Faster approximation schemes for the two-dimensional knapsack problem

Sandy Heydrich* Andreas Wiese†

Abstract

An important question in theoretical computer science is to determine the best possible running time for solving a problem at hand. For geometric optimization problems, we often understand their complexity on a rough scale, but not very well on a finer scale. One such example is the two-dimensional knapsack problem for squares. There is a polynomial time $(1+\epsilon)$-approximation algorithm for it (i.e., a PTAS) but the running time of this algorithm is triple exponential in $1/\epsilon$, i.e., $\Omega(n^{2^{2^{1/\epsilon}}})$. A double or triple exponential dependence on $1/\epsilon$ is inherent in how this and several other algorithms for other geometric problems work. In this paper, we present an EPTAS for knapsack for squares, i.e., a $(1+\epsilon)$-approximation algorithm with a running time of $O_\epsilon(1) \cdot n^{O(1)}$. In particular, the exponent of n in the running time does not depend on ϵ at all! Since there can be no FPTAS for the problem (unless $\mathsf{P} = \mathsf{NP}$) this is the best kind of approximation scheme we can hope for. To achieve this improvement, we introduce two new key ideas: We present a fast method to guess the $\Omega(2^{2^{1/\epsilon}})$ relatively large squares of a suitable near-optimal packing instead of using brute-force enumeration. Secondly, we introduce an *indirect guessing* framework to define sizes of cells for the remaining squares. In the previous PTAS each of these steps needs a running time of $\Omega(n^{2^{2^{1/\epsilon}}})$ and we improve both to $O_\epsilon(1) \cdot n^{O(1)}$.

We complete our result by giving an algorithm for two-dimensional knapsack for rectangles under $(1+\epsilon)$-resource augmentation. In this setting, we also improve the best known running time of $\Omega(n^{1/\epsilon^{1/\epsilon}})$ to $O_\epsilon(1) \cdot n^{O(1)}$ and compute even a solution with optimal profit, in contrast to the best previously known polynomial time algorithm for this setting that computes only an approximation. We believe that our new techniques have the potential to be useful for other settings as well.

1 Introduction.

There are many well-studied geometric problems in combinatorial optimization, for instance the natural geometric generalizations of fundamental one-dimensional problems like Knapsack or Bin Packing. For many settings of those, there are polynomial time algorithms known that give an approximation guarantee of $1+\epsilon$, e.g., [13, 4, 9]. A common theme in these PTASs is the shifting technique: items that need to be packed are classified into large and small squares such that intuitively the large squares are much larger than the small ones. This classification is done such that the "medium" squares inbetween can be neglected since, e.g., they give very little profit in the optimal solution. However, the relative size of the large squares can then be very small, e.g., $\epsilon^{1/\epsilon}$, and since most algorithms enumerate them, this leads to a running time whose exponent of n is exponential in $1/\epsilon$, i.e., $\Omega(n^{1/\epsilon^{1/\epsilon}})$. This raises the question whether this highly impractical running time is necessary or whether better running times are possible of e.g., $n^{O(1/\epsilon)}$ or even $O_\epsilon(1) \cdot n^{O(1)}$ (where $O_\epsilon(1)$ denotes a value that is constant for constant ϵ). Recently, there has been a lot of research on determining the best possible running times for specific problems, see e.g., [6, 1] and references therein. For some problems, we know even the best possible exponent of n under standard complexity assumptions. For PTASs in geometric settings we are far from that: the first question to answer is whether an exponential dependence on $1/\epsilon$ of this exponent is inherently necessary.

In this paper, we study this question for the two-dimensional knapsack for squares: we are given a collection I of n axis-parallel squares $i = 1, \dots, n$, specified by their side length $s_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$, and a knapsack of size $N \times N$ for some value $N \in \mathbb{N}$. We want to compute a subset $I' \subseteq I$ and a packing for I', i.e., assign a position to each square $i \in I'$ inside the knapsack such that the packed squares are pairwise non-overlapping. The objective is to maximize the total profit $p(I') := \sum_{i \in I'} p_i$. In a slight abuse of notation, we use OPT to denote the optimal solution as well as the profit of the optimal solution. The best known polynomial time approximation algorithm for the problem has an approximation ratio of $1+\epsilon$ [13] and a running time of $\Omega(n^{2^{2^{1/\epsilon}}})$.

