such that for any* $\varepsilon \in (0,1/\beta]$, *any dedicated rectangle* R *of width* $a > 0$ *and height* $b > 0$, *and any list* L *of rectangles* R_i *(*$i = 1,\ldots,n$*) with widths* $a_i \in (0,a]$ *and heights* $b_i \in (0,\varepsilon^3 \cdot b]$, *there exists an algorithm* A_ε *which finds a packing of a sublist of* L *in the area of the dedicated rectangle* R *whose total weight*

$$A_\varepsilon(L) \geq (1-\varepsilon)\mathrm{OPT}(L),$$

where $\mathrm{OPT}(L)$ *is the optimum. The running time of* A_ε *is polynomial in* n *and* $1/\varepsilon$.

Remark. In the theorem we can bound the value β by 2.6×10^3. If we assume that all rectangle widths $a_i \in (0, a]$ and heights $b_i \in (0, \varepsilon^4 \cdot b]$, then the value of β can be reduced to 2.5×10^2.

Organization of the Chapter. The rest of the chapter is organized as follows. Section 4.2 introduces notations, giving some preliminary results. Section 4.3 describes our algorithm. Section 4.4 consists of the analysis of the algorithm. The final section gives some concluding remarks.

4.2 PRELIMINARIES

We will use the following notations. We write (p, q) to denote a rectangle whose width $p > 0$ and height $q > 0$. In the input, we are given a dedicated rectangle $R = (a, b)$, a list L of rectangles $R_i = (a_i, b_i)$ $(i = 1, \ldots, n)$ with positive weights $w_i > 0$, and an accuracy $\varepsilon \in (0, 1]$ such that all $a_i \in (0, a]$ and $b_i \in (0, \varepsilon^3 \cdot b]$. We write $w_{\max} = \max_{i=1}^{n} w_i$ to denote the maximum rectangle weight, and OPT to denote the optimum weight.

For simplicity, we scale all the rectangle widths by a and all the heights by $\max_{R_i \in L} b_i$. Hence, throughout of the chapter we assume w.l.o.g. that each rectangle R_i in the list L has side lengths $a_i, b_i \in (0, 1]$, whereas the dedicated rectangle R has unit width $a = 1$ and height $b \geq 1/\varepsilon^3$. In addition, we also assume w.l.o.g. that $w_{max} \in [\varepsilon, 1]$, $\text{OPT} \in [w_{\max}, n \cdot w_{\max}]$, and ε is selected such that $\varepsilon \in (0, 1/4]$ and $1/\varepsilon$ is integral.

4.2.1 Solving the knapsack problem

In the knapsack problem we are given a knapsack capacity B and a set of n items, where each item i $(i = 1, \ldots, n)$ is associated with its size $s_i \in (0, 1]$ and positive profit $p_i > 0$. It is required to find a subset $I \subseteq \{1, 2, \ldots, n\}$ of items which maximizes the profit, $\sum_{i \in I} p_i$, given that $\sum_{i \in I} s_i \leq B$, i.e. it fits in a knapsack of size B.

This knapsack problem can be formulated as the following integer linear program:

$$\begin{array}{lrcl} \text{maximize} & \sum_{i=1}^{n} z_i \cdot p_i & & \\ \text{subject to} & \sum_{i=1}^{n} z_i \cdot s_i & \leq & B, \\ & z_i & \in & \{0,1\}, \quad \text{for all } i = 1, \ldots, n. \end{array} \tag{4.1}$$

Each z_i decides whether item i belongs to a solution or not. If $z_i = 1$, it does. Otherwise, it does not.

The problem is NP-hard, but it admits an FPTAS [36, 55, 60]: an algorithm which for any accuracy $\delta > 0$ finds a solution whose size is at most B and profit is at least a factor of $(1-\delta)$ of the knapsack optimum $\mathrm{OPT}(B)$. We will write $KS(n,\delta)$ to denote the running time of such an FPTAS, which is polynomial in n and $1/\delta$. (For example, in [60] it is shown that $KS(n,\delta) = O(n/\delta^3)$.)

If all $z_i \in \{0,1\}$ are relaxed to $z_i \in [0,1]$ in the above formulation, then the resulted linear program defines the fractional version of the knapsack problem. This relaxation means that any solution can be fractional. Assume w.l.o.g. that the items are ordered by non-increasing p_i/s_i ratio, i.e.

$$p_1/s_1 \geq p_2/s_2 \geq \ldots \geq p_n/s_n.$$

Then, contrasting with the integral version, any fractional optimal solution rounds to a solution with $z_i = 1$ ($i = 1, \ldots, k-1$), one $z_k \in [0,1]$, and $z_i = 0$ ($i = k+1, \ldots, n$) such that

$$\sum_{i=1}^{n} z_i \cdot s_i = \sum_{i=1}^{k-1} s_i + z_k \cdot s_k = B.$$

Then, the fractional optimum can be defined as

$$\sum_{i=1}^{n} z_i \cdot p_i = \sum_{i=1}^{k-1} p_i + z_k \cdot p_k,$$

that gives an upper bound on the integral knapsack optimum $\mathrm{OPT}(B)$. This fractional optimal solution is called *simple*. Notice that a simple optimal solution can be computed in $O(n \log n)$ time that is required to order the items by non-increasing p_i/s_i ratio.

Assume now that we have a simple optimal solution $z_i \in [0,1]$ ($i = 1,\ldots,n$) as it is described above. Then, by rounding just the value of z_k to 1 we can obtain an integral solution $\bar{z} \in \{0,1\}$ ($i = 1,\ldots,n$), which, however, is not feasible. From another side, its size can be bounded as

$$B \leq \sum_{i=1}^{n} \bar{z} \cdot s_i \leq B + \max_{i=1}^{n} s_i \leq B + 1$$

and the profit value can be bounded as

$$\sum_{i=1}^{n} \bar{z} \cdot p_i \geq \mathrm{OPT}(B).$$

We use this observation in the rounding part of our algorithm, Section 4.3.2.

4.2.2 *Approximating large LPs*

Here we briefly discuss the problem of approximating large LPs. Further information can be found in [40, 47].

Resource-sharing problem. Let M and N be two positive integers. Let B be a non-empty compact convex set in $\mathbb{R}^N$. Let $f_m : B \to \mathbb{R}_+$ ($m = 0,\ldots,M$) be non-negative linear functions over B. Then, the *resource-sharing problem* can be formulated as the following linear program:

$$\begin{array}{rrcl} \text{maximize} & \lambda & & \\ \text{subject to} & f_m(z) & \geq & \lambda, \text{ for } m = 0,\ldots,M. \\ & z & \in & B. \end{array} \tag{4.2}$$

Let λ^* be the optimum. For an accuracy $\bar{\varepsilon} \in (0,1]$, an $\bar{\varepsilon}$-approximate solution is a solution $z \in B$ such that

$$f_m(z) \geq (1 - \bar{\varepsilon})\lambda^*, \text{ for } m = 0,\ldots,M.$$

Block problem. A *price vector* is a vector p of non-negative values $p_m \geq 0$ ($m = 0, \ldots, M$) such that

$$\sum_{m=0}^{M} p_m = 1. \tag{4.3}$$

Then, for any fixed p, the *block problem* is defined as the following linear program:

$$\begin{array}{lrcl} \text{maximize} & \Lambda(p,z) & = & \sum_{m=0}^{M} p_m f_m(z) \\ \text{subject to} & z & \in & B. \end{array} \tag{4.4}$$

Let $\Lambda^*(p)$ be the optimum. For an accuracy $\bar{t} \in (0,1]$, a $(p,\bar{t})$-approximate solution is a solution $z(p) \in B$ such that

$$\Lambda(p,z(p)) \geq (1-\bar{t})\Lambda^*(p).$$

If N is polynomial in M, then we can use any standard LP technique and resolve the above LPs in time polynomial in M. However, in this chapter we meet the case when $N = O(2^M)$, i.e. N can be exponential in M. This means that our LP is large. In order to cope with that, we will use the following result.

Theorem 4.2.1 (Grigoriadis at al. [40], Jansen [47]). *For any given $\bar{\varepsilon} > 0$, there is a resource sharing algorithm $RSA(\bar{\varepsilon})$ which finds an $\bar{\varepsilon}$-approximate solution for the resource-sharing problem, provided that given any $\bar{t} = \Theta(\bar{\varepsilon})$, any price vector p there is a block solver algorithm $BSA(p,\bar{t})$ which finds a $(p,\bar{t})$-approximate solution for the block problem. The algorithm $RSA(\bar{\varepsilon})$ runs as a sequence of $O(M(\ln M + \bar{\varepsilon}^{-2}\ln\bar{\varepsilon}^{-1}))$ iterative steps, each of those requires a call to $BSA(p,\bar{t})$ and incurs an overhead of $O(M \ln\ln(M\bar{\varepsilon}^{-1}))$ elementary operations.*

Remark. The algorithm proposed in [47] uses price vectors p whose positive coordinates $p_m = \Omega([\bar{\varepsilon}/M]^q)$ ($m = 0, 1, \ldots, M$), for a constant $q \in \mathbb{N}$. We use this important fact in the analysis of our algorithm given in Section 4.4.3.

4.2.3 The LP formulation

Here we relax the problem to fractional packing: any rectangle can be first cut by horizontal lines into several rectangles of the same width, and then some of them can be independently packed into the dedicated rectangle. This relaxation can be formulated as an LP. We will use it in the design and analysis of our algorithm described in Sections 4.3 and 4.4.

Fractional packing. Let L be a list of rectangles. Then, for each rectangle R_i ($i = 1,\dots,n$) in L we introduce a variable $x_i \in [0,1]$, whose interpretation will be an x_ith fraction of rectangle R_i that is given as a rectangle $(a_i, x_i \cdot b_i)$ of weight $x_i \cdot w_i$.

For simplicity, we use x to denote the vector of all x_i ($i = 1,\dots,n$), and $L(x)$ to denote the *fractional* list which consists of all rectangles $(a_i, x_i \cdot b_i)$ ($i = 1,\dots,n$). We define the weight of $L(x)$ as the total fractional weight, $\sum_{i=1}^{n} x_i \cdot w_i$. We say that $L(x)$ is *integral* if all $x_i \in \{0,1\}$ ($i = 1,\dots,n$), i.e. $L(x)$ is a sublist of L which consists of the rectangles R_i whose $x_i = 1$.

Let $(1,h)$ be a rectangle of height $h \geq b$. For any fractional list $L(x)$, a *fractional* packing of $L(x)$ into $(1,h)$ is a packing in the area $[0,1] \times [0,h]$ of any list of rectangles obtained from $L(x)$ by subdividing some of its rectangles by horizontal cuts: each rectangle $(a_i, x_i \cdot b_i)$ is replaced by a sequence $(a_i, x_{i_1} \cdot b_i), (a_i, x_{i_2} \cdot b_i), \dots, (a_i, x_{i_k} \cdot b_i)$ of rectangles such that $x_i = \sum_{j=1}^{k} x_{i_j}$.

Configurations. Now we can define configurations. A configuration is a set of rectangles $C \subseteq L$ whose total width is at most 1, i.e. they are able to occur at the same level. Without loss of generality, the configurations can be assumed to be arbitrary ordered.

Let $\#C$ be the number of distinct configurations. (Notice that $\#C$ is $O(2^n)$.) Then, for each configuration C_j we introduce a variable $y_j \geq 0$, whose interpretation will be the height of C_j. For simplicity, we use y to denote the vector of all $y_j \geq 0$ ($j = 1,\dots,\#C$).

Let $(1,h)$ be a rectangle of height $h \geq b$. Then, for any (possibly fractional) packing of $L(x)$ into $(1,h)$ we can define the values of y_j ($j = 1,\ldots,\#C$) in the vector y as follows. We scan the area $[0,1] \times [0,h]$ bottom-up with a horizontal sweep line $y = \overline{h}$, $0 \leq \overline{h} \leq h$. (Here y means the ordinate axis, or Y-line.) Every such line canonically associates to a configuration, that consists of all the rectangles of L whose fractions' interior is intersected by the sweep line. The value of y_j, $1 \leq j \leq \#C$, is equal to the measure of the $\overline{h}$'s such that the sweep line $y = \overline{h}$ is associated to configuration C_j. Thus, the sum of y_j over all configurations C_j is at most h.

For example, let $h = 3$, and $L(x)$ be a list of rectangles $A = (6/7,1)$, $B = (4/7,3/4)$, $C = (3/7,1)$, $D = (3/7,1)$ and $E = (4/7,3/4)$. There are ten configurations: $C_1 = \{A\}$, $C_2 = \{C,B\}$, $C_3 = \{C,D\}$, $C_4 = \{E,D\}$, $C_5 = \{C,E\}$, $C_6 = \{D,B\}$, $C_7 = \{B\}$, $C_8 = \{C\}$, $C_9 = \{D\}$, $C_{10} = \{E\}$. The vector y corresponding to the packing in Fig 4.1 is $(1,3/4,1/4,3/4,0,0,0,0,0,0)$.

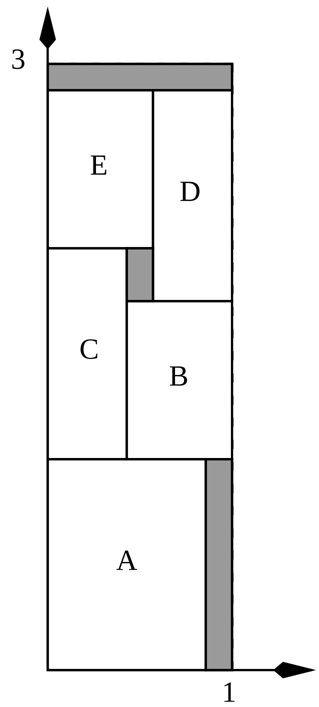


Figure 4.1: A packing of list $L(x) = \{A,B,C,D,E\}$ in the area $[0,1] \times [0,3]$.

LP formulation. Now we combine the two above ideas. First, we relax to a fractional list $L(x)$. Second, we relax to a fractional packing of $L(x)$. The goal is to maximize the fractional weight of $L(x)$. This can be formulated as the following linear program $LP(L,h)$:

$$\begin{array}{rrcll} \text{maximize} & \sum_{i=1}^{n} x_i \cdot w_i & & & \\ \text{subject to} & \sum_{j:R_i \in C_j} y_j & \geq & x_i \cdot b_i, & \text{for all } i = 1,\ldots,n, \\ & \sum_{j=1}^{\#C} y_j & \leq & h, & \\ & y_j & \geq & 0, & \text{for all } j = 1,\ldots,\#C, \\ & x_i & \in & [0,1], & \text{for all } i = 1,\ldots,n. \end{array} \tag{4.5}$$

Each x_i defines an x_ith fraction of rectangle R_i. Each y_j defines the height value of configuration C_j. The objective value defines the total fractional weight. In the first line, the sum of y_j over all configurations C_j that include rectangle R_i is at least x_i times its height b_i. In the second line, the sum of y_j over all configurations C_j is bounded by h. In the last two lines, all y_j are non-negative and all x_i are fractions in $[0,1]$.

One can see that the relaxation of our problem can be formulated as $LP(L,b)$. We can conclude the following result.

Lemma 4.2.2. *Let* $\overline{\mathrm{OPT}}$ *be the optimum of* $LP(L,b)$. *Then,* $\overline{\mathrm{OPT}}$ *is an upper bound on the optimum* OPT *which can be achieved by packing a sublist of* L *into the dedicated rectangle* $R = (1,b)$.

Proof. One can see that any optimal packing of L into $R = (1,b)$ defines a feasible solution for $LP(L,b)$. □

4.2.4 Separating rectangles

Let $\varepsilon' = \varepsilon/(2+\varepsilon)$. Let R_i be a rectangle in the list L. Let a_i be the width of R_i. If the value of a_i is at most ε', then rectangle R_i is called narrow. Otherwise, R_i is called wide. We will write L_{wide} to denote the list of wide rectangles, and

L_{narrow} to denote the list of narrow rectangles, respectively. So, L is partitioned into L_{narrow} and L_{wide}.

4.2.5 The KR-algorithm

We will use the following result which defines a relationship between fractional packing and "non-fractional" packing.

Theorem 4.2.3 (Kenyon & Rémila [57]). *Let $L' \subseteq L$ be an integral list of rectangles. Assume that the rectangles of L' can be fractionally packed in the area $[0,1] \times [0,h]$. Then, there is an algorithm which, given an accuracy $\varepsilon \in (0,1]$, finds a positioning of the rectangles from L' within the vertical strip $[0,1] \times [0,\infty)$ of unit width such that all the rectangles of L' have disjoint interiors and the height to which the strip is filled is bounded by*

$$h' \leq h(1+1/(m\varepsilon'))/(1-\varepsilon') + 4m + 1, \tag{4.6}$$

where $m = \lceil (1/\varepsilon')^2 \rceil$ and $\varepsilon' = \varepsilon/(2+\varepsilon)$. The running time of the algorithm is polynomial in n and $1/\varepsilon$.

For simplicity, such an algorithm is called the KR-algorithm, and its running time is denoted by $KR(n,\varepsilon)$.

Remark. In fact, the algorithm in [57] outputs a (non-fractional) packing of L' in $[0,1] \times [0,\infty)$ whose height can be bounded by

$$\begin{aligned} h' \leq \max\{\ & lin(L' \cap L_{wide})(1+1/(m\varepsilon')) + 2m + 1, \\ & size(L')(1+1/(m\varepsilon'))/(1-\varepsilon') + 4m + 1\ \}, \end{aligned}$$

where $size(L')$ is the area of L' and $lin(L' \cap L_{wide})$ is the height of the optimal fractional strip packing of $L' \cap L_{wide}$. So, in the above theorem we reformulated this result in its weak form. It is enough to mention that $lin(L' \cap L_{wide})$ and $size(L')$ are upper bounded by h.

4.3 THE PACKING ALGORITHM

Our algorithm consists of the three main steps: LP approximation, Rounding, and Shifting. The first step is described in Section 4.3.1, and the next two steps are described in Sections 4.3.2, 4.3.3 respectively. The overall outline of the algorithm is given in Section 5.2.7.

4.3.1 LP approximation

Here we work with the relaxation given by $LP(L,b)$. Due to the fact that the number of configurations $\#C = O(2^n)$, the number of variables in the LP can be exponential in n. Hence we cannot solve it directly. We look for an LP approximation. We transform the LP to the resource-sharing problem. By performing a linear search over approximate solutions for the latter problem, we are able to find a fractional list $L(x)$. This gives quite a good approximation for the relaxation of our problem. Notice that in order to resolve the resource-sharing problem we use the results described in Section 4.2.2. We formulate the block-problem and present a block solver for it. For simplicity, this part of the step is described later in the analysis, Section 4.4.1.

Resource-sharing problem. We can assume w.l.o.g. that the LP optimum $\overline{\mathrm{OPT}}$ is lower bounded by the maximum weight $w_{\max}$ and upper bounded by $n \cdot w_{\max}$, i.e. $\overline{\mathrm{OPT}} \in [w_{\max}, nw_{\max}]$. Then, for each value $w \in [w_{\max}, nw_{\max}]$ we introduce the following resource-sharing problem:

$$\begin{array}{llcll} \text{maximize} & \lambda & & & \\ \text{subject to} & \sum_{i=1}^{n} x_i \cdot (w_i/w) & \geq & \lambda, & \\ & \sum_{j:R_i \in C_j} [y_j/b_i] - x_i + 1 & \geq & \lambda, & \text{for all } i = 1,\ldots,n, \\ & \sum_{j=1}^{\#C} y_j/b & \leq & 1, & \\ & y_j & \geq & 0, & \text{for all } j = 1,\ldots,\#C, \\ & x_i & \in & [0,1], & \text{for all } i = 1,\ldots,n. \end{array} \tag{4.7}$$

Lemma 4.3.1. *Let λ^* be the optimum. If $\lambda^* < 1$, then the value of w is larger than* $\overline{\text{OPT}}$.