In this paper, we prove that for the two-dimensional knapsack problem for squares it is *not* necessary that the exponent of n depends exponentially on $1/\epsilon$. Even more, we show that the exponent of n *does not need*

*Max Planck Institute for Informatics and Graduate School of Computer Science, Saarland Informatics Campus, Germany. Supported in part by the *Google European Fellowship in Market Algorithms*. heydrich@mpi-inf.mpg.de

†University of Chile, Santiago de Chile, Chile. This research was carried out while the author was at the Max Planck Institute for Informatics. awiese@mpi-inf.mpg.de

to depend on ϵ at all. Our main result is a $(1+\epsilon)$-approximation algorithm that runs in time $O_\epsilon(1)\cdot n^{O(1)}$, i.e., an EPTAS.

1.1 Our techniques. The PTAS in [13] for geometric knapsack for squares (and many other algorithms for geometric problems, e.g., [2, 16, 3]) applies the mentioned shifting technique to distinguish squares into large and small squares. Eventually, it is shown that there is a $(1+\epsilon)$-approximative packing which is structured into $O_\epsilon(1)$ large squares and $O_\epsilon(1)$ cells. However, due to the shifting step both quantities can be $\Omega(2^{2^{1/\epsilon}})$. Roughly speaking, the PTAS in [13] then guesses the large squares and the sizes of the cells in time $O(n^{2^{2^{1/\epsilon}}})$ each. Then it assigns the small squares to the cells via an instance of the generalized assignment problem [20].

Guessing large squares fast. Instead, we would like to determine the large squares more quickly. However, since there can be up to $\Omega(2^{2^{1/\epsilon}})$ of them we do not want to simply enumerate over them since there are $n^{\Omega(2^{2^{1/\epsilon}})}$ possibilities. Instead, we show that by losing a factor $1+\epsilon$ we can assume that the squares have only $O_\epsilon(\log n)$ different profits and that each profit class with large squares in the solution contributes at most $O_\epsilon(1)$ squares to the solution in total. Then, we can show that there are only $O_\epsilon(\log n)$ squares in total that can potentially be large. This reduces the guessing time to $(\log n)^{O_\epsilon(1)} = O_\epsilon(1)\cdot n^{O(1)}$. More generally, with our approach we can obtain such an improved bound in any setting with only few (profit) classes of input squares where for each class there are only few relevant squares if we know that the class contributes only a small number of squares to a (near-)optimal solution.

Guessing cell decomposition via indirect guessing. Roughly speaking, there are two types of cells in the decomposition given by [13]. The first type are *block cells* that contain only squares that are much smaller than the cell itself (in both dimensions). For those we show that by losing a factor of $1+\epsilon$ we can round down their heights and widths to powers of $1+\epsilon$. This allows us to reduce the number of possibilities for those quantities to $O_\epsilon(\log n)$ and thus to $(\log n)^{O_\epsilon(1)}$ for all cells in parallel.

The other cells are further divided and the most complicated parts are *row subframes.* Each of them has the property that its height equals the height of some input square and inside it squares are lined up next to each other but never on top of each other (see Figure 1). Moreover, for the height of the row subframes there are only $O_\epsilon(1)$ (unknown) values $k_1, k_2, \ldots$ arising in the whole instance (as they stem from a harmonic grouping step). If we knew these values k_j then we would be essentially done. The PTAS in [13] just guesses them directly and *afterwards* assigns all squares into all subframes and all block cells. We cannot afford to guess all values k_j directly since a priori there are n possibilities for each of them and we cannot afford $n^{O_\epsilon(1)}$ guesses. Instead, we guess them *indirectly.* Intuitively, for a given value of x we ask ourselves: how much profit would we obtain from subframes of width k_1 if k_1 was equal to x? The larger x is, the more profit we could achieve since then more input squares would fit into these subframes. By losing a factor of $1+\epsilon$ in the objective it suffices to allow only values of x where this profit increases by a factor of $1+\epsilon$. This allows us to iteratively guess the values k_j such that for each of them there are only $O_\epsilon(\log n)$ possibilities. In particular, this interlaces the guessing of the subframe widths with the packing of the squares, instead of doing one after the other like in [13].

THEOREM 1.1. *There is an EPTAS for the two-dimensional knapsack problem for squares.*

We will present this result in Sections 2–5.

Guessing slots for large rectangles. For our second result we consider the setting where we can increase the size of the knapsack in both dimensions by a factor $1+\epsilon$ (resource augmentation) but we allow the input objects to be axis-parallel rectangles, not only squares. For this case, we improve the running time of the known $(1+\epsilon)$-approximation [9] from $\Omega(n^{1/\epsilon^{1/\epsilon}})$ to $O_\epsilon(1)\cdot n^{O(1)}$ and even compute a solution with *optimal* profit, rather than an approximation. The key idea here is that we do not guess the large rectangles directly (which would require a running time of $\Omega(n^{1/\epsilon^{1/\epsilon}})$) but we round their sizes and only guess their position in a suitable grid of complexity $O_\epsilon(1)$. Then, we assign the rectangles into the resulting slots in a greedy manner. The other rectangles are assigned via LP-rounding. In this step the algorithm in [9] sacrifices a factor $1+\epsilon$ in the approximation ratio. Instead, we increase the size of the knapsack slightly and thus do not lose anything in the approximation ratio.

THEOREM 1.2. *There is an algorithm for the two-dimensional knapsack problem for rectangles under $(1+\epsilon)$-resource augmentation with and without rotation with running time $O_\epsilon(1)\cdot n^{O(1)}$. This algorithm computes a solution with profit at least the profit of the optimal solution (that does not use resource augmentation).*

We will present this result in Section 6.

1.2 Other related work. For two-dimensional geometric knapsack for axis-parallel rectangles, the best known polynomial time result is a $(2+\epsilon)$-approximation

algorithm [16, 15]. However, there are PTASs for the resource augmentation setting in both [9] or in only one [12] dimension, and for the case that the profit of each item equals its area [3]. The running time of all these $(1+\epsilon)$-approximation algorithms is $\Omega(n^{1/\epsilon^{1/\epsilon}})$. Moreover, for quasi-polynomially bounded input there is even a $(1+\epsilon)$-approximation in quasi-polynomial time [2].

Closely related to the geometric knapsack problem is geometric bin packing. For the multi-dimensional case of packing hypercubes, there is an APTAS [4], and it is also shown that there exists an algorithm that packs (two-dimensional) rectangles into the optimal number of bins using resource augmentation. For the case of rectangles the currently best known result is a $(1.405+\epsilon)$-approximation [5]. Another related problem is strip packing, where squares need to be packed into a strip of width 1 such that the height of the packing is minimized. For this problem, an absolute approximation guarantee of $5/3+\epsilon$ can be achieved, while a lower bound of $3/2$ is known. [10]. Both of these results use the doubly-exponential algorithm in [3] as a subroutine. Moreover, for strip packing there is an asymptotic FPTAS [17] and a $(1.4+\epsilon)$-approximation algorithm in (doubly-exponential) pseudo-polynomial time.

2 Cell decomposition.

Let $\epsilon > 0$. Our algorithm is based on a structural result by Jansen and Solis-Oba [13]. They show that there is a near-optimal solution that packs its squares in a *structured packing*. Parts of this packing are done via the algorithm due to Kenyon and Remila [17] and we refer to the resulting type of packing as a KR-packing, see Figure 1 for a sketch. Intuitively, when packing squares into a rectangular region that has much larger width than height, the KR-algorithm partitions this region into frames (vertical slots within the region), and each frame into subframes (horizontal slots within the frame). There are two types of such subframes: row subframes, which contain squares of roughly the same size; and block subframes, which contain squares that are very small compared to the size of the subframe. In addition, for the heights of the row subframes, there are only few possible values, and within each row subframe, squares are only packed in one row from left to right (i.e. any vertical line going through a row subframe intersects at most one square).