Proof. Let x^* and y^* be an optimal solution of $LP(L,b)$. Then,

$$\overline{\text{OPT}} = \sum_{i=1}^{n} x_i^* \cdot w_i.$$

Assume now that $w \leq \overline{\text{OPT}}$, i.e. the value of w is at most $\overline{\text{OPT}}$. Then, in objective

$$\sum_{i=1}^{n} x_i^* \cdot w_i / w = \overline{\text{OPT}} / w \geq 1,$$

and in constraints

$$\begin{array}{rcll} \sum_{j:R_i \in C_j} [y_j^*/b_i] - x_i^* + 1 & \geq & 1, & \text{for all } i = 1, \ldots, n, \\ \sum_{j=1}^{\#C} y_j^*/b & \leq & 1, & \\ y_j^* & \geq & 0, & \text{for all } j = 1, \ldots, \#C, \\ x_i^* & \in & [0,1], & \text{for all } i = 1, \ldots, n. \end{array}$$

This defines a feasible solution in the resource-sharing problem with $\lambda = 1$. Hence, assuming $w \leq \overline{\text{OPT}}$ we can show that $\lambda^* \geq 1$. Thus, from $\lambda^* < 1$ it always follows that $w > \overline{\text{OPT}}$. □

Linear search. Assume that we can solve any instance of the resource-sharing problem to the optimum. Then, we can perform a search at each value

$$w \in \{(1 + \varepsilon^2 \cdot \ell) w_{max} | \ell = 0, 1, \ldots, (n-1)/\varepsilon^2\},$$

and simply take the optimal solution (x,y) given by the maximum value of w whose optimum $\lambda^* \geq 1$. First, this solution (x,y) is feasible for $LP(L,b)$. Second, we know that $\overline{\text{OPT}} \geq w_{\max}$, and, due to the search procedure, $w + \varepsilon^2 w_{max} > \overline{\text{OPT}}$. Hence, the objective value at (x,y) is at least

$$w \geq \overline{\text{OPT}} - \varepsilon^2 w_{max} \geq (1 - \varepsilon^2) \overline{\text{OPT}}.$$

Thus, this solution (x,y) is quite a good approximation for $LP(L,b)$.

Using $\bar{\varepsilon}$-approximate solutions. Since we cannot resolve the problem to the optimum, we use $\bar{\varepsilon}$-approximate solutions. Let $w \in [w_{\max}, n \cdot w_{\max}]$. Let λ^* be the optimum of the resource-sharing problem for given w. Then, for all

$$\lambda \geq \lambda^*(1-\bar{\varepsilon}),$$

an $\bar{\varepsilon}$-approximate solution (x,y) is such that

$$\begin{array}{rcll}
\sum_{i=1}^{n} x_i \cdot (w_i/w) & \geq & \lambda, & \\
\sum_{j:R_i \in C_j} y_j/b_i - x_i + 1 & \geq & \lambda, & \text{for all } i = 1,\dots,n, \\
\sum_{j=1}^{\#C} y_j/b & \leq & 1, & \\
y_j & \geq & 0, & \text{for all } j = 1,\dots,\#C, \\
x_i & \in & [0,1], & \text{for all } i = 1,\dots,n.
\end{array}$$

If $\lambda < (1-\bar{\varepsilon})$, then $\lambda^* < 1$. By Lemma 4.3.1, we can conclude that $\overline{\mathrm{OPT}}$ is smaller than w.

Now we assume that $\lambda \geq (1-\bar{\varepsilon})$ and $\bar{\varepsilon} = \varepsilon^2/n$. For such values of λ and $\bar{\varepsilon}$, we can observe the following three facts. First, consider all $x_i < \varepsilon/n$. Then, we can bound

$$\sum_{R_i \in L \,:\, x_i \,\leq\, \varepsilon/n} x_i \cdot w_i < (\varepsilon/n) \cdot \left[\sum_{i=1}^{n} w_i\right] \;\leq\; \varepsilon \cdot w_{\max}.$$

Second, for each $x_i \geq \varepsilon/n$, we have that

$$x_i - \bar{\varepsilon} = x_i - \varepsilon^2/n \geq x_i - \varepsilon x_i = (1-\varepsilon)x_i.$$

Third, for $\varepsilon \in (0, 1/4]$ we have that $(1-\varepsilon)(1+2\varepsilon) = 1+\varepsilon-2\varepsilon^2 > 1$. Hence,

$$\sum_{j=1}^{\#C} y_j/(1-\varepsilon) \leq b/(1-\varepsilon) \leq b(1+2\varepsilon). \tag{4.8}$$

Using this $\bar{\varepsilon}$-approximate solution (x,y) we can create a new solution as follows. For each $x_i < \varepsilon/n$, we set the value of x_i to 0. Then, from $w \in [w_{\max}, n \cdot w_{\max}]$ the objective function value can be bounded as

$$\begin{aligned}
\sum_{i=1}^{n} x_i \cdot w_i \geq \lambda \cdot w - \varepsilon \cdot w_{\max} &\geq (1-\varepsilon^2/n)w - \varepsilon \cdot w_{\max} \\
&\geq (1-\varepsilon^2/n - \varepsilon)w \geq (1-2\varepsilon)w.
\end{aligned} \tag{4.9}$$

Next, notice the following. If $x_i = 0$, then using $y_j \geq 0$ we obviously get

$$\sum_{j:R_i \in C_j} y_j/b_i \geq 0 = (1-\varepsilon)x_i. \tag{4.10}$$

If $x_i \geq \varepsilon/n$, then using $\lambda \geq (1-\bar{\varepsilon})$ we can bound

$$\sum_{j:R_i \in C_j} y_j/b_i \geq \lambda + x_i - 1 \geq x_i - \bar{\varepsilon} = x_i - \varepsilon^2/n \geq x_i - \varepsilon x_i = (1-\varepsilon)x_i. \tag{4.11}$$

By scaling the values of all y_j ($j = 1,\ldots,\#C$) by $1/(1-\varepsilon)$ in (4.8), (4.10) and (4.11), the new values of all x_i and y_j satisfy

$$\begin{array}{rcll} \sum_{i=1}^{n} x_i \cdot w_i & \geq & (1-2\varepsilon)w, & \\ \sum_{j:R_i \in C_j} y_j & \geq & b_i \cdot x_i, & \text{for all } i = 1,\ldots,n, \\ \sum_{j=1}^{\#C} y_j & \leq & b(1+2\varepsilon), & \\ y_j & \geq & 0, & \text{for all } j = 1,\ldots,\#C, \\ x_i & \in & [0,1], & \text{for all } i = 1,\ldots,n. \end{array} \tag{4.12}$$

Modified linear search. Now we can modify our linear search. We define $\bar{\varepsilon} = \varepsilon^2/n$. We perform a search at each value

$$w \in \{(1+\varepsilon^2 \cdot \ell)w_{max} | \ell = 0,1,\ldots,(n-1)/\varepsilon^2\}.$$

Each time we find an $\bar{\varepsilon}$-approximate solution, and then modify it as shown above. We take the modified $\bar{\varepsilon}$-approximate solution (x,y) given by the maximum value of w. Form (4.12) we can conclude that (x,y) is feasible for $LP(L,(1+2\varepsilon)b)$. Furthermore, the objective function value at (x,y) is at least

$$(1-\varepsilon^2)(1-2\varepsilon)\overline{\mathrm{OPT}} \geq (1-3\varepsilon)\overline{\mathrm{OPT}}.$$

Combining all the ideas we can conclude with the following result.

Lemma 4.3.2. *Let $\varepsilon \in (0,1/4]$, $\bar{\varepsilon} = \varepsilon^2/n$ and $h = (1+2\varepsilon)b$. Then, by performing the modified linear search over $\bar{\varepsilon}$-approximate solutions for a sequence of n/ε^2 instances of the resource-sharing problem, one can determine a feasible solution (x,y) for $LP(L,h)$ whose objective function is at least $(1-3\varepsilon)\overline{\mathrm{OPT}}$.*

Corollary 4.3.3. *Let* $\varepsilon \in (0,1/4]$ *and* $\bar{\varepsilon} = \varepsilon^2/n$*. Then, by performing the modified linear search over* $\bar{\varepsilon}$*-approximate solutions for a sequence of* n/ε^2 *instances of the resource-sharing problem, one can determine a fractional list* $L(x)$ *such that the rectangles of* $L(x)$ *can be fractionally packed in the area* $[0,1] \times [0,(1+2\varepsilon)b]$*, and the weight of* $L(x)$ *is at least* $(1-3\varepsilon)\mathrm{OPT}$.

4.3.2 Rounding

Here we show how our fractional list $L(x)$ can be rounded to an integral list $L(\bar{x}) \subseteq L$. We relay on the procedure of rounding of a simple optimal solution of the fractional knapsack problem, as it is described in Section 4.2.1. We handle narrow and wide rectangles separately, using some techniques from [56].

Rounding narrow rectangles. Here we first define the size of all fractional narrow rectangles as

$$S = \sum_{R_i \in L_{narrow}} x_i \cdot (b_i \cdot a_i).$$

Next, we work with rectangles as items. We formulate the following fractional knapsack problem:

$$\begin{array}{lrcl} \text{maximize} & \sum_{R_i \in L_{narrow}} \bar{x}_i \cdot w_i, & & \\ \text{subject to} & \sum_{R_i \in L_{narrow}} \bar{x}_i \cdot (a_i \cdot b_i) & \leq & S, \\ & \bar{x}_i & \in & [0,1], \quad \text{for all } R_i \in L_{narrow}. \end{array}$$

We find a simple optimal solution, and then round it to an integral solution. This defines some integral value $\bar{x}_i \in \{0,1\}$ for each narrow rectangle R_i in L_{narrow}. We can provide the following result.

Lemma 4.3.4. *For the list of narrow rectangles* L_{narrow}*, in* $O(n \log n)$ *time one can round the fractional list* $L_{narrow}(x)$ *to an integral list* $L_{narrow}(\bar{x}) \subseteq L_{narrow}$*. The size of* $L_{narrow}(\bar{x})$ *differs from the size of* $L_{narrow}(x)$ *by at most* 1*, the maximum size of one rectangle. The weight of* $L_{narrow}(\bar{x})$ *is at least the weight of* $L_{narrow}(x)$*.*

Proof. One can see that $L_{narrow}(x)$ defines a feasible fractional solution. Hence, the bounds easily follow from the knapsack rounding procedure, and the fact that all $a_i, b_i \in (0,1]$. $\square$

Rounding wide rectangles. Here we first order all the wide rectangles in L_{wide} by non-increasing widths. We assume w.l.o.g. that there are n' wide rectangles $R_1 = (a_1, b_1), R_2 = (a_2, b_2), \ldots, R_{n'} = (a_{n'}, b_{n'})$ with widths $a_1 \geq a_2 \geq \ldots \geq a_{n'} \geq \varepsilon'$. We define the height of all fractional wide rectangles as

$$H = \sum_{R_i \in L_{wide}} x_i \cdot b_i.$$

Next, for each wide rectangle R_i in L_{wide} we take its x_ith fraction $(a_i, x_i \cdot b_i)$. Then, we stack up all these fractions by order of non-increasing widths, i.e. from 1 to n'. This gives a left-justified stack whose total height is equal to H.

Let $m = \lceil 1/(\varepsilon')^2 \rceil$. We define m threshold rectangles as follows. We draw $m-1$ horizontal lines at points $y = k \cdot [(\varepsilon')^2 \cdot H]$, for k between 1 and $m-1$, see Fig. 4.2. The kth threshold rectangle is defined as a fractional rectangle whose interior or lower boundary is intersected by the kth line, respectively.

These $m-1$ threshold rectangles separate the list L_{wide} of all wide rectangles into m non-intersecting groups. Each threshold rectangle has the least width in its group.

Let $L_{wide}^{(k)}$ be the kth group ($k = 1, \ldots, m$). We define its fractional height

$$H^{(k)} = \sum_{R_i \in L_{wide}^{(k)}} x_i \cdot b_i.$$

Next, we work with wide rectangles as items. For each group $L_{wide}^{(k)}$ ($k = 1, \ldots, m$), we formulate the following fractional knapsack problem:

$$\begin{array}{lrcl} \text{maximize} & \sum_{R_i \in L_{wide}^{(k)}} \bar{x}_i \cdot w_i, & & \\ \text{subject to} & \sum_{R_i \in L_{wide}^{(k)}} \bar{x}_i \cdot b_i & \leq & H^{(k)}, \\ & \bar{x}_i & \in & [0,1], \quad \text{for all } R_i \in L_{wide}^{(k)}. \end{array}$$

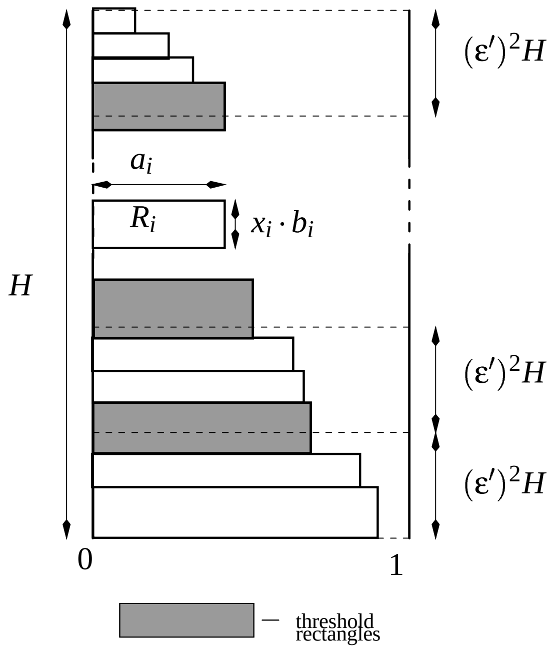


Figure 4.2: Threshold rectangles

We find a simple optimal solution, and then round it to an integral solution. This defines some integral value $\bar{x}_i \in \{0,1\}$ for each wide rectangle R_i in group $L^{(k)}_{wide}$. We can provide the following simple result.

Lemma 4.3.5. *For each kth group of wide rectangles $L^{(k)}_{wide}$, in $O(n\log n)$ time one can round the fractional list $L^{(k)}_{wide}(x)$ to an integral list $L^{(k)}_{wide}(\bar{x}) \subseteq L^{(k)}_{wide}$. The height of $L^{(k)}_{wide}(\bar{x})$ differs from the height of $L^{(k)}_{wide}$ by at most 1, the maximum height of one rectangle. The weight of $L^{(k)}_{wide}(\bar{x})$ is at least the weight of $L^{(k)}_{wide}(x)$.*

Proof. One can see that $L^{(k)}_{wide}(x)$ defines a feasible fractional solution. Hence, the bounds easily follow from the knapsack rounding procedure, and the fact that all $a_i, b_i \in (0,1]$. □

Applying the KR-algorithm. Our next idea is to apply the KR-algorithm to the rounded integral list $L(\bar{x})$. Combining all the ideas, we can prove the following result.

Lemma 4.3.6. *Let* $\varepsilon \in (0,1/4]$. *Then, by solving and rounding* $O(1/\varepsilon^2)$ *instances of the fractional knapsack problem one can round the fractional list* $L(x)$ *to an integral list* $L(\bar{x}) \subseteq L$ *such that the weight of* $L(\bar{x})$ *is at least the weight of* $L(x)$, *and the KR-algorithm outputs a packing of* $L(\bar{x})$ *in the area* $[0,1]\times[0,h']$, *where*

$$h' \leq (1+O(\varepsilon))b + O(1/\varepsilon^2).$$

The complete rounding and packing procedure requires at most $O(\lceil 1/\varepsilon^2 \rceil \cdot (n\log n) + KR(n,\varepsilon))$ *running time.*

Proof. The proof is given in Section 4.4.2. □

Corollary 4.3.7. *The weight of* $L(\bar{x})$ *is at least* $(1-\delta_1\cdot\varepsilon)\mathrm{OPT}$, *where* $\delta_1 = 3$. *Let* $\alpha \geq 1$ *and* $\beta \geq 4$. *Let* $b \geq (\alpha/\varepsilon^3)$ *and* $\varepsilon \in (0,1/\beta]$. *Then, one can obtain a packing of* $L(\bar{x})$ *in the area* $[0,1]\times[(1+\delta_2\cdot\varepsilon)b]$, *where* $\delta_2 = 4+(33/\beta)+(82/\beta^2)$ *if* $\alpha = 1/\varepsilon$, *and* $\delta_2 = 32+(45/\beta)+(42/\beta^2)$ *if* $\alpha = 1$.

Proof. The proof is given in Section 4.4.2. □

Remark: One can also obtain slightly different bounds by taking $\alpha \in \{10,20\}$.

4.3.3 Shifting

Assume that we are given a packing of the rounded integral list $L(\bar{x}) \subseteq L$ in the area $[0,1]\times[0,(1+\delta_2\cdot\varepsilon)b]$, whose weight is at least $(1-\delta_1\cdot\varepsilon)\mathrm{OPT}$, for some $\delta_1,\delta_2 = O(1)$. The idea of our shifting technique is to remove some less weighted piece of height $(\delta_2\cdot\varepsilon)b$ roughly. Then, the weight of the packing remains $(1-O(\varepsilon))\mathrm{OPT}$, but its height reduces to b, giving a packing in the area $[0,1]\times[0,b]$ of the dedicated rectangle $R=(1,b)$.

Recall that $\delta_2 = O(1)$ and $b \geq 1/\varepsilon^3$. We can assume w.l.o.g. that $weight(L(\bar{x})) \leq 2\mathrm{OPT}$, i.e. the weight of $L(\bar{x})$ is not larger than 2OPT. If it is larger than 2OPT, we could proceed as follows. We take the current packing of $L(\bar{x})$ of height

$(1+\delta_2\cdot\varepsilon)b$. Then, we cut it by a horizontal line at height point b. This gives the two packings of height at most b and at most $(\delta_2\cdot\varepsilon)b+1$, respectively. For an illustration see Fig. 4.3 a). So, either of the packings can be considered as a feasible packing in the area of the dedicated rectangle $R=(1,b)$. Furthermore, one of them must have the weight value larger than OPT. This gives a contradiction.

Now we define

$$k=\left\lfloor\frac{(1+\delta_2\cdot\varepsilon)b+2}{(\delta_2\cdot\varepsilon)b+2}\right\rfloor.$$

Since $b\geq 1/\varepsilon^3$ and $\varepsilon\in(0,1/4]$ we also have that

$$\begin{aligned} k &= \left\lfloor\frac{b}{(\delta_2\cdot\varepsilon)b+2}+1\right\rfloor \geq \left\lfloor\frac{1}{(\delta_2\cdot\varepsilon)+(2/b)}+1\right\rfloor \\ &\geq \left\lfloor\frac{1}{(\delta_2\cdot\varepsilon)+2\varepsilon^3}+1\right\rfloor \geq \left\lfloor\frac{1}{\varepsilon(\delta_2+1)}+1\right\rfloor. \end{aligned}$$

Assume now that

$$1/\varepsilon \;\geq\; \delta_2+1. \tag{4.13}$$

Then, $k\geq 2$. Next, we proceed as follows. We take the current strip packing of length $(1+(\delta_2\cdot\varepsilon))b$. We draw $k-1$ horizontal lines which divide the packing into k cuts, as shown in Fig. 4.3 b). Each of the cuts has the inner part of height $(\delta_2\cdot\varepsilon)b$ and the outer part of height 2. So, the height of the k cuts is $((\delta_2\cdot\varepsilon)b+2)k-2\leq(1+\delta_2\cdot\varepsilon)b$.

Let G_i be the list of rectangles which intersect the inner part of the ith cut. Each outer part has height 2, but no rectangle in the list L can be higher than 1. Hence, we have that $G_i\cap G_j=\emptyset$ for $i\neq j$. Furthermore,

$$\sum_{i=1}^{k} weight(G_i)\leq weight(L(\bar{x}))\leq 2\text{OPT}.$$

Since

$$k\geq\left\lfloor\frac{1}{\varepsilon(\delta_2+1)}+1\right\rfloor\geq\frac{1}{\varepsilon(\delta_2+1)},$$

there must exist at least one list G_ℓ such that

$$weight(G_\ell)\leq[2\text{OPT}](1/k)\leq 2\varepsilon(\delta_2+1)\text{OPT}.$$

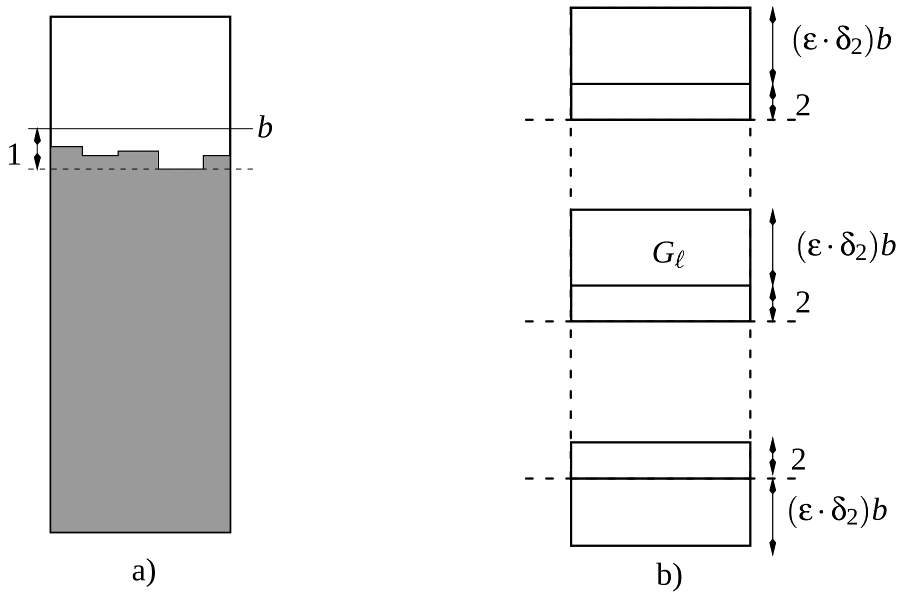


Figure 4.3: Shifting

So, we break the strip packing into two ones from both sides of the inner part of the ℓth cut. Next, we throw away the rectangles of G_ℓ, and put these two strip packing together. This gives a strip packing of height b. Its weight is bounded below by

$$\begin{aligned}(1-\delta_1\cdot\varepsilon)\mathrm{OPT}-\mathit{weight}(G_\ell) &\geq (1-\delta_1\cdot\varepsilon)\mathrm{OPT}-2\varepsilon(\delta_2+1)\mathrm{OPT}\\ &= (1-(\delta_1+2\delta_2+2)\varepsilon)\mathrm{OPT}.\end{aligned}$$

The construction requires at most $O(n+k)$ time. From $\delta_1,\delta_2=O(1)$, this turns to $O(n+1/\varepsilon)$. Combining these ideas with Lemma 4.3.6, we can conclude with the following result.

Lemma 4.3.8. *Let* $\delta_1,\delta_2=O(1)$. *Given a packing in the area* $[0,1]\times[0,(1+\delta_2\cdot\varepsilon)b]$ *whose weight is at least* $(1-\delta_1\cdot\varepsilon)\mathrm{OPT}$, *in* $O(n+1/\varepsilon)$ *time one can obtain a packing in the area* $[0,1]\times[0,b]$ *whose weight is at least* $(1-(\delta_1+2\delta_2+2)\varepsilon)\mathrm{OPT}$, *provided* $1/\varepsilon\geq\delta_2+1$.

Corollary 4.3.9. *Let* $\alpha\geq 1$ *and* $\beta\geq 4$. *Let* $b\geq\alpha/\varepsilon^3$ *and* $\varepsilon\in(0,1/\beta]$. *Then, given a packing of* $L(\bar{x})$ *in the area* $[0,1]\times[0,(1+\delta_2\cdot\varepsilon)b]$ *whose weight is at* *least* $(1-\delta_1 \cdot \varepsilon)\mathrm{OPT}$*, in time* $O(n+1/\varepsilon)$ *one can obtain a packing in the area* $[0,1]\times[0,b]$ *whose weight is at least* $(1-22\varepsilon)\mathrm{OPT}$ *if* $\alpha = 1/\varepsilon$ *and* $\beta = 10$*, and at least* $(1-72\varepsilon)\mathrm{OPT}$ *if* $\alpha = 1$ *and* $\beta = 35$*.*

Proof. Let $b \geq 1/\varepsilon^4$, $\varepsilon \in (0,1/\beta]$, $\delta_1 = 3$, and $\delta_2 = 4+(33/\beta)+(82/\beta^2)$. Then, for $\beta = 10$ we have that

$$\frac{1}{\varepsilon} \;\geq\; \beta = 10 \;\geq\; 1+\delta_2 = 5+(33/10)+(82/100).$$

Hence, by Corollary 4.3.7 and Lemma 4.3.8, the shifting procedure outputs a packing whose weight is at least

$$(1-(\delta_1+2\delta_2+2)\varepsilon)\mathrm{OPT} \geq (1-22\varepsilon)\mathrm{OPT}.$$

Let $b \geq 1/\varepsilon^3$, $\varepsilon \in (0,1/\beta]$, $\delta_1 = 3$, and $\delta_2 = 32+(45/\beta)+(42/\beta^2)$. Then, for $\beta = 35$ we have that

$$\frac{1}{\varepsilon} \geq \beta = 35 \geq 1+\delta_1 = 33+(45/33)+(42/33^2).$$

Hence, by Corollary 4.3.7 and Lemma 4.3.8, the shifting procedure outputs a packing whose weight is at least

$$(1-(\delta_1+2\delta_2+2)\varepsilon)\mathrm{OPT} \geq (1-72\varepsilon)\mathrm{OPT}.$$

□

4.3.4 The overall algorithm

Here we describe an outline of our algorithm. In the following sections we give more details for each step.

ALGORITHM A_ε:

Input: List L of rectangles, dedicated rectangle $R = (1, b)$, accuracy $\varepsilon > 0$.

Output: A sublist of L and its packing in the area of R.

1. **[LP approximation]** Define $\bar{\varepsilon} = \varepsilon^2/n$. Perform the modified linear search over $\bar{\varepsilon}$-approximate solutions for a sequence of n/ε^2 instances of the resource-sharing problem. This defines a fractional list $L(x)$. The weight of $L(x)$ is at least $(1-3\varepsilon)$OPT. The rectangles of $L(x)$ can be fractionally packed in the area $[0,1] \times [0,(1+2\varepsilon)b]$.

2. **[Rounding]** Define $\varepsilon' = \varepsilon/(2+\varepsilon)$ and $m = \lceil 1/(\varepsilon')^2 \rceil$. Perform the partition $L = L_{wide} \cup L_{narrow}$ to set aside the rectangles of width less than ε'. Sort L_{wide} in order of non-increasing widths. Define $m-1$ threshold rectangles in $L_{wide}(x)$. They partition L_{wide} into m groups $L_{wide}^{(k)}$, $k = 1, \ldots, m$. Using $L(x)$, for L_{narrow} and each group $L_{wide}^{(k)}$ $(k = 1, \ldots, m)$ formulate an instance of the fractional knapsack problem, $O(1/\varepsilon^2)$ instances in total. Find a simple optimal solution for each of these instances, and then round them. This rounds $L(x)$ to an integral list $L(\bar{x}) \subseteq L$. The weight of $L(\bar{x})$ is at least $(1-3\varepsilon)$OPT. Apply the KR-algorithm on $L(\bar{x})$ with accuracy ε. This gives a packing of $L(\bar{x})$ in the area $[0,1] \times [0,(1+O(\varepsilon))b + O(1/\varepsilon^2)]$.

3. **[Shifting]** Apply the shifting technique to the current packing. This defines a sublist of $L(\bar{x})$ and its packing in the area $[0,1] \times [0,b]$ of the dedicated rectangle R. The weight of the packing is at least $(1 - O(\varepsilon))$OPT.

Remark on scaling. In order to obtain a required algorithm as defined in Theorem 4.1.1, we first need to define bounds on b and ε, use the above described algorithm together with Lemmas 4.3.2, 4.3.6, 4.3.8, and then scale ε in an appropriate way. If $b \geq 1/\varepsilon^4$ and $\varepsilon \in (0, 1/10]$, then the algorithm outputs a packing whose weight is at least $(1-22\varepsilon)$OPT. If $b \geq 1/\varepsilon^3$ and $\varepsilon \in (0, 1/35]$, the algorithm outputs a packing whose weight is at least $(1-72\varepsilon)\mathrm{OPT}$. Hence, we can obtain a required algorithm either for $b \geq 1/\varepsilon^4$ and $\varepsilon \in (0, 1/220]$, or for $b \geq 1/\varepsilon^3$ and $\varepsilon \in (0, 1/2520]$. This gives quite close bounds on b. In the first case $b \approx 2.4 \times 10^9$. In the second case $b \approx 1.6 \times 10^{10}$.

Remark on efficiency. Notice that some steps of our algorithm can be performed in a more efficient way. For example, one can use a binary search at Step 1. Here we mainly concentrate our attention on the polynomial time efficiency of the algorithm.

4.4 THE ANALYSIS

There are two parts in the analysis of our algorithm. First, we need to show that the three algorithm's steps can be performed in time polynomial in n and $1/\varepsilon$. Second, we need to show that any packing output by our algorithm is “near" optimal. Regarding running time, Step 2 and 3 relay on solving fractional knapsacks and applying the KR-algorithm along with the shifting technique, that can be done efficiently. Hence, the only one bottleneck lies in Step 1 where it is required to find approximate solutions for the resource-sharing problem. However, here we can use the results of Theorem 4.2.1. In Section 4.4.1 we show that approximate solutions for the associated block problem can be found in an efficient way. Regarding a “near" optimal, we give a proof for Lemma 4.3.6 in Section 4.4.2. Finally, we give the overall analysis in Section 4.4.3, that completes the proof of Theorem **??**.

4.4.1 The running time: Approximating the block problem

Here we first recall the resource-sharing problem given in Section 4.3.1. Then, we formulate the block problem. We show that this problem can be rewritten as two linear programs which then shown to be efficiently solved.

Resource-sharing. Let $w \in [w_{\max}, nw_{\max}]$. Recall that the resource-sharing problem is defined as follows:

$$
\begin{array}{llcll}
\text{maximize} & \lambda & & & \\
\text{subject to} & \sum_{i=1}^{n} x_i \cdot (w_i/w) & \geq & \lambda & \\
 & \sum_{j:R_i \in C_j} [y_j/b_i] - x_i + 1 & \geq & \lambda & \text{for all } i = 1, \ldots, n, \\
 & \sum_{j=1}^{\#C} y_j/b & \leq & 1, & \\
 & y_j & \geq & 0, & \text{for all } j = 1, \ldots, \#C, \\
 & x_i & \in & [0,1], & \text{for all } i = 1, \ldots, n.
\end{array}
\tag{4.14}
$$

Let x and y denote the vectors of all x_i's and y_j's. Let $B(x)$ be the set of all x such that

$$x_i \in [0,1] \text{ for all } i = 1, \ldots, n. \tag{4.15}$$

Let $B(y)$ be the set of all y such that

$$
\begin{array}{rcll}
\sum_{j=1}^{\#C} y_j/b & \leq & 1, & \\
y_j & \geq & 0, & \text{for all } j = 1, \ldots, \#C.
\end{array}
\tag{4.16}
$$

Then, $B(x)$ and $B(y)$ are both non-empty, compact and convex.

Block problem. For any given price vector p of non-negative values $p_i \geq 0$ ($i = 0, \ldots, n$) such that

$$\sum_{i=0}^{n} p_i = 1$$

we can define the objective function of the block problem as

$$\Lambda(p,x,y) = p_0[\sum_{i=1}^{n} x_i \cdot (w_i/w)] + \sum_{i=1}^{n} p_i[\sum_{j:R_i \in C_j} y_j/b_i - x_i + 1]. \tag{4.17}$$

For simplicity, we combine the coefficients for each of the variables. For x_i and y_j we get

$$c_i = p_0(w_i/w) - p_i \sum_{j:R_i \in C_j} 1 \tag{4.18}$$

and

$$d_j = \sum_{R_i \in C_j} p_i/b_i, \tag{4.19}$$

respectively. Hence, we can formulate the block problem as follows:

$$\begin{array}{lrcl} \text{maximize} & \Lambda(p,x,y) & = & \sum_{i=1}^{n} c_i \cdot x_i + \sum_{j=1}^{\#C} d_j \cdot y_j \\ \text{subject to} & x & \in & B(x), \\ & y & \in & B(y). \end{array} \tag{4.20}$$

Notice that x and y are independent. Thus, the block problem rewrites as the two linear programs:

$$\begin{array}{lrcl} \text{maximize} & \Lambda(p,x) & = & \sum_{i=1}^{n} c_i \cdot x_i \\ \text{subject to} & x & \in & B(x), \end{array} \tag{4.21}$$

and

$$\begin{array}{lrcl} \text{maximize} & \Lambda(p,y) & = & \sum_{j=1}^{\#C} d_j \cdot y_j \\ \text{subject to} & y & \in & B(y). \end{array} \tag{4.22}$$

The problems are both simple. It is quite an easy task to define optimal solutions for them in an analytical way. We can conclude with the following result.

Lemma 4.4.1. *Let x^* and y^* be defined such that*

- *$x_i^* = 0$ if c_i is non-positive, and $x_i^* = 1$ otherwise ($i = 1, \ldots, n$),*
- *$y_k^* = b$, and $y_j^* = 0$ for all $C_j \neq C_k$ ($j = 1, \ldots, \#C$), where C_k is a configuration with $d_k = \max_{j=1}^{\#C} d_j$.*

Then, x^* and y^* define an optimal solution for the block problem.

Approximation. Recall that the number of configurations $\#C$ can be exponential. Hence, we cannot find an optimal solution as defined above in a straightforward way. Our idea is to look for an approximation. In order to determine x^* we can apply the following result.

Lemma 4.4.2. *Let T be some positive value. If $p_i = \Omega(1/T)$, then there is an algorithm which in $O(n \cdot T)$ time decides whether c_i is non-positive.*

Proof. Our task is to decide whether

$$c_i = p_0(w_i/w) - p_i \sum_{j:R_i \in C_j} 1$$

is non-positive. Notice that in c_i we sum up over all configurations C_j that include rectangle R_i. Hence, equally, we need to solve the following decision problem: Given a rectangle R_i and the list of all rectangles L, is the number of configurations C_j of L that include R_i is at least $K_i := (w_i/w)(p_0/p_i)$?

Recall that any configuration is a set of rectangles whose total width is at most 1. So, one way to solve the problem is to generate a list of all configurations, each of those include R_i. Each configuration in the list is represented by a pair (C,A), where C is a set of rectangles and A is their total width.

Initially, $U := L \setminus \{R_i\}$ and only $(\{R_i\}, a_i)$ placed in the list. Until U is not empty, iterate: (1) take a rectangle R_ℓ from U and scan the list; (2) from each pair (C,A) in the list form a “candidate” $(C \cup \{R_\ell\}, A + a_\ell)$, provided that $A + a_\ell$ is at most 1, i.e. it gives a configuration; (3) merge the existing list and the list of candidates; (4) delete R_ℓ from U.

At the end of the procedure, each pair in the list represents a configuration of L that includes rectangle R_i, and each such configuration is represented by a pair. Hence, if at the end of the procedure the size of the list is at least K_i, the answer to the above question is “YES”, and “NO” otherwise.

We do not affect either answer if no candidates are produced as soon as the size of the list becomes larger than K_i. The procedure is now revised as follows. In the end of each iteration, check the size of the list. If it is smaller than $\lceil K_i \rceil$ (this is true initially), proceed with no changes. Otherwise, skip in the next iteration steps (2) and (3).

Since $|L \setminus \{R_i\}| = O(n)$, there are at most $O(n)$ iterations. The size of the list at each iteration is $O(\lceil K_i \rceil)$. Hence, the running time of the above procedure is $O(n \cdot K_i)$.

As we defined before, $w \in [w_{\max}, n \cdot w_{\max}]$, all p_i are positive and $\sum_{i=0}^{n} p_i = 1$. Hence, $K_i = (w_i/w)(p_0/p_i) = O(1/p_i)$. Assuming that $p_i = \Omega(1/T)$, we get $K_i = O(T)$. Substituting, we finally have $O(n \cdot T)$ for the running time. $\square$

It is more hard to handle y^*. However, we apply the following approximation result.

Lemma 4.4.3. *There is an algorithm which for any given accuracy $\bar{t} > 0$ finds a configuration C_ℓ with $d_\ell \geq (1-\bar{t}) \max_{j=1}^{\#C} d_j$ in $KS(n,\bar{t})$ time, that is required to approximate a knapsack instance with n items and accuracy $\bar{t} > 0$.*

Proof. Our original task is to find a configuration C_k of the maximum value

$$d_k = \max_{j=1}^{\#C} d_j = \max\{\sum_{R_i \in C_j} p_i/b_i \mid j = 1\ldots,\#C\}.$$

Consider the following instance of the knapsack problem. There are n items (rectangles) R_i ($i = 1,\ldots,n$) with sizes b_i and profits p_i/b_i, and a knapsack of capacity $B = 1$. It is required to find a set of items (rectangles) whose total size is at most B and the total profit is maximum.

Recall that any configuration is a set of rectangles whose total width is at most 1. Hence, any knapsack solution is feasible if and only if it forms a configuration. Furthermore, the profit of any configuration C_j ($j = 1,\ldots,\#C$) is equal to the value of d_j. Thus, the knapsack optimum is equal to $d_k = \max_{j=1}^{\#C} d_j$.

We simply run an FPTAS for the knapsack problem with given accuracy $\bar{t} > 0$. This gives a configuration C_ℓ such that

$$d_\ell \geq (1-\bar{t})d_k = (1-\bar{t}) \max_{j=1}^{\#C} d_j.$$

The result of lemma follows. □

Combining all above ideas we can prove the following result.

Lemma 4.4.4. *Let T be some positive value. Then, for any price vector p whose positive coordinates $p_i = \Omega(1/T)$ ($i = 0,\ldots,n$) and any accuracy $\bar{t} > 0$, there is a block solver algorithm $BSA(p,\bar{t})$ which finds a $(p,\bar{t})$-approximate solution for the block problem in $O(n^2 \cdot T) + KS(n,\bar{t})$ time.*

Proof. By Lemma 4.4.1, an optimal solution x^* can be defined by setting $x_i^* = 0$ if c_i is non-positive, and $x_i^* = 1$ otherwise ($i = 1,\ldots,n$). We simply apply an algorithm from Lemma 4.4.2 for each c_i ($i = 1, \ldots, n$). So, we obtain an algorithm which finds x^* in $O(n \cdot [n \cdot T])$ time.

By Lemma 4.4.1, an optimal solution y^* can be found by setting $y_k^* = b$, and $y_j^* = 0$ for all $C_j \neq C_k$ ($j = 1, \ldots, \#C$), where C_k is a configuration with $d_k = \max_{j=1}^{\#C} d_j$. Here we find an approximation y' for y^*. We take an accuracy $\bar{t} > 0$ and apply an algorithm from Lemma 4.4.3. This gives a configuration C_ℓ such that

$$d_\ell \geq (1 - \bar{t}) d_k = (1 - \bar{t}) \max_{j=1}^{\#C} d_j.$$

Then, we define $y' \in B(y)$ by setting $y'_\ell = b$ for C_ℓ, and $y_j = 0$ for all $C_j \neq C_\ell$ ($j = 1, \ldots, \#C$). So, we obtain an algorithm which finds y' in $KS(n, \bar{t})$ time, that is requited to approximate an instance of the knapsack problem with n items and accuracy $\bar{t}$.

Now we can compare the objective function values of y^* and y' as follows

$$\sum_{j=1}^{\#C} d_j \cdot y'_j = d_\ell \cdot b \geq [(1 - \bar{t}) \max_{j=1}^{\#C} d_j] \cdot b = [(1 - \bar{t}) d_k] \cdot b = (1 - \bar{t}) \sum_{j=1}^{\#C} d_j \cdot y_j^*.$$

Hence, combining x^* and y', we get a $(p, \bar{t})$-approximate solution for the block problem. The result of lemma follows. □

4.4.2 A near-optimal packing: Proof of Lemma 4.3.6

Recall Corollary 4.3.3. Let $L(x)$ be the fractional list given by an LP approximation. The weight of $L(x)$ is at least $(1 - 3\varepsilon)\mathrm{OPT}$. There is a fractional packing of $L(x)$ in the area $[0, 1] \times [0, (1 + 2\varepsilon)b]$.

Let $L(\bar{x})$ be the integral list found by rounding of $L(x)$. First, we need to show that the weight of $L(\bar{x})$ is at least the weight of $L(x)$. Second, we need to show that the KR-algorithm finds a packing of the rectangles of $L(\bar{x})$ in the area $[0, 1] \times [0, (1 + O(\varepsilon))b + O(1/\varepsilon^2)]$.

In the rounding procedure by using $L(x)$ we formulate $O(1/\varepsilon^2)$ instances of the fractional knapsack problem. One can see that $L(x)$ defines a feasible solution for each of the instances. Since $L(\bar{x})$ is found by rounding simple optimal solutions, the weight of $L(\bar{x})$ is at least of the weight of $L(x)$.

It remains to show how the KR-algorithm can pack the rectangles of $L(\bar{x})$. As a tool, we use the results of Theorem 4.2.3. We also use the facts that $L(x)$ and $L(\bar{x})$ are quite similar, and that there is a fractional packing of $L(x)$ in the area $[0,1] \times [0,(1+2\varepsilon)b]$.

Our simple idea is to take such a fractional packing of $L(x)$ and modify it to a fractional packing of $L(\bar{x})$. Informally, we replace the rectangles of $L(x)$ by the rectangles of $L(\bar{x})$. Our goal is to show that this modification can be completed with some small increase in the height of the packing.

Assume that we are given a fractional packing of $L(x)$ in the area $[0,1] \times [0,(1+2\varepsilon)b]$. As we noted in Section 4.2.3, in this case for all configurations C_j ($j = 1,\ldots,n$) we can find some values $y_j \geq 0$ such that

$$\begin{aligned} \textstyle\sum_{j=1}^{\#C} y_j &\leq b(1+2\varepsilon), \\ \textstyle\sum_{j:R_i \in C_j} y_j &\geq b_i \cdot x_i, \qquad \text{for all } i = 1,\ldots,n. \end{aligned}$$

Let c_j be the width of all wide rectangles in configuration C_j. Then, we can construct a layered fractional packing of $L(x)$ in the area $[0,1] \times [0,(1+2\varepsilon)b]$ as follows. We first define the values $\ell_0 = 0$ and $\ell_j = \ell_{j-1} + y_j$ ($j = 1,\ldots,\#C$). The jth layer is defined as the two rectangles $Q_j = [0,c_j] \times [\ell_{j-1},\ell_j]$ and $Q'_j = [c_j,1] \times [\ell_{j-1},\ell_j]$, see Fig 4.4.