Definition 2.1. (KR-packing) *Consider a rectangular area C with height h_C and width w_C and a set of squares S. Assume w.l.o.g. that $h_C \leq w_C$. We say that the squares in S are packed into C in a* KR-packing *if*

- *there are values $k_0, ..., k_{1/\epsilon^4} \in \mathbb{R}$ with $k_0 = \epsilon^4 \cdot h_C \leq k_1 \leq \ldots \leq k_{1/\epsilon^4}$*
- *C is partitioned (vertically) into at most $(\frac{2+\epsilon^2}{\epsilon^2})^2 = O(1/\epsilon^4)$ frames $F_{C,1}, F_{C,2}, ...$ of height h_C and widths $w_{C,1}, w_{C,2}, ...$,*
- *each frame $F_{C,\ell}$ is (horizontally) partitioned into at most $1/\epsilon^2$ subframes $F_{C,\ell,1}, F_{C,\ell,2}, ...$. For each such subframe $F_{C,\ell,j}$ denote by $w_{C,\ell,j}$ its width and by $h_{C,\ell,j}$ its height. We have that $w_{C,\ell,j} \geq \epsilon^2 \cdot h_C$. For a subframe $F_{C,\ell,j}$ we have that either*
  1. *there is an index $f(C,\ell,j) \in \{1, ..., 1/\epsilon^4\}$ such that $h_{C,\ell,j} = k_{f(C,\ell,j)}$, each vertical line crossing $F_{C,\ell,j}$ intersects at most one square i packed in $F_{C,\ell,j}$, and for this square i we have that $s_i \in (k_{f(C,\ell,j)-1}, k_{f(C,\ell,j)}]$ and if $F_{C,\ell,j}$ contains $\bar{w}_{C,\ell,j}$ such squares then $w_{C,\ell,j} \geq \bar{w}_{C,\ell,j} \cdot k_{f(C,\ell,j)}$ (a* row *subframe or a* row *for short) or*
  2. *it contains only squares i with $s_i \leq \epsilon^4 \cdot h_C$ and $h_{C,\ell,j} \geq \epsilon^2 \cdot h_C$ (a* block *subframe or a* block *for short) or*
  3. *it does not contain any squares (an* empty *subframe).*
- *each frame $F_{C,\ell}$ has at most one subframe which is a block or empty.*

We define KR-packings for areas C with $h_C > w_C$ in a similar manner. Note that all squares i with $s_i > \epsilon^4 \cdot h_C$ are packed into row subframes and all squares i' with $s_{i'} \leq \epsilon^4 \cdot h_C$ are packed into block subframes.

Lemma 2.1. (Structured packing [13]) *For any instance there exists a solution consisting of a set of squares $I' \subseteq I$ such that $p(I') \geq (1 - O(\epsilon))\mathrm{OPT}$ and there is a packing for I' with the following properties:*

- *there is a subset $I'_L \subseteq I'$ of squares with $|I'_L| \leq F(\epsilon)$ that are labeled as* big squares*, where $F(\epsilon) = (1/\epsilon)^{2^{O(1/\epsilon^4)}}$,*
- *the space not occupied by the big squares can be partitioned into at most $F(\epsilon)$ rectangular cells $\mathcal{C}$,*
- *each square in $I'_S := I' \setminus I'_L$ is packed into a cell such that it does not overlap any cell boundary,*
- *each cell in $\mathcal{C}$ is either labeled as a* block *cell or as an* elongated *cell. Denote its height by h_C and its width by w_C.*
  - *For each block cell $C \in \mathcal{C}$ we have that each square $i \in I'_S$ packed into C satisfies $s_i \leq \epsilon^2 \cdot \min\{h_C, w_C\}$.*
  - *For each elongated cell $C \in \mathcal{C}$ we have that the squares in it are packed in a KR packing.*