For each rectangle $R_i = (a_i,b_i)$ from the list L, we consider all the configurations C_j that include R_i. The sum of y_j over all such configurations, $\sum_{j:R_i \in C_j} y_j$, is at least $x_i \cdot b_i$. So, we select these configurations C_j one by one in a greedy manner, and place a rectangle (a_i,y_j) in the jth layer defined by C_j. If R_i is wide, we place it into Q_j. Otherwise, we place it into Q'_j. At the end of this procedure, we obtain a fractional packing of the rectangles in $L(x)$ where all Q_j are filled with the wide rectangles, and all Q'_j are filled with the narrow rectangles, see Fig. 4.5. The height of the packing is at most

$$\sum_{j=1}^{\#C} y_j \leq (1+2\varepsilon)b.$$

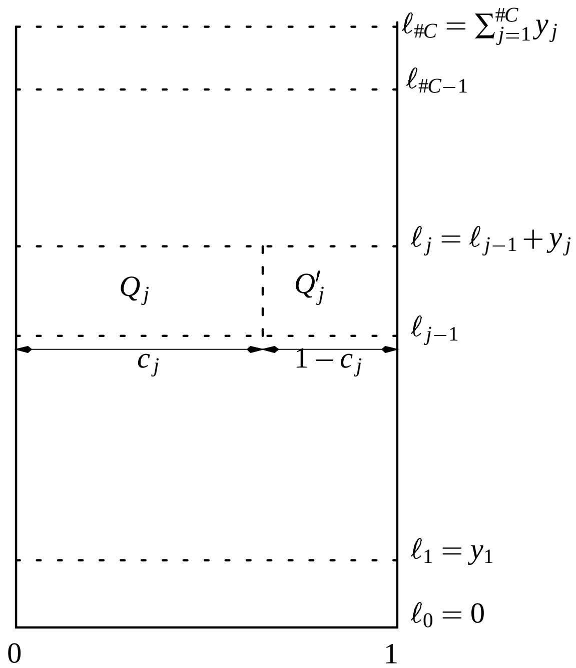


Figure 4.4: The jth layer

Recall the rounding procedure in Section 4.3.2. Let $\varepsilon' = \varepsilon/(2+\varepsilon)$ and $m = \lceil 1/(\varepsilon')^2 \rceil$. There are $m-1$ threshold rectangles and m groups, see Fig 4.2. Let a_{i_k} be the width of the kth threshold rectangle ($k = 1, \dots, m-1$). Let $L^{(k)}_{wide}(x)$ and $L^{(k)}_{wide}(\bar{x})$ denote the kth groups with respect to $L(x)$ and $L(\bar{x})$.

Due to the input, any rectangle has side lengths in $(0,1]$. One can see that the height values of any two consecutive groups $L^{(k)}_{wide}(x)$ and $L^{(k+1)}_{wide}(x)$ are roughly $(\varepsilon')^2 H$. They differ by at most 2, the maximum height of two rectangles. By the knapsack formulations and the rounding procedure, the height of $L^{(k+1)}_{wide}(x)$ can differ from the height of $L^{(k+1)}_{wide}(\bar{x})$ by at most 1, the maximum height of one rectangle.

There are two nice facts. The width of any rectangle in $L^{(k)}_{wide}(x)$ is at least a_{i_k}. The width of any rectangle in $L^{(k+1)}_{wide}(\bar{x})$ is most a_{i_k}. From the above observation, the height values of $L^{(k)}_{wide}(x)$ and $L^{(k+1)}_{wide}(\bar{x})$ are roughly the same, differing by at most 3. So, if $L^{(k+1)}_{wide}(\bar{x})$ fractionally replaces $L^{(k)}_{wide}(x)$ in the packing, then the height of

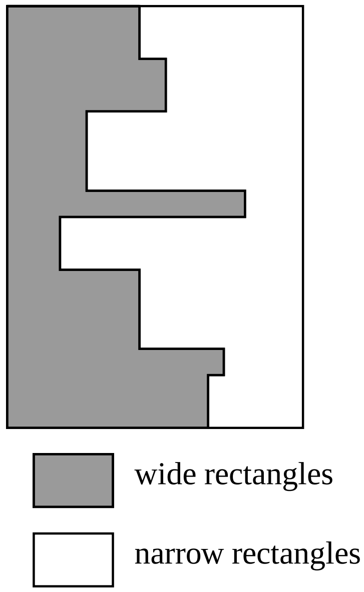


Figure 4.5: A layered packing

the packing increases by a small value.

We take the above constructed fractional packing of $L(x)$, and go from one group to another. We replace all the wide rectangles in regions $Q_1, Q_2, \ldots, Q_{\#C}$ as follows. The rectangles in the first group $L^{(1)}_{wide}(x)$ are the widest ones. We simply delete them from the packing. This creates a set of gaps. Each gap has width at least a_{k_1}. Since any rectangle of $L^{(2)}_{wide}(\bar{x})$ has width at most a_{k_1}, it can be fractionally packed inside these gaps. So, we simply put all the rectangles of $L^{(2)}_{wide}(\bar{x})$ in a greedy manner, filling the gaps. If some rectangles are left, we pack them one by one above all the rectangles, i.e. on the top of the packing. Similarly, we create some gaps by deleting the rectangles of $L^{(k)}_{wide}(x)$, and then fractionally pack the rectangles of $L^{(k+1)}_{wide}(\bar{x})$. At the end, we take all the rectangles which are still left, including the rectangles of the first integral group $L^{(1)}_{wide}(\bar{x})$, and pack them one by one on the top of the packing.

There are at most m groups. In each of the groups, the total height of the rectangles which go on the top of the packing is at most 3. The height of the first group $L^{(1)}_{wide}(\bar{x})$ is at most $(\frac{1}{\varepsilon})^2 H + 2$. (Here, 1 for one threshold rectangle and 1 for rounding.) Hence, the height of the packing increases by at most

$$\Delta_{wide} = 3m + (\varepsilon')^2 H + 2.$$

Recall that the height of all the wide rectangles in $L_{wide}(x)$ is given by

$$H = \sum_{R_i \in L_{wide}} x_i \cdot b_i.$$

Since all the wide rectangles in L_{large} are lager than ε', the total size of the wide rectangles in $L_{wide}(x)$ is at least $\varepsilon' H$. Since the rectangle of $L(x)$ can be fractionally packed in the area $[0,1] \times [0,(1+2\varepsilon)b]$, this total size cannot be larger than $(1+2\varepsilon)b$. Hence, $\varepsilon' H \leq (1+2\varepsilon)b$, and a possible increase can be bounded by

$$\Delta_{wide} = 3m + 2 + (\varepsilon')^2 H \leq 3m + 2 + \varepsilon'(1+2\varepsilon)b.$$

Let $L_{narrow}(x)$ and $L_{narrow}(\bar{x})$ denote the lists of narrow rectangles with respect to $L(x)$ and $L(\bar{x})$. There is one nice fact. By the knapsack formulation, the size of $L_{narrow}(\bar{x})$ differ from the size of $L_{narrow}(x)$ by at most 1, the maximum size of one rectangle. So, if $L_{narrow}(\bar{x})$ fractionally replaces $L_{narrow}(x)$, the height of the packing increases by a small value.

We take the current packing. Then, in all $Q'_1, Q'_2, \ldots, Q'_{\#C}$ we delete the rectangles of $L_{narrow}(x)$. Next, we fill all these empty rectangles one by one with the rectangles from $L_{narrow}(\bar{x})$ as follows. We form a queue which consists of the rectangles from $L_{narrow}(\bar{x})$, and it is always sorted by non-increasing of heights. In each rectangle $Q'_j = [c_j, 1] \times [\ell_{j-1}, \ell_j]$ $(j = 1, \ldots, \#C)$ we organize a fractional packing by using the modified Next-Fit-Decreasing-Height (NFDH); The rectangles are packed so as to form a sequence of sublevels. Each sublevel consists of (probably fractional) rectangles of the same height. The first sublevel is defined at ℓ_{j-1}, i.e. just the bottom line of Q'_j. Then, each subsequent level is defined by a horizontal cut line drawn through the top of the previous sublevel. For the current sublevel, starting from c_j rectangles are packed in a left-justified greedy manner, until there is sufficient space to the right boundary at point 1 to place the next rectangle. If the first rectangle on the sublevel goes above ℓ_j, i.e. the top of Q'_j, then a horizontal cut line is drown at point ℓ_j. Otherwise, it is drown on the top of the last rectangle on this sublevel. At that moment, the fractions (if any) above the cut line return to the queue and get sorted, the current sublevel is discontinued, the next sublevel is defined, and packing proceeds on the new sublevel until either the top of Q'_j is not reached or the queue is not empty. For an illustration see Fig. 4.6.

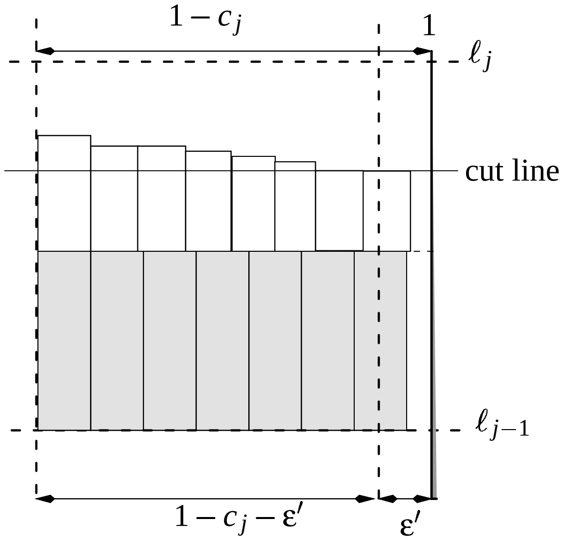


Figure 4.6: Packing of narrow rectangles

Assume that the above procedure completes with a non-empty queue, i.e. there are some unpacked narrow rectangles. Recall that the width of any narrow rectangle is at most ε'. Hence, in all $Q'_1, Q'_2, \ldots, Q'_{\#C}$ the uncovered area is at most ε' times the height of the packing, i.e. bounded by $\varepsilon'(1+2\varepsilon)b$. Also, recall that the narrow rectangles of $L_{narrow}(x)$ can be fractionally packed in the area of all $Q'_1, Q'_2, \ldots, Q'_{\#C}$, and that the side of $L_{narrow}(\bar{x})$ differs from the size of $L_{narrow}(\bar{x})$ by at most 1. Thus, we can bound the size of the unpacked narrow rectangles by $\varepsilon'(1+2\varepsilon)b+1$.

Next, we can simply pack all the unpacked narrow rectangles from $L_{narrow}(\bar{x})$ (if any) above all the rectangles, i.e. on the top of the packing. In order to organize a packing, we again use the modified NFDH, which now works with the strip of unit width and unbounded height. Let Δ_{narrow} be the height of that additional packing.

Using the ideas from the above paragraph, we can obtain the following bound on the area covered by the narrow rectangles

$$\Delta_{narrow}(1-\varepsilon') \leq \varepsilon'(1+2\varepsilon)b+1.$$

It follows that

$$\Delta_{narrow} \leq [\varepsilon'(1+2\varepsilon)b+1]/(1-\varepsilon').$$

In overall, summing for wide and narrow rectangles, we can produce a fractional packing of the rectangles from $L(\bar{x})$ in the strip $[0,1]\times[0,+\infty)$. The height to which the strip is filled can be bounded by

$$\begin{aligned} h &= (1+2\varepsilon)b+\Delta_{wide}+\Delta_{narrow} \\ &\leq (1+2\varepsilon)b+[3m+\varepsilon'(1+2\varepsilon)b+2]+[\varepsilon'(1+2\varepsilon)b+1]/(1-\varepsilon'). \end{aligned}$$

Now we can use the results of Theorem 4.2.3. After applying the KR-algorithm, we get a packing of the rectangles from $L(\bar{x})$ in the strip $[0,1]\times[1,+\infty)$ such that the height to which the strip is filled is bounded by

$$h' \leq h(1+1/(m\varepsilon'))/(1-\varepsilon')+4m+1.$$

Recall that $b \geq 1/\varepsilon^3$, $m = \lceil (1/\varepsilon')^2 \rceil$, $\varepsilon' = \varepsilon/(2+\varepsilon)$, $\varepsilon \in (0,1/4]$ and $1/\varepsilon$ is integral. Hence, we have that $\varepsilon'/(1-\varepsilon') = \varepsilon/2$ and $1/(1-\varepsilon') = (2+\varepsilon)/2$. Thus, we can estimate

$$\begin{aligned} h &\leq (1+2\varepsilon)b+[3m+2+\varepsilon'(1+2\varepsilon)b]+[\varepsilon'(1+2\varepsilon)b+1]/(1-\varepsilon') \\ &\leq (1+2\varepsilon)b+\left[3m+2+\frac{\varepsilon}{2+\varepsilon}(1+2\varepsilon)b\right]+\left[\frac{\varepsilon}{2}(1+2\varepsilon)b+\frac{2+\varepsilon}{2}\right] \\ &\leq (1+2\varepsilon)b+3m+\left[\frac{\varepsilon}{2}(1+2\varepsilon)b+\frac{\varepsilon}{2}(1+2\varepsilon)b\right]+[3+\varepsilon/2] \quad \text{since } \varepsilon>0 \\ &\leq (1+2\varepsilon+\varepsilon+2\varepsilon^2)b+3m+3+\varepsilon/2 \\ &\leq (1+3\varepsilon+2\varepsilon^2)b+3m+3+\varepsilon/2. \end{aligned}$$

Notice that $(1+1/(m\varepsilon'))/(1-\varepsilon') \leq 1+\varepsilon$. Hence, the height of the packing is bounded by

$$\begin{aligned}
h' &\leq [(1+3\varepsilon+2\varepsilon^2)b+3m+3+\varepsilon/2](1+\varepsilon)+4m+1\\
&\leq (1+3\varepsilon+2\varepsilon^2+\varepsilon+3\varepsilon^2+2\varepsilon^3)b+3m(1+\varepsilon)+3(1+\varepsilon)+(\varepsilon/2)(1+\varepsilon)+4m+1\\
&\leq (1+4\varepsilon+5\varepsilon^2+2\varepsilon^3)b+m(7+3\varepsilon)+4+(7/2)\varepsilon+(1/2)\varepsilon^2\\
&\leq (1+4\varepsilon+5\varepsilon^2+2\varepsilon^3)b+m(7+3\varepsilon)+4+4\varepsilon \;\text{ since } \varepsilon\in(0,1].
\end{aligned}$$

Recall that

$$m=\lceil 1/(\varepsilon')^2\rceil=\lceil (2+\varepsilon)^2/\varepsilon^2\rceil\leq (2+\varepsilon)^2/\varepsilon^2+1=\frac{4+4\varepsilon+2\varepsilon^2}{\varepsilon^2}.$$

So, we finally have that

$$\begin{aligned}
h' &\leq (1+4\varepsilon+5\varepsilon^2+2\varepsilon^3)b+\frac{(4+4\varepsilon+2\varepsilon^2)(7+3\varepsilon)}{\varepsilon^2}+\frac{4\varepsilon^2+4\varepsilon^3}{\varepsilon^2}\\
&= (1+4\varepsilon+5\varepsilon^2+2\varepsilon^3)b+\frac{28+40\varepsilon+30\varepsilon^2+10\varepsilon^3}{\varepsilon^2} \qquad (4.23)\\
&\leq (1+4\varepsilon+5\varepsilon^2+2\varepsilon^3)b+\frac{28+40\varepsilon+40\varepsilon^2}{\varepsilon^2} \;\text{ since } \varepsilon\in(0,1].
\end{aligned}$$

This completes the proof of Lemma 4.3.6.

Proof of Corollary 4.3.7: By Corollary 4.3.3 the weight of $L(x)$ is at least $(1-3\varepsilon)$OPT. By Lemma 4.3.6, the weight of $L(\bar{x})$ is at least the weight of $L(x)$. So, the weight of $L(\bar{x})$ is at least $(1-\delta_1\cdot\varepsilon)$OPT, where $\delta_1=3$.

Let $b\geq 1/\varepsilon^4$, $\varepsilon\in(0,1/\beta]$ and $\beta\geq 4$. Then, form (4.23) we can obtain that

$$\begin{aligned}
h' &\leq (1+4\varepsilon+5\varepsilon^2+2\varepsilon^3)b+(28+40\varepsilon+40\varepsilon^2)\varepsilon^2 b \;\text{ since } \varepsilon^2 b\geq 1/\varepsilon^2\\
&\leq (1+4\varepsilon+33\varepsilon^2+42\varepsilon^3+40\varepsilon^4)b\\
&\leq (1+[4+(33/\beta)+(42/\beta^2)+(40/\beta^3)]\varepsilon)b \;\text{ since } \varepsilon\leq 1/\beta\\
&\leq (1+[4+(33/\beta)+(82/\beta^2)]\varepsilon)b \;\text{ since } \beta\geq 4.
\end{aligned}$$

Let $b\geq 1/\varepsilon^3$, $\varepsilon\in(0,1/\beta]$ and $\beta\geq 4$. Then, in a similar way we can obtain that

$$\begin{aligned}
h' &\leq (1+4\varepsilon+5\varepsilon^2+2\varepsilon^3)b+(28+40\varepsilon+40\varepsilon^2)\varepsilon b \;\text{ since } \varepsilon b\geq 1/\varepsilon^2\\
&\leq (1+32\varepsilon+45\varepsilon^2+42\varepsilon^3)b\\
&\leq (1+[32+(45/\beta)+(42/\beta^2)\varepsilon])b.
\end{aligned}$$

This completes the proof.

4.4.3 The overall analysis: The proof of Theorem 4.1.1

The correctness of our algorithm follows from Lemma 4.3.6 and Lemma 4.3.8. We first apply the KR-algorithm, and then use the shifting technique. Hence, the algorithm always outputs a packing in the area $[0,1] \times [0,b]$ of the dedicated rectangle $R = (1,b)$.

Regarding the running time of our algorithm we can estimate each of the three steps. In Step 1, as it is described in Section 4.3.1, we solve a sequence of n/ε^2 resource-sharing problems. We find and round $\bar{\varepsilon}$-approximate solutions, where $\bar{\varepsilon} = \varepsilon^2/n$. So, we are required to obtain a resource sharing algorithm $RSA(\bar{\varepsilon})$ which finds any $\bar{\varepsilon}$-approximate solution in time polynomial in n and $1/\varepsilon$. By Theorem 4.2.1, there exists some constant $q \in \mathbb{N}$ such that it is enough to present a block solver algorithm $BSA(p,\bar{t})$ for any $\bar{t} = \Theta(\bar{\varepsilon})$ and any price vector p whose positive coordinates $p_i = \Omega([\bar{\varepsilon}/n]^q)$ $(i = 0,\ldots,n)$. Let $T = (n/\bar{\varepsilon})^q$. Then, by Lemma 4.4.4, we can obtain $BSA(p,\bar{t})$ whose running time is bounded by $O(n^2 \cdot T) + KS(n,\bar{t})$. Recall that $KS(n,\bar{t})$ is the running time of an FPTAS for the knapsack problem with accuracy $\bar{t}$, that is polynomial in n and $1/\varepsilon$. Hence, a required $RSA(\bar{\varepsilon})$ can be obtained by Theorem 4.2.1. In Steps 2, we partition the rectangles into wide and narrow, solve $O(1/\varepsilon^2)$ fractional knapsacks, and perform rounding. These require at most $O((1/\varepsilon^2)n\log n)$ time. Next, we apply the KR-algorithm. By Theorem 4.2.3, its running time $KR(n,\varepsilon)$ is polynomial in n and $1/\varepsilon$. In Step 3, we finally apply the shifting technique. By Lemma 4.3.8, this requires at most $O(n+1/\varepsilon)$ time. Summing up, the running time of our algorithm is polynomial in n and $1/\varepsilon$.

It remains to show that weight of the output packing is close to the optimum. In step 1, as it is stated in Corollary 4.3.3, we find a fractional list of $L(x)$ whose weight is at least $(1-3\varepsilon)\mathrm{OPT}$. In Steps 2, by Lemma 4.3.6, we round $L(x)$ to an integral list $L(\bar{x})$. We use simple optimal fractional solutions. Hence, the weight of $L(\bar{x})$ remains at least $(1-3\varepsilon)\mathrm{OPT}$. Finally, by Lemma 4.3.8, in Step 3 the shifting technique outputs a packing in the area $[0,1] \times [0,b]$ whose weight is at least $(1-O(\varepsilon))\mathrm{OPT}$. This completes the proof of Theorem 4.1.1.