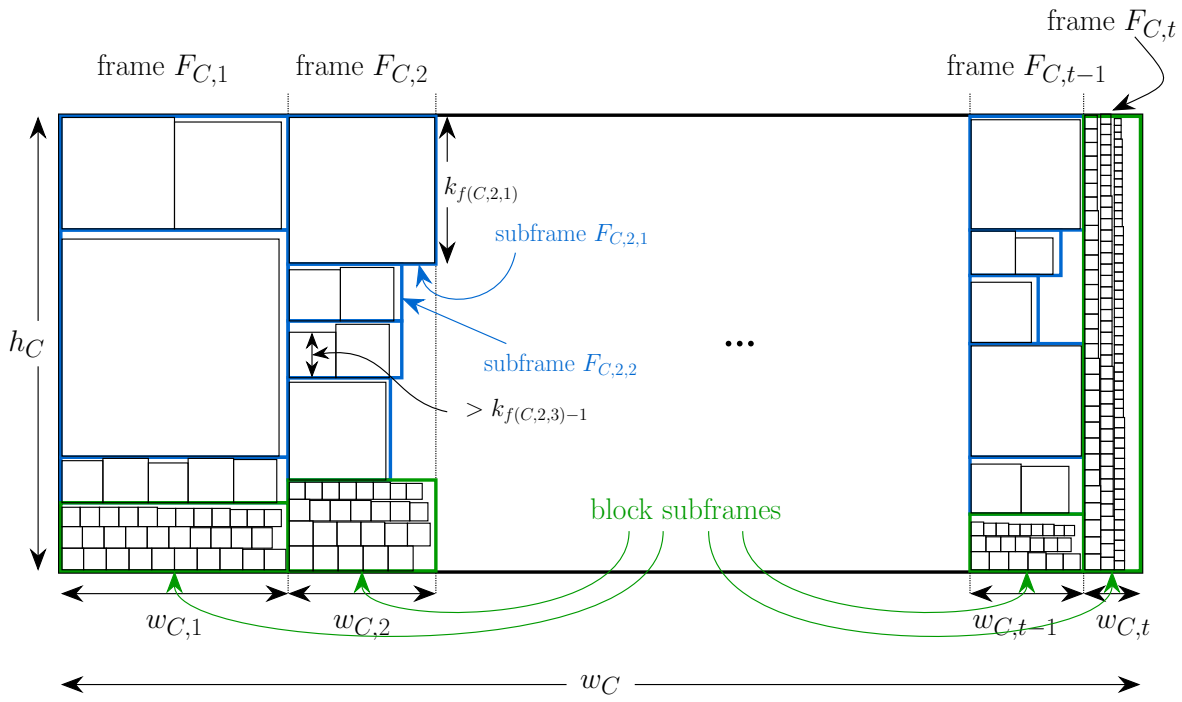


Figure 1: Illustration of a KR-packing with t frames.

Unfortunately, $F(\epsilon)$ can be as large as $(1/\epsilon)^{2^{O(1/\epsilon^4)}}$. Using Lemma 2.1 one can obtain a PTAS with a running time of $n^{O(F(\epsilon))} = n^{O_\epsilon(1)}$: first, we guess the at most $F(\epsilon)$ large squares in time $n^{O(F(\epsilon))}$. Secondly, we guess the sizes of the cells in $\mathcal{C}$. This can be done directly in (pseudopolynomial) time $(n \cdot \max_i s_i)^{O(F(\epsilon))}$ or in time $n^{O(F(\epsilon))}$ by exploiting some further properties of the decomposition (not stated above, see [13] for details). Then we compute a packing for the guessed large squares and the cells. It remains to select some of the remaining squares and to pack them into the cells. To this end, we guess the heights and widths of all frames and subframes in each elongated cell (again, either directly in pseudo-polynomial time or in polynomial time using some further properties of the decomposition). This splits each elongated block into a set of rows and a set of blocks. It remains to pack squares such that

- into each block cell or block subframe C we pack only squares $i \in I$ with $s_i \leq \epsilon^2 \cdot \min\{h_C, w_C\}$ such that the total area of the assigned squares does not exceed the area of C
- into each row subframe C we pack only squares i with $s_i \leq h_C$. Also, we ensure that if C is contained in a elongated cell C' with $h_{C'} \leq w_{C'}$ then each vertical line intersects at most one square packed into C. (If $h_{C'} \geq w_{C'}$, the same holds for vertical lines.)