4.5 CONCLUDING REMARKS

In this chapter we present an algorithm, which significantly improves the running time of the algorithm in Chapter 3. Namely, we present an FPTAS with large resources for the general version of the storage packing problem of packing weighted rectangles into a larger rectangle. Given a set of rectangles, our algorithm finds a subset of rectangles and it's packing into a dedicated rectangle with total weight at least $(1-\varepsilon)\mathrm{OPT}$. The running time is polynomial in the number of rectangles and, contrasting to the previous result, is also polynomial in $1/\varepsilon$.

CHAPTER 5

ON PACKING RECTANGLES WITH ROTATIONS BY 90 DEGREES

5.1 INTRODUCTION

In this chapter we address one of the classical NP-hard problems: strip packing. In this problem a set of rectangles is packed into a vertical strip of unit width so that the height to which the strip is filled is minimized.

Indeed, a significant number of known theoretical results in packing are devoted to this problem. Of course, the strip packing problem is strongly NP-hard since it includes the bin packing problem as a special case.

On the other hand, there are still a few important theoretical questions that remain open. Currently, the most interesting question is to finalize all natural extensions of the problem for which the known approximation schemes can be generalized. Here we give a positive answer for the strip packing problem in the case when rotations of the rectangles are allowed. Besides that, we develop new techniques which allow us to use the known algorithm for the strip-packing (without rotations) in [57]. So, this closes the gap between the classical statement of the problem and it's extension.

The strip packing problem with rotations by 90 degrees is stated as follows. In the input we are given a set of n rectangles, $R = \{(a_1, b_2), (a_2, b_2), \ldots, (a_n, b_n)\}$, with side lengths a_j, b_j $(j = 1, \ldots, n)$ in the interval $[0, 1]$. Rotations of 90 degrees are allowed. That is, for each rectangle (a_j, b_j) $(j = 1, \ldots, n)$ there is a binary variable

$x_j \in \{0,1\}$: if $x_j = 1$, we allocate (a_j, b_j) to a *non-rotated* rectangular frame, $R_j(x_j) = a_j \times b_j \cdot x_j$, whose width is a_j and height is $b_j \cdot x_j$; otherwise $x_j = 0$, and we allocate (a_j, b_j) to a *rotated* rectangular frame, $R'_j(x_j) = b_j \times a_j \cdot (1 - x_j)$, whose width is b_j and height is $a_j \cdot (1 - x_j)$, respectively, see Fig. 5.1.

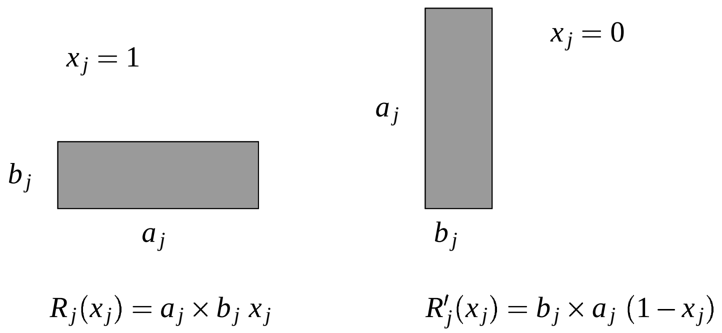


Figure 5.1: Rotated and non-rotated frames $R'_j(x_j)$ and $R_j(x_j)$

The area of the two frames, $a_j \cdot (x_j \cdot b_j) + b_j \cdot (1 - x_j) \cdot a_j$, is exactly $a_j \cdot b_j$, that is the area of rectangle (a_j, b_j). Then, a set of (rotated and non-rotated) frames, $R(x)$, defines an allocation for R. A strip-packing of $R(x)$ is a positioning of the frames of $R(x)$ within the vertical strip of unit width, $[0,1] \times [0, \infty)$, so that no two frames have intersecting interiors. The height of a strip-packing is defined as the height to which the strip is filled by the frames. In the strip-packing with rotations by 90 degrees it is required to find an allocation, $R(x)$, and a strip-packing of the frames of $R(x)$ so as the packing height is minimized.

Theorem 5.1.1. *There is an algorithm which given a set of n rectangles, R, with side lengths at most* 1*, and a positive accuracy,* $\varepsilon > 0$*, finds an allocation of R to a set of frames,* $R(x)$*, and a strip-packing of the frames of* $R(x)$ *whose height is at most*

$$(1 + \varepsilon)\mathrm{OPT}(R) + O(1/\varepsilon^2),$$

where $\mathrm{OPT}(R)$ *is the height of the optimal strip-packing of R with rotations by* 90 *degrees. The running time of the algorithm is polynomial in n and* $1/\varepsilon$*.*

In other words, we present an asymptotic fully polynomial time approximation scheme (AFPTAS) (an equivalent result has been independently obtained by Jansen and van Stee in [49]). The exitance of such a scheme has been an open theoretical problem [19].

Organization of the Chapter. The rest of the chapter is organized as follows. In Section 5.2 we describe our algorithm. Section 5.3 consists of the analysis of the algorithm. In the final section we give some concluding remarks.

5.2 AN ALGORITHM FOR STRIP PACKING WITH ROTATIONS

5.2.1 Separating of Rectangles: Sets L and S

Let $\varepsilon' = \varepsilon/(2+\varepsilon)$. We say that a rectangle (a_j, b_j) is small if at least one of its side lengths, a_j or b_j, is smaller than ε', and large otherwise. We partition R into a set of large rectangles, L, and a set of small rectangles, S, respectively. So, either side length of each large rectangle from L is at least ε', and one side of each small rectangle in S is less than ε'. For simplicity, frames are also called small and large.

5.2.2 Fractional Strip-Packing: The algorithm STRIP

Let $L(x)$ be an (possibly fractional) allocation of the large rectangles in L to frames. A fractional strip-packing of $L(x)$ is a strip-packing of any set of frames obtained from $L(x)$ by cutting any frame into a set frames of the same width: each large frame $R_j(x_j)$ ($R'_j(x_j)$) is replaced by a sequence of frames $R_j(z_{j_1})$,$R_j(z_{j_2})$, $\ldots$, $R_j(z_{j_q})$ (respectively $R'_j(z_{j_1}), R'_j(z_{j_2}), \ldots, R'_j(z_{j_q})$), where $\sum_{\ell=1}^{q} z_{j_\ell} = x_j$. We use the following result which defines a relationship between fractional packing and integral packing.

Theorem 5.2.1 (Kenyon & Rémila [57]). *Let $\varepsilon' = \varepsilon/(2+\varepsilon)$ and $m = \lceil (1/\varepsilon')^2 \rceil$. Let $L(x)$ be an allocation of L to large frames. Let $S(x)$ be an allocation of S*

to small frames such that all frame widths are less than ε'*. Then, there is an algorithm,* STRIP*, which given an accuracy,* $\varepsilon \in (0,1]$*, and a set of frames,* $R(x) = [L \cup S](x)$*, finds a positioning of the frames in* $R(x)$ *within the vertical strip* $[0,1] \times [0,\infty)$ *of unit width such that no two frames have intersecting interiors and the height to which the strip is filled is bounded by*

$$\begin{aligned} STRIP(R(x)) \leq \max\{ & \, lin(L(x))(1+1/(m\varepsilon')) + 2m + 1, \\ & area(R)(1+1/(m\varepsilon'))/(1-\varepsilon') + 4m + 1 \, \}, \end{aligned}$$

where $area(R)$ *is the total area of the rectangles in* R*, and* $lin(L(x))$ *is the height of the optimal fractional strip packing of the large frames in* $L(x)$*. The running time of* STRIP *is polynomial in* n *and* $1/\varepsilon$*.*

Remark. Notice that the theorem deals with an allocation of rectangles to frames. As one can see, it is not that hard to allocate the small rectangles in S. The main difficulty comes from the large rectangles in L. In order to cope with that we introduce an LP formulation in the next section.

5.2.3 LP formulation

Let R_j denote a non-rotated frame, $a_j \times b_j$, and R'_j denote a rotated frame, $b_j \times a_j$. Now we can define configurations as follows. A configuration, C, is a set of rotated and non-rotated large frames such that there is no large rectangle (a_j, b_j) whose both R_j, R'_j, non-rotated and rotated frames, belong to C. The total width of C, $\sum_{j:R_j \in C} a_j + \sum_{j:R'_j \in C} b_j$, cannot exceed the width of the strip, 1.

Informally, every configuration defines a set of large frames that can be packed on the same horizontal level of the strip packing. Without loss of generality, we assume that the configurations are arbitrary ordered. Let N be the total number of configurations, and C_i be configuration i. (Notice that $N = O(1)$.) For each configuration C_i, let $W(C_i)$ be the total width of C_i.

Now, we are ready to formulate a relaxation of the problem as the following LP:

$$\begin{array}{lrcl} \text{minimize} & h & & \\ \text{subject to} & \sum_{i=1}^{N} y_i & = & h \\ & \sum_{i:R_j \in C_i} y_i & \geq & x_j \cdot b_j, \text{ for all } j \in L, \\ & \sum_{i:R'_j \in C_i} y_i & \geq & (1-x_j) \cdot a_j, \text{ for all } j \in L, \\ & x_j & \in & [0,1], \text{ for all } j \in L, \\ & y_i & \geq & 0, \text{ for all } i = 1, \ldots, N. \end{array} \tag{5.1}$$

Here, x_j is a fraction of rectangle (a_j, b_j), and y_i is the height of configuration C_i. In the constrains, each fractional non-rotated frame, $R_j(x_j) = a_j \times b_j \cdot x_j$, is fractionally packed within configurations C_i that include R_j, and each fractional rotated frame, $R'_j(x_j) = b_j \times a_j \cdot (1 - x_j)$, is fractionally packed within configurations C_i that include R'_j. In the objective function, the total height over all configurations, h, is minimized. We can provide the following result.

Lemma 5.2.2 (Jansen [46, 48]). *The LP can be solved in time polynomial in n and $1/\varepsilon$. The optimal objective function value of the LP, $h = \sum_{i=1}^{N} y_i$, is upper bounded by $lin(L)$, the height of the optimal fractional strip packing of the large rectangles in L.*

Proof Sketch. The LP consists of $O(N)$ variables and $O(n)$ constrains. The number of configurations depends on $1/\varepsilon$. So, LP can be solved in a required time [46, 48]. In the LP, we relax the problem in two ways. First, each decision variable is relaxed to $x_j \in [0,1]$. Second, $R_j(x_j)$ and $R'_j(x_j)$ are two fractions of (a_j, b_j), and either of them can be cut by horizontal lines in a strip-packing. So, an optimal fractional strip-packing of the small rectangles in L gives a feasible solution of the LP. Hence, $lin(L)$ is an upper bound on h. □

5.2.4 Rounding

Here we round our (possibly fractional) LP allocation, $L(x)$. For each large rectangle (a_j, b_j) in L, there are two fractional frames, $R_j(x_j)$ and $R'_j(x_j)$. So, we order all the frames in $L(x)$ by non-increasing widths. Next, we select the frames one by one in this order and stack them left justified, see Fig. 5.2. Let H be the height of the stack.

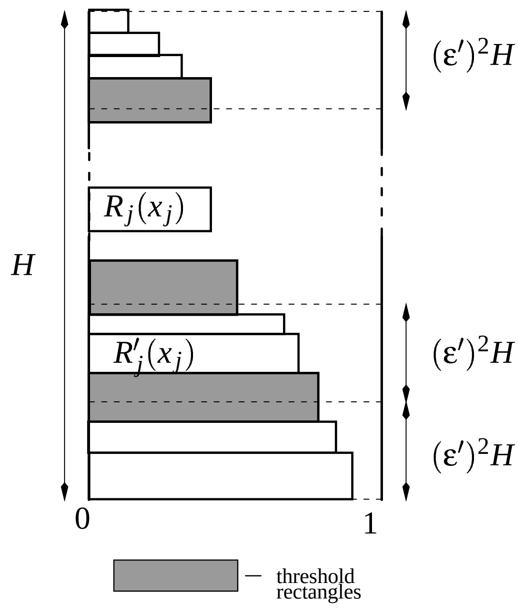


Figure 5.2: A stack with threshold frames

Let $m = \lceil 1/(\varepsilon')^2 \rceil$. Next, we define $m-1$ threshold frames as follows. We draw $m-1$ horizontal lines at points $y = k \cdot [(\varepsilon')^2 \cdot H]$, for k between 1 and $m-1$, see Fig. 5.2. The kth threshold frame is defined as a fractional frame whose interior or lower boundary is intersected by the kth line, respectively. These $m-1$ threshold frames separate the set of all large frames into m non-intersecting groups, $L^1(x), L^2(x), \ldots, L^m(x)$. Each threshold frame has the least width in its group.

The width of $L^1(x)$ is at most unit, 1. The width of $L^k(x)$ is at most the width of the $(k-1)$th threshold frame. Let $g(j)$ and $g'(j)$ from $\{1, 2, \ldots, m\}$ be defined such that a fractional non-rotated frame $R_j(x_j)$ belongs to a group $L^{g(j)}(x)$ and a fractional rotated frame $R'_j(x_j)$ belongs to a group $L^{g'(j)}(x)$, respectively. (Notice that that two groups may not match.) Then, the height of the kth group, $H(L^k(x))$, is defined as follows

$$H(L^k(x)) = \left(\sum_{j:g(j)=k} x_j \cdot b_j\right) + \left(\sum_{j:g'(j)=k} (1-x_j)\cdot a_j\right).$$

Now we round the values of x such that there is no change in these m height values.

Lemma 5.2.3. *An optimal LP allocation, $L(x)$, can be rounded to an allocation, $L(\bar{x})$, such that there are at most m large rectangles with $\bar{x}_j \in (0,1)$, and all other large rectangles with $\bar{x}_j \in \{0,1\}$. Furthermore, the width of each rounded group $L^k(\bar{x})$ is at most the width of group $L^{k-1}(x)$, whereas the height of $L^k(\bar{x})$ is at most the height of $L^{k-1}(x)$ plus 2, the maximum height of two frames. The required rounding time is polynomial in the number of rectangles, n, and the number of groups, m.*

Proof Sketch. We have a system of m linear equations with $O(n)$ variables. We also have constraints, $x_j \in [0,1]$. Using polyhedral theory it can be shown that a rounded solution, $\bar{x}$, can be found in time polynomial in n and m, see Section 5.3.1.

Due to the input, any rectangle has side lengths in $(0,1]$. One can see that the height values of any two consecutive groups $L^k(x)$ and $L^{k-1}(x)$ are roughly $(\varepsilon')^2 H$. They can differ by at most 2, the maximum side length of two frames. By the rounding procedure, the heights of groups remain the same. Thus, the width of $L^k(\bar{x})$ is at most the width of $L^{k-1}(x)$, and the height of $L^k(\bar{x})$ is at most the height of $L^{k-1}(x)$ plus 2. □

Furthermore, we can prove that our rounding leads to a small increase in height.

Lemma 5.2.4. *Let $L(\bar{x})$ be a rounded allocation. Then, the height of the optimal fractional strip-packing of the frames in $L(\bar{x})$, $lin(L(\bar{x}))$, can be bounded as follows*

$$lin(L(\bar{x})) \leq (1+\varepsilon)lin(L) + 3m,$$

where $lin(L)$ is the optimal fractional strip packing of the rectangles in L.

Proof. See Section 5.3.2. □

5.2.5 *An Allocation of Large Rectangles to Frames: A Trash Set*

We define a set T of all rectangles in L with $x_j \in (0,1)$. So, T is further called the *trash set* of L. Now we are ready to define an allocation of the large rectangles in L to frames. We first use a rounded LP allocation, $L(\bar{x})$. Let T be a *trash* set of large rectangles (a_j, b_j) in L with $x_j \in (0,1)$. By Lemma 5.2.3, there are at most m rectangles in T, and we will pack them at the end of the algorithm. For the other rectangles in $L' = L \setminus T$ we define an integral allocation, $L'(\bar{x})$, as $[L \setminus T](\bar{x})$, that is, for each (a_j, b_j) in L' we take $R_j(\bar{x}_j)$ if $x_j = 1$ and $R'_j(\bar{x}_j)$ if $x_j = 0$. Finally, we arbitrary define the frames for the trash rectangles in T and add them on the top of the packing.

5.2.6 *An Allocation of Small Rectangles to Frames*

We handle the small rectangles from S in a very easy manner. We allocate the rectangles from S to small frames such that all frame width are less than ε'. This gives us an integral allocation $S(\bar{x})$.

5.2.7 *The overall algorithm*

Here we describe an outline of our algorithm.
ALGORITHM A_ε:
Input: A set of rectangles, R, and an accuracy, $\varepsilon > 0$.
Output: A strip-packing of R with rotations by 90 degrees.

1. **Partition.** Let $\varepsilon' = \varepsilon/(2+\varepsilon)$. Perform partition $R = L \setminus S$ to set aside rectangles with at least one side smaller than ε'.

2. **LP & Rounding.** Solve the LP. Find a (fractional) LP allocation, $L(x)$. Find m threshold frames. Perform rounding of $L(x)$ to $L(\bar{x})$.

3. **Frames.** Define a trash set, T. Let $L' = L \setminus T$. Define an integral allocation $L'(\bar{x})$ as $[L \setminus T](\bar{x})$. Find an integral allocation, $S(\bar{x})$, for the small rectangles in S to have widths at most ε'. Let $R' = L' \cup S$ and $R'(\bar{x}) = L'(\bar{x}) \cup S(\bar{x})$. Then, $R'(\bar{x})$ is an integral allocation of R'.

4. **Packing.** Use the algorithm STRIP on an integral allocation $R'(\bar{x})$. This gives a strip-packing of $R'(\bar{x})$.

5. **Trash.** Add the trash rectangles of T to the packing.

Lemma 5.2.5. *The height of the packing output by* A_ε *is at most* $(1+2\varepsilon)\mathrm{OPT}(R) + 81/\varepsilon^2 + 1$, *where* $\mathrm{OPT}(R)$ *is the height of the optimal strip-packing of* R *with rotations by* 90 *degrees.*

Proof. See Section 5.3.3 □

5.3 THE ANALYSIS OF STRIP-PACKING WITH ROTATIONS

5.3.1 Proof of Lemma 5.2.3

Here, we just briefly sketch a required rounding technique for the following linear system (LS):

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} \cdot x_j & = & b_i & \text{for } i = 1, \ldots, m, \\ x_j & \in & [0,1], \text{ for} & j = 1, \ldots, n. \end{array}$$

We can rewrite it as

$$\begin{array}{rcl} Ax = \sum_{j=1}^{n} A_j \cdot x_j & = & b, \\ x_j & \in & [0,1], \text{ for } j = 1, \ldots, n. \end{array}$$

where $A_j = (a_{1j}, a_{2j}, \ldots, a_{mj})^T$, $x = (x_1, x_2, \ldots, x_n)$, and $b = (b_1, b_2, \ldots, b_m)^T$.

We modify a solution x to a new solution $\bar{x}$ as follows. Consider a solution x. We can always update LS in two cases: (1) there exists $x_k = 1$, (2) there exists $x_k = 0$.

In the first case we remove x_k from the LS and define b to be equal $b - A_k \cdot x_k$. In the second case, we just remove x_k. Informally, this eliminates integral x_k from x.

Assume now that are $m+1$ fractions $x_k \in (0,1)$. Then, we can select the corresponding columns and form an induced matrix, A'. Clearly, A' is a system of linearly dependent vectors, and one can find a non-zero vector y in the null space, $A'y = 0$.

Let $\delta \in \mathbb{R}$ and $\bar{x} = x + \delta y$. (If the dimension of y is smaller than the dimension of x, we augment it by adding an appropriate number of zero entries and denote it as y'.) Then, $Ay' = A'y = 0$ and

$$A\bar{x} \;=\; Ax \;+\; \delta Ay' \;=\; b \;+\; \delta A'y \;=\; b.$$

Since all $x_k \in (0,1)$, there exists δ (if δ tends to 0) such that all $x_k + \delta y_k \in (0,1)$. Thus, one can increase or decrease the value of δ until at least one $\bar{x}_k$ gets either to 0 or 1.

We iteratively repeat the above rounding and removing procedures until there are at most m fractions left. (Here A' can become a system of linearly independent vectors.) At the end of this iterative process there are at most m fractions, $\bar{x}_j \in (0,1)$, all other $\bar{x}_j \in \{0,1\}$. The total number of iterations is at most $O(n)$. Each iteration can be completed in time polynomial in $O(m)$.

5.3.2 *Proof of Lemma 5.2.4*

Our simple idea is to take a fractional packing of $L(x)$ and modify it to a fractional packing of $L(\bar{x})$. Informally, we replace the frames of $L(x)$ by the frames of $L(\bar{x})$. The goal is to show that this modification can be completed with some small increase in the height of the packing.