This yields an instance of the generalized assignment problem (GAP) [20] with a constant number of containers (or machines as stated in the terminology of [20]). In particular, we can compute a $(1+\epsilon)$-approximation for this instance in time $n^{O_\epsilon(1)}$. One can show that the squares assigned to a block cell/subframe can be packed into them in a greedy manner, while losing only a factor of $1+\epsilon$ in the objective. This yields a PTAS for the overall problem.

There are two crucial steps in which this algorithm needs a running time of $n^{O_\epsilon(1)}$:

- guessing the large squares
- guessing the sizes of all cells and all subframes

In the next two sections, we explain how we address these two problems. In Section 3 we present a technique that allows us to guess the large squares in time $O_\epsilon(1) \cdot n^{O(1)}$. Then in Section 4 we present an EPTAS for the special case that $N \gg N'$, i.e., the knapsack is a rectangle with very large aspect ratio. In that case, there is a near-optimal packing with no large squares and only one cell that is elongated. This case will illustrate our techniques to address the second issue: to guess the sizes of all cells and frames. As we will see, this step will be interlaced with the selection of the squares and the computation of their packing. Building

on these techniques, in Section 5 we present our EPTAS for the general case.

3 Guessing the large squares faster.

First, by losing at most a factor of $1+O(\epsilon)$ we round the profits of the squares to powers of $1+\epsilon$ in the range $[1, n/\epsilon)$.

LEMMA 3.1. *We can modify the input such that there are only $k^* = O(\frac{\log n}{\log(1+\epsilon)})$ different square profits while losing at most a factor of $1+O(\epsilon)$.*

Proof. In a first step, we rescale the profits of the squares such that the square with largest profit has profit 1 and all profits lie in the interval $[0,1]$. We now know that the optimal solution for this rescaled instance has profit at least 1. Therefore, we lose at most a factor $1+\epsilon$ if we remove all squares from the input that have profit less than ϵ/n, as these squares can never contribute more than ϵOPT to the solution. Finally, we multiply each profit by n/ϵ and hence end up with an instance where square profits are in the range $[1, n/\epsilon]$. Losing another factor of $1+\epsilon$, we round down the profit of each square to the next smaller powers of $1+\epsilon$. As a result, we obtain $k^* = \log_{1+\epsilon}(n/\epsilon) = O(\frac{\log n}{\log(1+\epsilon)})$ different values for the square profits. □

As a result, we obtain k^* different values for the square profits and we define $I^{(k)} := \{i \in I | p_i = (1+\epsilon)^k\}$ for each $k = 0, ..., k^*$. This is a standard step that can be applied to any packing problem where we are guaranteed that $|OPT| \leq n$.

We would like to guess the large squares I'_L. Unfortunately, $|I'_L|$ can be as large as $F(\epsilon) = (1/\epsilon)^{2^{O(1/\epsilon^4)}}$ so we cannot afford to enumerate all possibilities for I'_L. Also, for an square $i \in I$ it is a priori not even clear whether it appears as a large or small square in I' (some square in I'_L might even be smaller than some square in I'_S). To address this, first we guess $B := |I'_L|$; there are only $O_\epsilon(1)$ possibilities. Then we do the following transformation of I'. If for a profit class $I^{(k)}$ we have that $|I' \cap I^{(k)}| \geq B/\epsilon$ then we remove all squares in $I'_L \cap I^{(k)}$ from I'.