Recall that x is an LP solution, i.e.

$$\begin{aligned}
\textstyle\sum_{i=1}^{N} y_i &= h \\
\textstyle\sum_{i:R_j \in C_i} y_i &\geq x_j \cdot b_j, \text{ for all } j \in L, \\
\textstyle\sum_{i:R'_j \in C_i} y_i &\geq (1 - x_j) \cdot a_j, \text{ for all } j \in L.
\end{aligned}$$

Let $W(C_i)$ is the width of configuration C_i. We can construct a layered fractional packing of the frames in $L(x)$ as follows. We first define $\ell_0 = 0$ and $\ell_i = \ell_{i-1} + y_i$ ($i = 1, \ldots, N$). The ith layer is a rectangle, $Q_i = [0, W(C_i)] \times [\ell_{i-1}, \ell_i]$, see Fig. 5.3

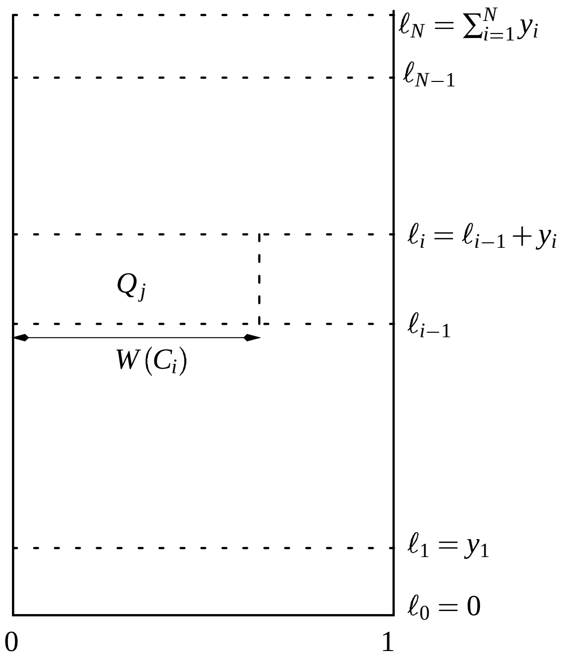


Figure 5.3: The ith layer

Next, we take large rectangles (a_j, b_j) from L one by one. The sum of y_i over all configurations C_i that include a non-rotated frame R_j, $\sum_{i:R_j \in C_i} y_i$, is at least $x_j \cdot b_j$. So, we select these C_i one by one in a greedy manner, and place a fractional non-rotated frame $a_j \times y_i$ in the ith layer. Similarly we deal with the rotated frame of (a_j, b_j), R'_j. In the end of this procedure, we obtain a fractional packing of the frames in $L(x)$, where all Q_i are filled with the frames from $L(x)$. The height of the packing is at most

$$\sum_{i=1}^{N} y_i = h.$$

Recall the rounding procedure. Let $\varepsilon' = \varepsilon/(2+\varepsilon)$ and $m = \lceil 1/(\varepsilon')^2 \rceil$. There are $m-1$ threshold frames and m groups. Let $L^k(x)$ and $L^k(\bar{x})$ denote the kth groups with respect to $L(x)$ and $L(\bar{x})$. Due to rounding, the width of $L^k(\bar{x})$ is at most the width of $L^{k-1}(x)$, whereas the height of $L^k(\bar{x})$ is at most the height of $L^{k-1}(x)$ plus 2. Hence, $L^k(\bar{x})$ can fractionally replace $L^{k-1}(x)$ in the layres, leaving just a small portion of frames.

We take the packing of $L(x)$, and go from one group to another. We replace the large frames in $Q_1, Q_2, \ldots, Q_N$ as follows. The frames of $L^1(x)$ are the widest ones. We simply delete them from the packing. This creates a set of gaps. Each gap has width at least the width of $L^2(\bar{x})$. So, we can fractionally pack inside these gaps. We put all the frames of $L^2(\bar{x})$ in a greedy manner while filling the gaps. Similarly, we create some gaps by deleting the frames of $L^{k-1}(x)$, and then fractionally pack the frames of $L^k(\bar{x})$. In the end, we take all the frames that are still left, including the frames of $L^1(\bar{x})$, and pack them one by one on the top of the packing.

There are at most m groups. In each group, the total height of the frames that go on the top of the packing is at most 2. The height of $L^1(\bar{x})$ is at most $(\varepsilon')^2 H + 1$. (Here, 1 for one threshold frame.) Hence, the height of the packing increases by at most

$$\Delta = 2m + (\varepsilon')^2 H + 1.$$

Recall that either side length of a large rectangle in L is at least ε'. Hence, the total area of L, $area(L)$, is at least $\varepsilon' H$. From another side, $area(L)$ is a lower bound on $lin(L)$. So, a possible increase can be bounded by

$$\Delta = 2m + 1 + (\varepsilon')^2 H \leq 2m + 1 + \varepsilon' \cdot lin(L).$$

Recall that h is also a lower bound on $lin(L)$. Thus, the total height of the packing of $L(\bar{x})$ is at most

$$h + \Delta \leq (1 + \varepsilon') lin(L) + 2m + 1 \leq (1 + \varepsilon') lin(L) + 3m.$$

This completes the proof of Lemma 5.2.4.

5.3.3 Proof of Lemma 5.2.5

Here we combine all obtained results together. The height of the packing can be bounded by

$$
\begin{aligned}
H \leq\ & STRIP(R'(\bar{x})) + |T| \\
& \text{since Lemma 5.2.1} \\
& \leq \max\{lin(L(\bar{x}))(1+\varepsilon) + 2m + 1, area(R)(1+\varepsilon) + 4m + 1\} + |T| \\
& \text{since } \varepsilon' = \varepsilon/(2+\varepsilon) \text{ and } m = \lceil (1/\varepsilon')^2 \rceil \\
& \leq \max\{lin(L)(1+\varepsilon')^2 + 3m(1+\varepsilon') + 2m + 1, area(R)(1+\varepsilon) + 4m + 1\} + m \\
& \text{since Lemma 5.2.4 and 5.2.3} \\
& \leq \max\{lin(L)(1+\varepsilon/2)^2 + 8m + 1, area(R)(1+\varepsilon) + 4m + 1\} + m \\
& \text{since } \varepsilon' \leq \varepsilon/2 \leq 1/2, \\
& \leq (1+\varepsilon/2)^2 \mathrm{OPT}(R) + 9m + 1 \\
& \text{since } \mathrm{OPT}(R) \geq \min\{lin(L), area(R)\}, \\
& \leq (1+2\varepsilon)\mathrm{OPT}(R) + 81/\varepsilon^2 + 1 \text{ since } m \leq 9/\varepsilon^2.
\end{aligned}
$$

This completes the proof of Lemma 5.2.5.

5.3.4 Proof of Theorem 5.1.1

The running time of the algorithm follows from the fact that the LP relaxation can be solved in time polynomial in n and $1/\varepsilon$ [46, 48], as well as from the running time mentioned in Lemma 5.2.2 and Lemma 5.2.1. A bound on the height of the packing output by A_ε follows from Lemma 5.2.5.

5.4 CONCLUDING REMARKS

In this chapter we consider the strip packing problem with rotations by 90 degrees. The problem has been an open question for some time. We close this gap, obtaining the algorithm which, given a set of rectangles, finds a strip packing of them (rotations by 90 degrees are allowed) of total height at most $(1+\varepsilon)\mathrm{OPT}+O(1/\varepsilon^2)$, where OPT is the height of the optimal strip packing with rotations by 90 degrees. The running time of the algorithm is polynomial in the number of rectangles and $1/\varepsilon$. In other words we have obtained an asymptotic FPTAS for the strip packing problem with rotations.

APPENDIX A: COMPLEXITY AND NPO PROBLEMS

Here we give an overview of complexity theory for the algorithm designer. This only includes some main definitions. For more details we refer to the following excellent books [6, 36, 73].

Complexity Classes. Let $\{0,1,\}^*$ be the set of all possible strings over alphabet $\{0,1\}$. Denote by $|x|$ the length of a string x. A language $L \subseteq \{0,1\}^*$ is any collection of strings over $\{0,1\}$. The corresponding *language recognition* problem is to decide whether a given string $x \in \{0,1\}^*$ belongs to L. An algorithm solves a language recognition problem for a specific language L by *accepting* (output "yes") any input string contained in L, and *rejecting* (output "no") any input string not contained in L.

A complexity class is a collection of languages all of whose recognition problems can be solved under prescribed bounds on the the computational resources. We are primarily interested in various of efficient algorithms, where efficient is defined as being *polynomial time*. Recall that an algorithm has polynomial running time if it halts within $n^{O(1)}$ on any input of length n.

The class P consists of all languages L that have a polynomial time algorithm ALG such that for any input string $x \in \{0,1\}^*$,

- $x \in L \Longrightarrow \mathrm{ALG}(x)$ accepts, and
- $x \notin L \Longrightarrow \mathrm{ALG}(x)$ rejects.

The class NP consists of all languages L that have a polynomial time algorithm ALG such that for any input string $x \in \{0,1\}^*$,

- $x \in L \Longrightarrow$ there is a string $y \in \{0,1\}^*$, $\mathrm{ALG}(x,y)$ accepts, where length $|y|$ is polynomial in $|x|$.

- $x \notin L \Longrightarrow$ for any string $y \in \{0,1\}^*$, $\mathrm{ALG}(x,y)$ rejects.

Obviously, $\mathrm{P} \subseteq \mathrm{NP}$, but it is not known whether $\mathrm{P} = \mathrm{NP}$.

For any complexity class $\mathcal{C}$, we define the complexity class co-$\mathcal{C}$ as the set of languages whose complement is in class $\mathcal{C}$. That is

$$\text{co-}\mathcal{C} \;=\; \{L | \overline{L} \in \mathcal{C}\}.$$

It is obvious that $\mathrm{P} = \text{co-P}$ and $\mathrm{P} \subseteq \mathrm{NP} \cap \text{co-NP}$.

NP-**completeness.** A *polynomial reduction* from a language $L' \subseteq \{1,0\}^*$ to a language $L \subseteq \{1,0\}^*$ is function $f : \{1,0\}^* \to \{1,0\}^*$ such that:

- There is a polynomial time algorithm that computes f.

- For all $x \in \{1,0\}^*$, $x \in L'$ if and only if $f(x) \in L$.

Clearly, if there is a polynomial reduction from L' to L, then $L \in \mathrm{P}$ implies that $L' \in \mathrm{P}$.

A language L is NP-*hard* if for every language $L' \in \mathrm{NP}$, there is a polynomial reduction from L to L'. A language L is NP-*complete* if $L \in \mathrm{NP}$ and L is NP-hard.

Randomized Complexity Classes. The class RP (for Randomized Polynomial Time) consists of all languages $L \subseteq \{0,1\}^*$ that have a randomized algorithm ALG running in worst-case polynomial time such that for any $x \in \{0,1\}^*$:

- $x \in L \Longrightarrow \mathbf{Pr}[\mathrm{ALG}(x) \text{ accepts}] \geq \frac{1}{2}$.

- $x \notin L \Longrightarrow \mathbf{Pr}[\mathrm{ALG}(x) \text{ accepts}] = 0$.

Clearly,

$$\mathrm{P} \subseteq \mathrm{RP} \subseteq \mathrm{NP}.$$

A language belonging to both RP and co-*RP* can be solved by a randomized algorithm with *zero-sided error*, i.e., a *Las Vegas* algorithm. The class ZPP (for Zero-error Probabilistic Polynomial time) is the class of all languages that have Las Vegas algorithms running in expected polynomial time. Clearly,

$$\mathrm{ZPP} \; = \; \mathrm{RP} \cap \text{co-RP}.$$

NP**-hard Decision Problems.** Informally, a *decision problem* is one whose answer is either "yes" or "no", and it can be treated as a language recognition problem.

Abstractly, a decision problem Π consists simply of a set D_Π of *instances* and a subset $Y_\Pi \subseteq D_\Pi$ of *yes-instances*. An *encoding scheme* for problem Π provides a way of describing each instance I in D_Π by an appropriate string in $\{0,1\}^*$. Then, the language assosited with Π is defined as

$$L[\Pi] := \{x \in 0,1^* | x \text{ is the encoding under } e \text{ of an instance } I \in Y_\Pi\}.$$

We say that a decision problem Π is NP-hard (complete) if $L[\Pi]$ is NP-hard (complete).

There are two common ways for encoding numbers (integers): *unary* and *binary*. Clearly, the hardness of a decision problem can change when one switches from binary to unary encoding.

We say that a decision problem Π is NP-hard (complete) in the *strong sense* or Π is *strongly* NP-hard (complete) if $L[\Pi]$ is NP-hard (complete) under an unary encoding scheme.

NPO Problems. An NP-*optimization problem* (NPO), Π, consists of:

- A set of *input instances*, $\mathfrak{I}$, recognized in polynomial time. The *size* of instance $I \in \mathfrak{I}$, denoted by $|I|$, is defined as the number of bits needed to write I under the assumption that all numbers occurring in I are written in binary.

- Each instance $I \in \mathfrak{I}$ has a set of feasible solutions $F(I)$. We require that $F(I) \neq \emptyset$, and that every solution $S \in F(I)$ is of length polynomial in $|I|$. Furthermore, there is polynomial time algorithm that, given a pair (I,S), decides whether $S \in F(I)$.

- There is a polynomial time computable *objective function*, obj, that assigns a nonnegative rational number to each pair (I,S), where $I \in \mathfrak{I}$ and $S \in F(I)$.

- Finally, Π is specified to be either a *minimization problem* or a *maximization problem*.

An *optimal solution* for an instance of a minimization (maximization) NPO problem is a feasible solution that achieves the smallest (largest) objective function value. $OPT(I)$ will denote the objective value of an optimal solution for instance I.

An algorithm ALG is said to be *optimal* for an NPO problem Π if, on each instance I, ALG computes an *optimal solution*, i.e. a feasible solution $S \in F(I)$ such that $obj(I,S) = OPT(I)$, and the running time of ALG is polynomial in I.

The decision version of an NPO problem Π consists of pairs (I,B), where I is an instance of I and B is a rational number. If Π is a minimization problem (maximization problem), then the answer to the decision problem is “yes” iff there is a feasible solution to I of the objective function value $\leq B$ ($\geq B$). If so, we will say that (I,B) is a yes-instance.

An NPO problem Π is said to be (strongly) NP-hard if its decision version is (strongly) NP-complete. Assuming P $\neq$ NP, no (strongly) NP-hard NPO problem has an optimal algorithm.

Approximation Algorithms. An approximation algorithm produces a feasibel "near-optimal" solution, and it is time efficient. The formal definition differs for minimization and maximization problems. Let Π be a minimization problem. An algorithm ALG is said to be a ρ-approximation algorithm for Π, if on every instance I of Π, ALG computes a feasible solution $S \in F(I)$ such that

$$obj(I,S) \;\leq\; \rho \cdot OPT(I),$$

and the running time of ALG is polynomial in $|I|$. For a maximization problem Π, a ρ-approximation algorithm satisfies

$$obj(I,S) \;\geq\; \frac{1}{\rho} \cdot OPT(I).$$

The asymmetry in the definition is due to ensure that $\rho \geq 1$. The value of $\rho \geq 1$ is called the *approximation ratio* or *performance ratio* or *worst-case ratio* of ALG and in general can be a function of $|I|$.

A family of approximation algorithms, $\{A_\varepsilon\}_{\varepsilon>0}$, for an NPO problem Π, is called a *polynomial time approximation scheme* or a PTAS, if algorithm A_ε is a $(1+\varepsilon)$-approximation algorithm and its running time is polynomial in the size of the instance for a fixed ε. If the running time of each A_ε is polynomial in the size of the instance and in $1/\varepsilon$, then $\{A_\varepsilon\}_{\varepsilon>0}$ is called a *fully polynomial time approximation scheme* or a FPTAS.

Assuming $P \neq NP$, a PTAS is the best result we can obtain for a strongly NP-hard problem, and a FPTAS is the best result we can obtain for an NP-hard problem.

AP-Reduction. The concept of approximation preserving reductions primarily provides a method for proving that an NPO problem does not admit any PTAS, unless $P = NP$.

For a constant $\alpha \geq 0$ and two NPO problems A and B, we say that A is α-AP-reducible to B if two polynomial-time computable functions f and g exist such that the following holds:

- For any instance I of A, $f(I)$ is an instance of B.

- For any instance I of A, and any feasible solution S' for $f(I)$, $g(I,S')$ is a feasible solution for I.

- For any instance I of A and any $r \geq 1$, if S' is is an r-approximate solution for $f(I)$, then $g(I,S')$ is an $(1+(r-1)\alpha+o(1))$-approximate solution for I, where the o notation is with respect to $|I|$.

We say that A is AP-reducible to B if a constant $\alpha \geq 0$ exists such that A is α-AP-reducible to B. Clearly, if A is AP-reducible to B, then an ρ-approximate solution for B is mapped to an $h(\rho)$ approximate solution for A, where $h(\rho) \to 1$ as $\rho \to 1$.

The class APX consists of all NPO problems that have a constant factor approximation. Then, AP-reductions preserve membership in APX. Furthermore, if A is AP-reducible to B and there is a PTAS for B, there is a PTAS for A as well.

An NPO problem Π is APX-hard if every APX problem is AP-reducible to Π. An NPO problem Π is APX-*complete* if $\Pi \in$ APX and Π is APX-hard.

Assuming P $\neq$ NP, no APX-hard (complete) problem has a PTAS.

A Little Bit of History. In [39] a simple algorithm for scheduling jobs on a single machine was presented: Suppose we are given a single machine and a list of n jobs in some order. Whenever a machine becomes available, it starts processing the next job on the list. Graham made a complete worst-case analysis of this algorithm and showed that the maximum job completion time (or *makespan*) of the schedule is at most twice the makespan of an optimal schedule. It was perhaps the first polynomial time approximation algorithm for an NP-hard optimization problem, and at the same time, the first competitive analysis of an on-line algorithm.

Only several years later, immediately after the concepts of NP-completeness and approximation algorithms were formalized [16, 34]. However, a paper [52] of Johnson may be regarded as the real starting point in the field. The terms "approximation scheme", "PTAS","FPTAS" are due to a seminal paper [35]. The first inapproximability results were also derived about this time, see e.g. [77, 61].

Much of the work has been also devoted to classifying the optimization problems with respect to their polynomial time approximability. The notion of *strong* NP-*completeness* was introduced in [35]. It was also shown that *strong* NP-*hard* problems do not have FPTASs unless P $=$ NP [36]. A *strongly* NP-hard problem is a problem that remains NP-hard even if the numbers in its input are unary encoded [36].

In [75] the class MAX-SNP was introduced by a logical characterization and the notion of completeness for this class by using the so-called *L*-reduction. The idea behind this concept was that every MAX-SNP-complete optimization problem does not admit any PTAS iff MAX-3SAT does not admit any PTAS. A number of optimization problem were proven to be MAX-SNP-complete. In a remarkable line of work that culminated in [5], it was shown that MAX-3SAT has no PTAS, unless P $=$ NP.

Later, based on known results about the approximability thresholds of various problems, researches have classified problems into a number of classes [4]. One of these classes is APX. It was established in [58, 20, 21] that MAX-3SAT is APX-complete under AP-reduction and under subtler notion of reductions. Many problems have been shown to be either APX-complete or APX-hard, and thus do not have a PTAS, unless P $=$ NP.

Generalizing NP to allow for *randomized* algorithms has led to a number of new complexity classes, e.g. ZPP (Zero-error Probabilistic Polynomial) and PCP (Probabilistically Checkable Proofs). It was shown that the so-called PCP-theorem ($\mathrm{NP} = \mathrm{PCP}(\log n, 1)$) implies that the problem of finding a maximum clique in an n-vertex graph cannot be approximated within a factor of $n^{1-\varepsilon}$, neither for some $\varepsilon > 0$, unless P $=$ NP; nor for any $\varepsilon > 0$, unless NP $=$ ZPP [3, 4, 6, 64].

APPENDIX B: KR - ALGORITHM

Here we briefly describe the algorithm of C. Kenyon and E. Rémila for the strip packing problem. For more details we refer to the original paper [57].

Definitions. A rectangle is given by its width w_i and height h_i, with $0 \leq w_i, h_i \leq 1$. The area (resp. height) of a list $L = ((w_1, h_1), (w_2, h_2), \dots, (w_n, h_n))$ of rectangles is the sum of the areas (resp. heights) of the rectangles of L. We assume that the list is ordered by nonincreasing widths: $w_1 \geq w_2 \geq \dots \geq w_n$.

A *strip-packing of a list L of rectangles is a positioning of the rectangles of L within the vertical strip* $[0,1] \times [0,+\infty)$*, so that all rectangles have disjoint interiors. If rectangle* (w_i, h_i) *is positioned at* $[x, x+w_i] \times [y, y+h_i]$*, then y is called the lower boundary* $(y+h_i)$ *the upper boundary of the rectangle. The height of a strip-packing is the uppermost boundary of any rectangle. Let* $Opt(L)$ *denote the minimum height of a strip-packing of L:*
$Opt(L) = \inf\{\textit{height of f such that f is a packing of L}\}$.