LEMMA 3.2. *Suppose that $|I' \cap I^{(k)}| \geq B/\epsilon$. Then $p(I'_L \cap I^{(k)}) \leq \epsilon \cdot p(I' \cap I^{(k)})$.*

Proof. All squares in $I^{(k)}$ have the same profit and we have that $|I'_L| = B$, hence $|I'_L \cap I^{(k)}| \leq B$. Therefore, $|I'_S \cap I^{(k)}| \geq \frac{B(1-\epsilon)}{\epsilon}$ and $\frac{p(I'_L \cap I^{(k)})}{p(I' \cap I^{(k)})} \leq \frac{B}{B + \frac{B(1-\epsilon)}{\epsilon}} = \epsilon$. □

Denote by I'' the resulting solution and let $I''_L := I'_L \cap (\bigcup_{k:|I' \cap I^{(k)}| < B/\epsilon} I^{(k)})$. In I'' each profit class contributes at most B/ϵ squares in total or no large squares. This is useful since we can guess now the large squares in I'' fast. If a class $I^{(k)}$ contributes n_k squares to the solution we can assume w.l.o.g. that those are the n_k smallest squares of this class (since they are squares). Therefore, let $\tilde{I}$ denote the union of the B/ϵ smallest squares from each class $I^{(k)}$. We know that $|\tilde{I}| \leq O(\frac{\log n}{\log(1+\epsilon)}) \cdot B/\epsilon$ and that $I''_L \subseteq \tilde{I}$. We can now guess the correct choice of the at most B squares in I''_L in time $\binom{|\tilde{I}|}{B} \leq (\log_{1+\epsilon}(n/\epsilon) \cdot B/\epsilon)^B = (\frac{\log n}{\log(1+\epsilon)})^{F(\epsilon)} \leq (\frac{1}{\log(1+\epsilon)})^{O(F(\epsilon))} \cdot n$.

LEMMA 3.3. *There are only $O_\epsilon(1) \cdot n$ possibilities for the large squares I''_L in I''.*

4 Special case: only one elongated cell.

Suppose that the input knapsack has size $N \times N'$ with $N \geq \frac{1}{\epsilon^4} N'$. For such an instance, there exists a $(1+\epsilon)$-approximative solution which is a KR-packing (i.e., behaves like a single elongated cell) and there are no big squares [13].

From this packing, we first guess the number of frames and the number of subframes of each of them. There are $1/\epsilon^{O(1/\epsilon^4)}$ possibilities. For each subframe we guess whether it is a row subframe, a block subframe, or an empty subframe.

4.1 Guessing the block sizes.

Then we guess the heights and widths of all block subframes, or *blocks* for short. This needs some preparation. Denote by K the number of blocks (which is implied by our previous guesses). We have that $K \leq O(1/\epsilon^4)$ as each frame can contain at most one block subframe. Let α be the area of the block with largest area. We first guess an estimate of α. Intuitively, for doing this we use that α is by at most a factor $O(n)$ larger than the area of the largest square contained in the largest block.

LEMMA 4.1. *We can compute a set of $T \leq \frac{n \log n}{\log(1+\epsilon)}$ values $\alpha_1, ..., \alpha_T$ such that there exists an $i \in \{1, ..., T\}$ such that $\alpha_i \leq \alpha \leq (1+\epsilon)\alpha_i$. Define $\tilde{\alpha} := \alpha_i$.*

Proof. Denote by $\hat{s}$ the size of the largest square packed in a block in the near-optimal solution we are considering. We have that $\hat{s}^2 \leq \alpha \leq n\hat{s}^2$. Therefore, consider the set $\{(1+\epsilon)^j s_i^2 | \forall i, \forall j \in \{0, \dots, \log_{1+\epsilon} n\}\}$. It contains $\log_{1+\epsilon} n$ values between s_i^2 and ns_i^2 for every square size s_i, hence at most $n \log_{1+\epsilon} n$ values in total. As the values are by at most a factor $1+\epsilon$ apart, the claim follows by choosing α_i to be the largest value in this set below α. □

We guess $\tilde{\alpha}$ in time $n \log n / \log(1+\epsilon)$.