A fractional strip-packing of L is a packing of any list L' obtained from L by subdividing some of its rectangles by horizontal cuts: Each rectangle (w_i, h_i) is replaced by a sequence $(w_i, h_{i_1}), (w_i, h_{i_2}), \dots, (w_i, h_{i_{k_i}})$ of rectangles, such that $h_i = \sum_j h_{i_j}$.

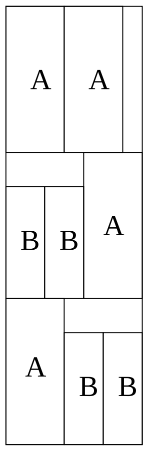
First we present the algorithm when the number of distinct widths of the rectangles is bounded by some value m, and all widths are larger than some constant ε'. This special case is called the "few and wide" case.

From the "few and wide" case to fractional strip-packing. Throughout this paragraph, one assumes that the n rectangles of L only have m distinct widths,

$w'_1 \geq w'_2 \cdots \geq w'_m \geq \varepsilon$.

To the input L, one associates a set of configurations. A configuration is defned as a nonempty multiset of widths (chosen among the m widths) that sum to less than 1 (i.e., capable of occurring at the same level). Their sum is called the *width* of the confguration. Without loss of generality, the configurations can be assumed to be ordered by nonincreasing widths.

Let q be the number of distinct configurations, and let α_{ij} denote the number of occurrences of width w'_i in configuration C_j. To each (possibly fractional) strip



	configuration	α_{1j} = number of As	α_{2j} = number of Bs
C_1	3/7,2/7,2/7	1	2
C_2	3/7,3/7	2	0
C_3	2/7,2/7,2/7	0	3
C_4	3/7,2/7	1	1
C_5	2/7,2/7	0	2
C_6	3/7	1	0
C_7	2/7	0	1

Figure 5.4: A strip packing of L.

packing of L of height h, one associates a vector $(x_1,\ldots,x_q)$, $x_i \geq 0$, in the following manner. Scan the packing bottom-up with a horizontal sweep line $y = a$, $0 \leq a \leq h$. Each such line is canonically associated to a configuration $(\alpha_1,\ldots,\alpha_m)$, where α_i is the number of rectangles of width w'_i whose interior is intersected by the sweep line. Let x_j, $1 \leq j \leq q$, denote the measure of the as such that the sweep line $y = a$ is associated with configuration C_j. For example, let A denote the rectangle $3/7 \times 1$ and B denote the rectangle $2/7 \times 3/4$, and assume that the input L consists of three rectangles of type A and four rectangles of type B. There are seven configurations, listed in Fig. 5.4.

The vector corresponding to the strip packing in Fig. 5.4 is $(3/2, 5/4, 0, 0, 0, 0, 0)$. The fractional strip-packing problem is canonically defined as follows: Given a list L of rectangles, construct a fractional strip packing of minimal height.

Lemma 5.4.1. *Consider the linear program:*

$$\text{minimize } (1.x) \textit{ subject to } x \geq 0 \text{ and } Ax \geq B,$$

where 1 *is the all-ones vector,* A *is the* $m \times q$ *matrix* $(\alpha_{ij})_{1 \leq i \leq m, 1 \leq j \leq q}$*, and* $B = (\beta_1, \ldots, \beta_m)$*,* β_i *denoting the sum of the heights of all rectangles of width* w'_i*. Then any fractional strip packing naturally corresponds to a feasible vector* x*, and conversely to any feasible vector* x *one can associate a fractional strip packing of height* $(1.x)$ *and in which the number of configurations actually occurring is at most* m *plus the number of nonzero variables* x_i*.*

We now recall the fractional bin packing problem studied by Karmarkar and Karp [53]. In this problem, the input is a set of n items of m different types, i.e., they take only m distinct sizes in $(\varepsilon, 1]$. A configuration is a multi-set of types which sum to at most 1 (i.e., capable of being packed within a bin). If q denotes the number of configurations, then a feasible solution to the fractional bin packing problem is a vector $(x1, \ldots, x_q)$ of nonnegative numbers such that if α_{ij} is the number of pieces of type i occurring in configuration j, then for every i, $\sum_j \alpha_{ij} x_j$ is at least equal to the number n_i of input pieces of type i. The goal is to minimize $\sum_j x_j$.

Notice that fractional bin packing and fractional strip packing give rise to the same linear program. The only difference is that vector $B = (\beta_1, \ldots, \beta_m)$ of the strip packing is replaced by the vector $B' = (n_1, \ldots, n_m)$ with integer coordinates.

Let OPT be the minimum possible value of $\sum_j x_j$. The fractional bin packing problem with tolerance t has for its goal to find a basic feasible solution such that

$\sum_j x_j \leq \mathrm{OPT}+t$, and was solved by Karmarkar and Karp [53] in polynomial time. More precisely, one has the following theorem:

Theorem 5.4.2. *(Karmarkar and Karp [53], Theorem 1.) There exists a polynomial-time algorithm for fractional bin packing with additive tolerance t, such that if n is the number of items, m the number of distinct items, and a the size of the smallest item, then the running time is*

$$O\left(m^8 \log m \log^2\left(\frac{mn}{at}\right) + \frac{m^4 n \log m}{t} \log\left(\frac{mn}{at}\right)\right).$$

The proof of this theorem uses linear programming techniques but does not use the fact that vector B' is integer. It can obviously be extended to strip packing: with the notations of Lemma 5.4.1, there exists an algorithm with positive tolerance t whose running time is polynomial in m, $\sum_i \beta_i$ (which is less than the number n of rectangles) and t, which gives a solution with at most $2m$ nonzero coordinates.

In our setting a, m and t will all be polynomials in $1/\varepsilon$, and so the running time will be $O_\varepsilon(n \log n)$. Note that using a Lagrangian relaxation technique, in [76] (Theorem 5.11), an alternative approach is proposed.

From fractional strip packing to strip packing.

Lemma 5.4.3. . *If L has a fractional strip packing $(x_1,\ldots,x_q)$ of height h and with at most $2m$ nonzero x_js, then L has an (integral) strip packing of height at most $h+2m$.*

Proof. Consider a fractional strip packing $(x1,\ldots,x_q)$ of L, of height $\sum_i xi = h$, and with at most $2m$ nonzero coordinates x_is. Up to renaming, one assumes that the nonzero coordinates are $x_1,\ldots,x'_m$, with $m' \leq 2m$. Let h_{max} be the maximum height of any rectangle of L. One constructs a strip packing of L of height $h+2mh_{max}$ in the following way.

One fills in the strip bottom up, taking each configuration in turn. Let $x_j \geq 0$ denote the variable corresponding to the current configuration. Configuration j will be used between level $l_j = (x_1+h_{max})+\cdots+(x_{j-1}+h_{max})$ and level $l_{j+1} =$

$l_j + x_j + h_{max}$ (initially $l_1 = 0$). For each i such that $\alpha_{ij} \neq 0$, we draw α_{ij} columns of width w'_i going from level l_j to level l_{j+1}.

In this way, each column C of the fractional strip packing of width w'_i and height x_j can be associated to a column C_+ width w'_i and height $x_j + h_{max}$. In C_+, we place the rectangles which are completely in C, and the rectangle whose bottom is in C and whose top is in another column. There is at most one rectangle of this type from the proof of Lemma 5.4.1. Obviously, C_+ is sufficiently large to contain those rectangles. This proves that the construction yields a valid strip packing of L. Its height is $(x_1 + h_{max}) + \cdots + (x_{m'} + h_{max}) = h + m'h_{max} \leq h + 2m$, hence the lemma. □

This gives a straightforward algorithm for strip-packing in the special case studied in this section.

1. Solve fractional strip packing on L with tolerance 1 (the solution has at most $2m$ nonzero coordinates).

2. From the fractional strip packing, construct a strip packing of L as in the proof of the lemma above.

Moreover, a crucial point for the sequel (i.e., for the addition of narrow rectangles) is that this strip packing leaves some well-structured free space. Note that in the proof of Lemma 5.4.3, column C_+ is almost fully used: the unused part of the column has height at most 2, one for the bottom rectangle of C which may have been placed in another column, and one for the extra space on top.

Important remark: Structure of a layer (See Fig. 5.5).

Let $c1 \geq c2 \geq \cdots \geq c_{m'}$ denote the widths of the m'_i configurations used above. The layer $[0,1] \times [l_i, l_{i+1}]$ can be divided into three rectangles:

(i) the rectangle $R_i = [c_i, 1] \times [l_i, l_{i+1}]$, which is completely free and will later be used to place the narrow rectangles;

(ii) the rectangle $R'_i = [0, c_i] \times [l_i, l_{i+1} - 2]$, which is completely filled by wide rectangles; and

(iii) the rectangle $R_i'' = [0, c_i] \times [l_{i+1} - 2, l_{i+1}]$, which is partially filled in some complicated way by wide rectangles overlapping from R_i, and whose free space is now considered as wasted space, and will not be used in the remainder of the construction.

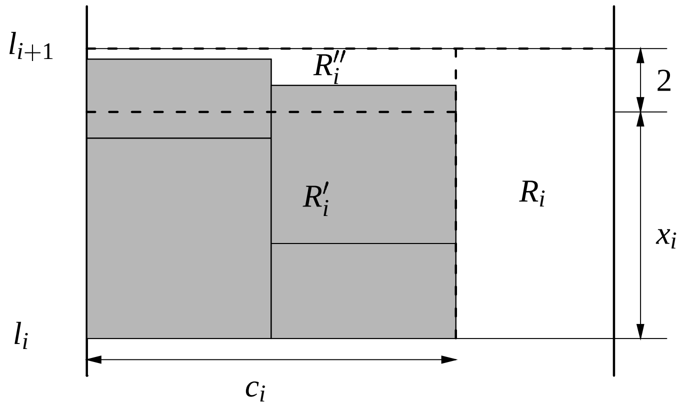


Figure 5.5: Structure of a layer.

From general strip-packing to the "few and wide" case. In the general case, one has a list $L_{general}$ with many distinct widths, some of which may be arbitrarily small.

One uses appropriate extensions of two ideas of Fernandez de la Vega and Lueker [23]: elimination of small pieces, and grouping. The purpose of elimination is to insure all rectangles are wider than some ε'. The purpose of grouping is to insure that the number of distinct widths of the wide rectangles is bounded.

Elimination of narrow rectangles. During the elimination phase, one partitions the list $L_{general}$ into two sublists: L_{narrow}, containing all the rectangles of width at most ε' , and L, containing all the rectangles of width larger than ε' . During the next stage, we will focus on L.

Grouping. One defines a partial order on lists of rectangles by saying that $L \le L'$ if there is an injection from L to L' such that each rectangle of L has smaller width and height than the associated rectangle of L'.

Given a list L of rectangles whose widths are larger than ε' , we will now approximate L by a list L_{sup} such that $L \le L_{sup}$, and such that the rectangles of L_{sup} only have m distinct widths.

To define L_{sup}, one first stacks up all the rectangles of L by order of nonincreasing widths, to obtain a left-justified stack of total height $h(L)$. One defines $(m-1)$ threshold rectangles, where a rectangle is a threshold rectangle if its interior or lower boundary intersects some line $y = ih(L)/m$, for some i between 1 and $m-1$ (see, for example, Fig. 5.6). The threshold rectangles separate the remaining rectangles into m groups. The widths of the rectangles in the first group are then rounded up to 1, and the widths of the rectangles in each subsequent group are then rounded up to the width of the threshold rectangle below their group. This defines L_{sup}. Note that if all rectangle heights are equal, this is exactly the linear grouping defined in [23], and thus this can be seen as an extension of that paper. Also note that L_{sup} consists of rectangles which have only m distinct widths, all greater than ε'.

One constructs a strip-packing of L_{sup} using the ideas of previous paragraphs. A packing of L is trivially deduced by using the relation $L \le L_{sup}$ and placing each rectangle of L inside the position of the associated rectangle of L_{sup}.

To get a packing of $L_{general}$, the narrow rectangles must now be added.

Adding the narrow rectangles. Order the rectangles of L_{narrow} by decreasing heights. We add the rectangles of L_{narrow} to the current strip packing, trying to use the m' free rectangular areas $R_1, R_2, \ldots, R_{m'}$ as much as possible, according to a Modified-Next- Fit-Decreasing-Height algorithm as follows. Use the Next-Fit-Decreasing-Height (NFDH) heuristic to pack rectangles in R_1: In this heuristic, the rectangles are packed so as to form a sequence of sublevels. The first sublevel is simply the bottom line. Each subsequent sublevel is defined by a horizontal

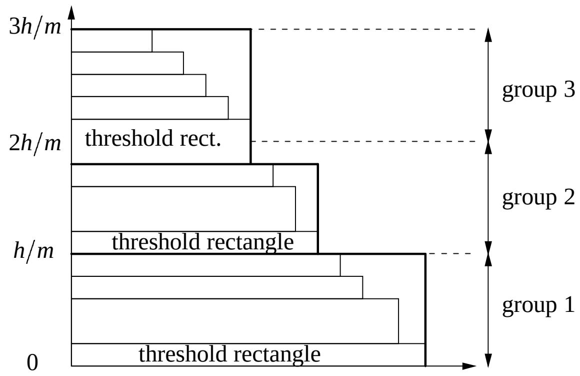


Figure 5.6: Grouping the rectangles, example when $m = 3$. The thick lines show how to extend the rectangles to construct L_{sup}.

line drawn through the top of the first (and hence highest) rectangle placed on the previous sublevel. Rectangles are packed in a left-justified greedy manner, until there is insufficient space to the right to accommodate the next rectangle; at that point, the current sublevel is discontinued, the next sublevel is defined and packing proceeds on the new sublevel.

When a new sublevel cannot be started in R_1, start the next sublevel at the bottom left corner of R_2 using NFDH again, and so on until $R_{m'}$. When a rectangle cannot be packed in $R_1, \ldots,$ or $R_{m'}$, use NFDH to pack the remaining rectangles in the strip of width 1 starting above $R_{m'}$, at level $l_{m'+1}$. This gives a packing of $L_{general}$.

We are now ready to summarize the overall algorithm.

The KR-algorithm. Parameters: ε' (the threshold narrow/wide) and m (the number of groups). We set $\varepsilon' = \varepsilon/(2+\varepsilon)$ and $m = (1/\varepsilon')^2$.

Input: a list of rectangles $L_{general}$.

1. Perform the partition $L_{general} = L_{narrow} \cup L$ to set aside the rectangles of width less than ε'.

2. Sort the rectangles of L according to their widths; form m groups of rectangles of approximately equal cumulative heights; round up the widths in

each group, to yield a list L_{sup} with $L \leq L_{sup}$.

3. Solve fractional strip packing on L_{sup} with tolerance 1.

4. From the fractional strip packing, construct an integral strip packing of L_{sup} and hence a well-structured strip packing of L.

5. Sort L_{narrow} according to decreasing heights and add the rectangles of L_{narrow} to the strip packing of L using the Modified-Next-Fit-Decreasing-Height heuristic.

Theorem 5.4.4. *For a given list L of n rectangles whose side lengths are at most* 1*, and a positive number ε, the KR-algorithm produces a packing of L in a strip of width* 1 *and height $A(L)$ such that:*

$$A(L) \leq (1+\varepsilon)Opt(L) + O(1/\varepsilon^2).$$

The time complexity of the algorithm is polynomial in n and $1/\varepsilon$.

APPENDIX C: RESOURCE SHARING PROBLEM

Here we briefly describe the algorithm of M.D. Grigoriadis, L.G. Khachiyan, L. Porkolab and J. Villavicencio for the max-min resource sharing problem. For more details we refer to the original paper [40].

We consider the approximate solution of concave max-min resource sharing problem of the form

$$\lambda^* = \max\{\lambda \mid f(x) \geq \lambda e, x \in B\}, \tag{$\mathcal{P}$}$$

where $f : B \to \mathbb{R}^M$ is a given vector of M nonnegative continuous concave functions defined on a nonempty convex compact set B, called *block*, e is the vector of all ones and with no loss of generality, $\lambda^* > 0$. We shall denote by $\mathbb{R}^M_+$ ($\mathbb{R}^M_{++}$) the nonnegative (positive) orthants of $\mathbb{R}^M$, and denote $\lambda(f) \doteq \min_{1 \leq m \leq M} f_m$ for any given $f \in \mathbb{R}^M_+$.

We shall be interested in computing an ε-approximate solution of this problem, i.e., for a given *relative tolerance* $\varepsilon \in (0,1)$,

$$\text{compute } x \in B \text{ such that } f(x) \geq [(1-\varepsilon)\lambda^*]e. \tag{$\mathcal{P}_\varepsilon$}$$

The approach is based on the well-known duality relation:

$$\lambda^* = max_{x \in B} min_{p \in P} p^T f(x) = min_{p \in P} max_{x \in B} p^T f(x), \tag{5.2}$$

where $P \doteq \{p \in \mathbb{R}^M_+ | e^T p = 1\}$. It follows that

$$\lambda^* = \min\{\Lambda(p) \mid p \in P\}, \tag{Lagrangian dual}$$

where

$$\Lambda(p) = \max\{p^T f(x) \,|\, x \in B\}. \qquad \text{(Block problem)}$$

The exact optimality conditions for $\mathcal{P}$ can thus be stated as follows: A pair $x \in B$, $p \in P$ is optimal if and only if $\Lambda(p) = \lambda(f(x))$.

In its simplest form, *Lagrangian* or *price-directive* decomposition is an iterative strategy that solves $\mathcal{P}$ via its Lagrangian dual by computing a sequence of pairs p, x as follows. A *coordinator uses* the current $x \in B$ to compute some weights $p = p(f(x)) \in P$ corresponding to the coupling constraints $f(x) \geq \lambda e$, calls a *block solver* to compute a solution $\hat{x} \in B$ of (Block problem) for this $p \in P$, and then makes a move from x to $(1-\tau)x + \tau\hat{x}$ with an appropriate step length $\tau \in (0,1]$. We call each such Lagrangian decomposition iteration a *coordination step*.

We shall only require an *approximate block solver* ($\mathcal{ABS}$), one that solves (Block problem) to a given optimization tolerance $t > 0$, defined below.

$$\mathcal{ABS}(p,t): \qquad \text{compute } \hat{x} = \hat{x}(p) \in B \text{ such that } p^T f(\hat{x}) \geq [(1-t)]\Lambda(p).$$

We shall eventually set $t = \Theta(\varepsilon)$ in our algorithm.

By analogy to $\mathcal{P}_\varepsilon$, and based on the fact that λ^* is the optimal value of the Lagrangian dual, we define the ε-approximate dual problem as follows:

$$\text{compute } p \in P \text{ such that } \Lambda(p) \leq [(1+\varepsilon)]\lambda^*. \qquad (\mathcal{D}_\varepsilon)$$

For a given relative accuracy $\varepsilon \in (0,1)$ a presented approximation algorithm solves problems $\mathcal{P}_\varepsilon$ and $\mathcal{D}_\varepsilon$ in $N = O(M(\varepsilon^{-2} + \ln M))$ coordination steps, each of which requires a call to $\mathcal{ABS}(p, \Theta(\varepsilon))$ and a coordination overhead of $O(M\ ln(M/\varepsilon))$ arithmetic operations.

The lemma below states that a pair x, p solves $\mathcal{P}_\varepsilon$ and $\mathcal{D}_\varepsilon$, respectively, whenever ν and t are of order ε.

Lemma 5.4.5. *Suppose $\varepsilon \in (0,1)$ and $t = \varepsilon/6$. For a given point $x \in B$, let $p \in P$ be computed by 5.4 and $\hat{x}$ computed by $\mathcal{ABS}(p,t)$. If $\nu(x,\hat{x}) \leq t$, then the pair x, p solves $\mathcal{P}_\varepsilon$ and $\mathcal{D}_\varepsilon$, respectively.*

Algorithm description. The algorithm solves $\mathcal{P}_\varepsilon$ (resp. $\mathcal{D}_\varepsilon$) approximately by computing a sequence of vectors $x_0, x_1, \ldots, x_n \in B$. In each step a price vector $p = p(x_i) \in P$ for the current vector $x_i \in B$ gets computed and the block solver is called to get an approximate solution $\hat{x} \in B$ of the block problem $\max\{p^T f(x) | x \in B\}$. The next vector gets set to $x_{i+1} = (1-\tau)x_i + \tau\hat{x}$ with an appropriate step length $\tau \in (0,1)$.

In computing the price vector $p(x)$ the standard logarithmic potential function is used of the form

$$\Phi_t(\theta, x) = \ln\theta + \frac{t}{M}\sum_{m=1}^{M} \ln(f_m(x) - \theta) \tag{5.3}$$

where $x \in B$, $\theta \in (0, \lambda(x))$ are variables and t is a tolerance parameter, the same as used for $\mathcal{ABS}(p,t)$. The potential function has an unique maximizer $\theta(x)$ for each $x \in B$. The *reduced potential function* $\phi_t(x) = \Phi_t(\theta(x), x)$ measures the improvement of the solution. The price vector $p = p(x)$ is defined through

$$p_m(x) = \frac{t}{M}\frac{\theta(x)}{f_m(x) - \theta(x)}, \quad m = 1, \ldots, M. \tag{5.4}$$

For deciding the stopping rule the following parameter is used:

$$\nu(x, \hat{x}) = \frac{p^T f(\hat{x}) - p^T f(x)}{p^T f(\hat{x}) + p^T f(x)}. \tag{5.5}$$

The algorithm can now be outlined as follows:

(1) compute initial solution $x^{(0)}$, $s := 0$, $\varepsilon_0 := 1/4$;

(2) **repeat** {scaling phase}

- **(2.1)** $s := s+1$; $\varepsilon_s := \varepsilon_{s-1}/2$; $t = \varepsilon_s/6$; $x := x^{(s-1)}$;
- **(2.2)** **while** *true* **do begin** {coordination phase}
  - **(2.2.1)** compute $\theta(x)$ and $p(x)$;
  - **(2.2.2)** $\hat{x} := \mathcal{ABS}(p(x), t)$;
  - **(2.2.2)** compute $\nu(x, \hat{x})$;
  - **(2.2.3)** **if** $\nu(x, \hat{x}) \le t$ **then begin** $x^{(s)} := x$; **break**; **end**

(2.2.4) compute step length τ and set $x' := (1-\tau)x + \tau\hat{x}$;

end

(2.3) **until** $\varepsilon_s \leq \varepsilon$;

(3) return($x^{(s)}$, $p(x^{(s)})$).

The initial solution is computed as $x^{(0)} = \frac{1}{M}\sum_{m=1}^{M} \mathcal{ABS}(e_m, 1/2)$, where e_m is the m-th unit vector. The step length used is

$$\tau = \frac{t\theta\nu}{2M(p^T f(\hat{x}) + p^T f(x))}. \tag{5.6}$$

In practice, one usually computes τ by performing a line search to maximize $\phi_t(x + \tau(\hat{x} - x))$, what does not worsen the complexity of the algorithm. The following result holds [40]:

Theorem 5.4.6. *For any given relative accuracy $\varepsilon \in (0, 1)$ the algorithm above computes a solution (x, p) of the problem $\mathcal{P}_\varepsilon$ (resp. $\mathcal{D}_\varepsilon$) in $N = O(M(\ln M + \varepsilon^{-2}))$ coordination steps.*

BIBLIOGRAPHY

[1] AARTS, E., AND LENSTRA, J. K., Eds. *Local search in Combinatorial Optimization*. Willey-Interscience series in discrete mathematics and optimization. John wiley and Sons, 1997.

[2] ADLER, M., GIBBONS, P., AND MATIAS, Y. Scheduling space-sharing for internet advertising. *Journal of Scheduling 5* (2002), 103–119.

[3] ARORA, S. *Probabilistic checking of proofs and the hardness of approximation problems*. PhD thesis, U.C. Berkley, 1994.

[4] ARORA, S., AND LUND, C. *Approximation algorithms for NP-hard problems*. PWS Publishing Company, Boston, 1996, ch. Hardness of approximation (D. S. Hochbaum ed.), pp. 1–45.

[5] ARORA, S., LUND, C., MOTWANI, R., SUDAN, M., AND SZEGEDY, M. Proof verification and hardness of approximation problems. In *Proceedings 33rd Annual IEEE Symposium on Foundations of Computer Science* (1992), pp. 14–23.

[6] AUSIELLO, G., GRESCENZI, P., GAMBOSI, G., KANN, V., MARCHETTI-SPACCAMELA, M., AND PROTASI, M. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Verlag, 1999.

[7] BAKER, B., BROWN, D., AND KATSEFF, H. A 5/4 algorithm for two dimensional packing. *Journal of Algorithms 2* (1981), 348–368.

[8] BAKER, B., CALDERBANK, A., COFFMAN, E., AND LAGARIAS, J. Approximation algorithms for maximizing the number of squares packed into a rectangle. *SIAM Journal on Algebraic and Discrete Methods 4* (1983), 383–397.

[9] BAKER, B., COFFMAN, E., AND RIVEST, R. Orthogonal packings in two dimensions. *SIAM Journal on Computing 9* (1980), 846–855.

[10] BANSAL, N., AND SVIRIDENKO, M. New approximability and inapproximability results for 2-dimensional bin packing. In *Proceedings 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2004), pp. 189–196.

[11] BERTSIMAS, D., AND TSITSIKLIS, J. N. *Introduction to Linear Optimization*. Athena Scientific, Belmont, Massachusetts, 1997.

[12] CAPRARA, A. Packing 2-dimensional bins in harmony. In *Proceedings 43rd Annual Symposium on Foundations of Computer Science(FOCS)* (2002), pp. 490–499.

[13] CHUNG, F., GAREY, M., AND JOHNSON, D. On packing two-dimentional bins. *SIAM Journal on Algebraic and Discrete Methods 3* (1982), 66–76.

[14] CLAY MATHEMATICS INSTITUTE. Millennium Prize Problems. URL: http://www.claymath.org/prizeproblems/index.htm, Announced 16:00, on Wednesday, May 24, 2000.

[15] COFFMAN, E., GAREY, M., JOHNSON, D., AND TARJAN, R. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing 9* (1980), 808–826.

[16] COOK, S. A. The complexity of theorem proving procedures. In *Proceedings 30th ACM Symposium on the Theory of Computing* (1971), pp. 151–158.

[17] COOK, W. J., CUNNINGHAM, W. H., PULLEYBLANK, W. R., AND SCHRIJVER, A. *Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley and Sons, Inc, 1998.

[18] CORREA, J., AND KENYON, C. Approximation schemes for multidimensional packing. In *Proceedings 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2004), pp. 179–188.

[19] CORREA, J. R. *Approximation algorithms for packing and scheduling problems*. PhD thesis, MIT, Cambridge, MA, 2004.

[20] CRESCENZI, P., KANN, V., SILVESTRI, R., AND TREVISAN, L. Structure in approximation classes. In *Proceedings 1st Computing and Combinatorics Conference* (1995), LNCS 959, Springer Verlag, pp. 539–548.

[21] CRESCENZI, P., AND TREVISAN, L. On approximation scheme preserving reducibility and its applications. *Theory of Computer Systems 33* (2000), 1–16.

[22] DANTZIG, G., AND THAPA, M. *Linear Programming 2: Theory and Extensions*. Springer Verlag, 2003.

[23] DE LA VEGA, W. F., AND LUEKER, S. G. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica 1* (1981), 349–355.

[24] DIEDRICH, F. Approximative Algorithmen für Rucksackprobleme, Diplomarbeit, University of Kiel, September 2004.

[25] DROZDOWSKI, M. Scheduling multiprocessor tasks - an overview. *European Journal of Operational Research 94* (1996), 215–230.

[26] FERREIRA, C., MIYAZAWA, F., AND WAKABAYASHI, Y. Packing squares into squares. *Perquisa Operacional 19* (1999), 349–355.

[27] FISHKIN, A., GERBER, O., AND K.JANSEN. On weighted rectangle packing with large resources. In *Proceedings 3rd IFIP International Conference on Theoretical Computer Science (TCS)* (2004), pp. 237–250.

[28] FISHKIN, A., GERBER, O., AND K.JANSEN. On efficient weighted rectangle packing with large resources. In *Proceedings 16th Annual International Symposium on Algorithms and Computation (ISAAC)* (2005), pp. 1039–1050.

[29] FISHKIN, A., GERBER, O., K.JANSEN, AND SOLIS-OBA, R. On packing rectangles with resource augmentation: maximizing the profit. Algorithmic Operations Research (to appear), 2005.

[30] FISHKIN, A., GERBER, O., K.JANSEN, AND SOLIS-OBA, R. On packing squares with resource augmentation: maximizing the profit. In *Proceedings 11th Computing: The Australasian Theory Symposium (CATS)* (2005), vol. 41, pp. 61–69.

[31] FISHKIN, A., GERBER, O., K.JANSEN, AND SOLIS-OBA, R. Packing weighted rectangles into a square. In *Proceedings 30th International Symposium on Mathematical Foundations of Computer Science* (2005), pp. 352–363.

[32] FISHKIN, A., AND ZHANG, G. On maximizing the throughput of multiprocessor tasks. *Theoretical Computer Science 302* (2003), 319–335.

[33] FREUND, A., AND NAOR, J. Approximating the advertisement placement problem. In *Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO)* (2002), LNCS 2337, pp. 415–424.

[34] GAREY, M. R., GRAHAM, R. L., AND ULLMAN, J. D. Worst case analysis of memory allocation algorithms. In *Proceedings 4th ACM Symposium on Theory of Computing* (1972), pp. 143–150.

[35] GAREY, M. R., AND JOHNSON, D. S. Strong NP-completeness results: Motivation, examples and applications. *Journal of the Association for Computing Machinery* (1978), 499–508.

[36] GAREY, M. R., AND JOHNSON, D. S. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, CA, 1979.

[37] GILMORE, P., AND GOMORY, R. Multistage cutting stock problems of two and more dimensions. *Operations Research 13* (1965), 94–120.

[38] GOLAN, I. Performance bounds for orthogonal, oriented two-dimensional packing algorithms. *SIAM Journal on Computing 10* (1981), 571–582.

[39] GRAHAM, R. L. Bounds for certain multiprocessing anomalies. *Bell system Technical Journal 45* (1966), 1563–1581.

[40] GRIGORIADIS, M., KHACHIYAN, L., PORKOLAB, L., AND VILLAVICENCIO, J. Approximate max-min resource sharing for structured concave optimization. *SIAM Journal on Optimization 41* (2001), 1081–1091.

[41] GRÖTSCHEL, M., LOVÁSZ, L., AND SCHRIJVER, A. *Geometric Algorithms and Combinatorial Optimization*. Springer Verlag, 1994.

[42] HALL, L. A. Approximability of flow shop scheduling. *Mathematical Programming 82* (1998), 175–190.

[43] HOCHBAUM, D. S., Ed. *Approximation algorithms for NP-hard problems*. Thomson, 1996.

[44] HOROWITZ, E., AND SAHNI, S. *Fundamentals of Computer Algorithms*. Computer Science Press, 1978.

[45] HROMKOVIČ, J. *Algorithms for Hard Problems*. Springer Verlag, 2001.

[46] JANSEN, K. Scheduling malleable parallel tasks: An asymptotic fully polynomial-time approximation scheme. In *Proceedings 10th European Symposium on Algorithms (ESA)* (2002).

[47] JANSEN, K. Approximation algorithms for the general max-min resource sharing problem: faster and simpler. In *Proceedings 9th Scandinavian Workshop on Algorithm Theory (SWAT)* (2004).

[48] JANSEN, K. Scheduling malleable parallel tasks: An asymptotic fully polynomial-time approximation scheme. *Algorithmica 39* (January 2004), 59–81.

[49] JANSEN, K., AND VAN STEE, R. On strip packing with rotations. In *to appear in Proceedings 37th ACM Symposium on Theory of Computing (STOC)* (2005).

[50] JANSEN, K., AND ZHANG, G. Maximizing the number of packed rectangles. In *Proceedings 9th Scandinavian Workshop on Algorithm Theory (SWAT)* (2004), pp. 362–371.

[51] JANSEN, K., AND ZHANG, G. On rectangle packing: maximizing benefits. In *Proceedings 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2004), pp. 197–206.

[52] JOHNSON, D. S. Approximation algorithms for combinatorial problems. *Journal of Computational System Science 9* (1974), 256–278.

[53] KARMARKAR, N., AND KARP, R. M. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings 23rd Annual Symposium on Foundations of Computer Science(FOCS)* (1982), pp. 312–320.

[54] KARP, R. M. Reducibility among combinatorial problems. In *Proceedings of the Symposium on the Complexity of Computer Computations* (York-town Heights, NY, March 1972), Plenum Press, New York, pp. 85–103.

[55] KELLERER, H., PFERSCHY, U., AND PISINGER, D. *Knapsack problems*. Springer, 2004.

[56] KENYON, C., AND RÉMILA, E. Approximate strip-packing. In *Proceedings 37th Annual Symposium on Foundations of Computer Science (FOCS)* (1996), pp. 31–36.

[57] KENYON, C., AND RÉMILA, E. A near-optimal solution to a two dimentional cutting stock problem. *Mathematics of Operations Research 25* (2000), 645–656.

[58] KHANNA, S., MOTWANI, R., SUDAN, M., AND VAZIRANI, U. On syntactic versus computational views of approximability. In *Proceedings 35th Annual Symposium on Foundations of Computer Science* (Santa Fe, New Mexico, 20-22 November 1994), IEEE, pp. 819–830.

[59] KLEITMAN, D., AND KRIEGER, M. An optimal bound for two dimensional bin packing. In *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (1975), pp. 163–168.

[60] LAWLER, E. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research 4* (1979), 339–356.

[61] LENSTRA, J. K., AND KAN, A. H. G. R. Complexity of scheduling under precedence constraints. *Operations Research 26* (1978), 22–35.

[62] LEUNG, J.-T., TAM, T., WONG, C., YOUNG, G., AND CHIN, F. Packing squares into a square. *Journal of Parallel and Distributed Computing 10* (1990), 271–275.

[63] LI, K., AND CHENG, K. Complexity of resource allocation and job scheduling problems in partitionable mesh connected systems. In *Proceedings 1st Annual IEEE Symposium of Parallel and Distributed Processing* (1989), IEEE Computer Society, Silver Spring, MD, pp. 358–365.

[64] MAYR, E. W., PRÓMEL, H. J., AND STEGER, A., Eds. *Lectures on proof verification and approximation algorithms*. LNCS 1367. Springer Verlag, 1998.

[65] MEIR, A., AND MOSER, L. On packing of squares and cubes. *Journal of Combinatorial Theory 5* (1968), 126–134.

[66] MOON, J., AND MOSER, L. Some packing and covering theorems. *Colloquium Mathematicum 17* (1967), 103–110.

[67] MOSER, L. Poorly formulated unsolved problems of combinatorial geometry. Mimeographed, 1965.

[68] MOSER, W., AND PACH, J. Problem N.108. In *Research Problems in Discrete Geometry*. McGill University, Montreal, 1989.

[69] NEMHAUSER, G. L., AND WOLSEY, L. A. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York, 1988.

[70] NOVOTNY, P. A note on packing of squares. In *Studies of University of Transport and Communications in Zilina*, no. 10 in Math.-Phys. series. 1995.

[71] NOVOTNY, P. On packing of squares into a rectangle. *Archivum Mathematicum 32* (1996), 75–83.

[72] PADBERG, M. *Linear Optimization and Extensions*. Springer Verlag, 1999.

[73] PAPADIMITRIOU, C. H. *Computational Complexity*. Addison Wesley, 1994.

[74] PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial optimization: Algorithms and complexity*. Prentice-Hall, 1982.

[75] PAPADIMITRIOU, C. H., AND YANNAKAKIS, M. Optimization, approximation, and complexity classes. *Journal of Computer and System Science 43* (1991), 425–440.

[76] PLOTKIN, S. A., SHMOYS, D. B., AND TARDOS, E. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research 20(2)* (1995), 257–301.

[77] SAHNI, S., AND GONZALEZ, T. F. P-complete approximation problems. *Journal of the ACM 23* (1976), 555–565.

[78] SAIGAL, R. *Linear programming: A modern integrated analysis*. Kluwer Academic Publishers, 1995.

[79] SCHIERMEYER, I. Reverse fit : a 2-optimal algorithm for packing rectangles. In *Proceedings 2nd European Symposium on Algorithms (ESA)* (1994), pp. 290–299.

[80] SCHRIJVER, A. *Theory of Linear and Integer Programming*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley and Sons, Inc, 1986.

[81] SEIDEN, S., AND VAN STEE, R. New bounds for multi-dimentional packing. *Algorithmica 36(3)* (2003), 261–293.

[82] SEVASTIANOV, S., AND WOEGINGER, G. Makespan minimization in open shops: a polynomial time approximation scheme. *Mathematical Programming 82* (1998), 191–198.

[83] SLEATOR, D. A 2.5 times optimal algorithm for bin packing in two dimensions. *Information Processing Letters 10* (1980), 37–40.

[84] STEINBERG, A. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing 26(2)* (1997), 401–409.

[85] VAZIRANI, V. V. *Approximation Algorithms*. Springer Verlag, 2001.

Index

CONCLUSIONS

In this thesis we address such 2-dimensional packing problems as strip packing, bin packing and storage packing. These problems play an important role in many application areas, e.g. cutting stock, VLSI design, image processing, and multiprocessor scheduling.

The larger part of the work is devoted to the storage packing problem, that is the problem of packing weighted rectangles into a single rectangle so as to maximize the total weight of the packed rectangles. Despite the practical importance of the problem, there are just a few known results in the literature. The main objective was to fill this gap and also to build the bridges to already known algorithmic solutions for strip packing and bin packing problems. This was successfully achieved. Considering natural relaxations of the storage packing problem we proposed a number of efficient algorithms which are able to find solutions within a factor of $(1-\varepsilon)$OPT. We have used the approach of Grigoriadis et.al. [40] for the case of packing with large resources (see Section 4.4 and Appendix C), that can lead to further practical algorithms.

Still, our work on the storage packing problem was primarily motivated by some theoretical questions which have been open for a number of years. In the first chapter we present a PTAS for the special case of the problem where a set of weighted squares is packed into a unit size square frame, when square's weights are equal to their areas. In other words, we are interested in covering the maximum area of a unit square frame by squares, and we try to generalize this result for the d-dimensional case.

In the second chapter, we address the problem of packing rectangles with weights into a unit size square region so as to maximize the total weight of the packed rectangles. We consider the so-called resource augmentation version of the general storage packing problem. That is, we allow the length of the unit square region, where the rectangles are to be packed, to be increased by some small value. We derive an algorithm which finds a packing of a subset of rectangles within a slightly augmented unit square frame with a weight at least $(1-\varepsilon)$ times the optimum. In other words we present a PTAS with resource augmentation. We also address the special case of the problem, when all rectangles to be packed are squares, and we give some ideas about how to generalize this result for the d-dimensional case.

Next, in the third chapter, we address the problem of packing weighted rectangles into a rectangle and consider the so-called case of large resources, where the number of packed rectangles is relatively large. We present an algorithm, which finds a packing of a sublist of rectangles within a given dedicated rectangle of total weight at least $(1-\varepsilon)$OPT, where OPT is the optimum weight. The running time of the algorithm is polynomial in the number of rectangles. In Chapter 4 we continue our work on this version. By using new techniques we improve the algorithm to be polynomial in both the number of rectangles and $1/\varepsilon$. In other words we derive a fully polynomial time approximation scheme (FPTAS) with large resources. Here, as an application of our algorithm, we provide a $(\frac{1}{2}-\varepsilon)$-approximation algorithm for the advertisement placement problem for newspapers and the Internet, which can be formulated as the problem of packing weighted rectangles into k identical rectangular bins so as to maximize the total weight of the packed rectangles. The running time of the algorithm is polynomial in the number of rectangles and $1/\varepsilon$.

Finally, in the last chapter we address the strip packing problem with rotations by 90 degrees. In this problem a set of rectangles is packed into a vertical strip of unit width so that the height to which the strip is filled is minimized. We present an asymptotic fully polynomial time approximation scheme (AFPTAS), which gives a positive answer to an open theoretical problem in [19]. We develop new techniques which allow us to use the known algorithm for the strip packing problem without rotations [56, 57]. So, this closes the gap between classical statement of the problem and its extension.

In spite of the fact that significant progress has been achieved, there are still a number of interesting theoretical questions which remain open. One of such open questions is the existence of an algorithm, which would find in time polynomial in the number of rectangles and $1/\varepsilon$, $(1-\varepsilon)$OPT solutions for the storage packing problem without resource augmentation, large resources or any other conditions on resources. We conjecture that this can be done, indeed.

Curriculum Vitae

February 17, 1978:	born in Yuzhno-Sakhalinsk, Russia.
1985-1995 :	student at Secondary School No.23, Yuzhno-Sakhalinsk, Russia.
1995 - 1999 :	bachelor student at the Technical Department, Institute of Economics, Law and Informatics, Yuzhno-Sakhalinsk, Russia.
June 1999:	Bachelor’s degree in Mathematics (with honors).
1999 - 2001 :	master student at Department of Mechanics and Mathematics, Novosibirsk State University, Novosibirsk, Russia.
June 2001:	Master’s degree in Computer Science and Applied Mathematics (with honors).
July 2001 : **Aug. 2002**	post-graduate at Sobolev Institute of Mathematics, Novosibirsk State University, Novosibirsk, Russia.
since Sept. 2002:	PhD student at Graduiertenkolleg 357 “Effiziente Algorithmen und Mehrskalenmethoden”, Institute for Computer Science and Applied Mathematics, Christian-Albrechts-University of Kiel, Germany.