We will now prove the following lemma, which we will use at various places:

LEMMA 4.2. *Let R be a rectangle of side lengths a, b and S a set of squares of total profit $p(S)$ such that each of the squares in S has side length at most $\epsilon \min\{a, b\}$ and the total area of S is $a(S) \leq ab$. Then we can pack a subset $S' \subseteq S$ with total profit $p(S') \geq (1 - O(\epsilon))p(S)$ into R.*

Proof. We use the following lemma from Jansen and Solis-Oba:

LEMMA 4.3. ([14]) *Let S be a set of squares each of size at most δ and let R be a rectangular bin of length a and width b. NFDH either packs in R all squares from S, or it packs squares of total area at least $(b - 2\delta) \times (a - \delta)$.*

Each square has area at most $\epsilon^2 \min\{a, b\}^2 \leq \epsilon^2 ab$. If $a(S) \leq ab(1 - 2\epsilon)^2$, then $a(S) \leq a(1 - \epsilon)b(1 - 2\epsilon) \leq (a - \epsilon)(b - 2\epsilon)$ and hence by Lemma 4.3 we can pack all squares from S into the rectangle R. Therefore, assume that $a(S) > ab(1 - 2\epsilon)^2$, and try to find a subset of squares from S that has small profit (at most $O(\epsilon)p(S)$) such that when we remove these squares from S, the total area drops below the bound from Lemma 4.3. We assign the squares to groups such that each group contains squares with a total area between $4\epsilon ab$ and $(4\epsilon + \epsilon^2)ab$ (we can achieve this by a simple greedy algorithm: assign squares into one group until the total area of squares in this group is at least $4\epsilon ab$; the upper bound follows from the upper bound on the square size). We get that $\frac{ab(1-2\epsilon)^2}{(4\epsilon+\epsilon^2)ab} = \Omega(1/\epsilon)$, and thus there is one group j whose squares S_j have a total profit of at most $O(\epsilon)p(S)$. The squares in this group have an area of at least $4\epsilon ab \geq (1 - (1 - 2\epsilon)^2)ab$ and hence the squares in $S \backslash S_j$ have total area at most $ab - (1 - (1 - 2\epsilon)^2)ab = (1 - 2\epsilon)^2 ab$ and we can pack them using Lemma 4.3. □

Next, we simplify the packing by removing blocks that are very small.

LEMMA 4.4. *By losing a factor $1 + O(\epsilon)$ we can remove all blocks with area less than $\tilde{\alpha}\frac{\epsilon}{KN}$.*

Proof. Consider the largest block (according to area) B, let w be its width and h its height and suppose that $w \geq h$. Define $1/\epsilon$ vertical strips of equal width in this cell, and let $p(S_i)$ ("the profit of strip S_i") denote the total profit of all squares intersecting the i-th strip. A square cannot intersect more than two strips, as the square size is at most ϵw and this is also the width of the strips. This means that $\sum_{i=1}^{1/\epsilon} p(S_i) \leq 2 \sum_{i \text{ packed in } B} p_i = 2p(B)$ (where $p(B)$ denotes the total profit of squares in block B). Consider the strip with smallest profit. It must have total profit $p(S_i) \leq 2\epsilon p(B)$. Hence, we can remove all squares touching the strip with lowest $p(S_i)$ at cost of at most an $O(\epsilon)$ fraction of the profit of all squares in this cell.

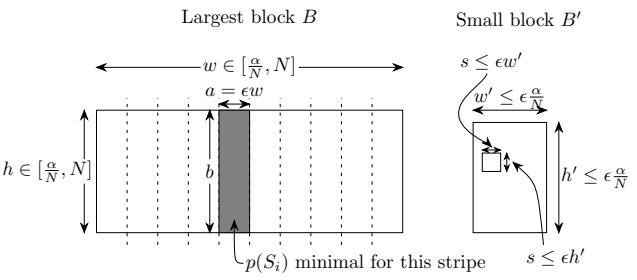


Figure 2: Illustration for proof of Lemma 4.4

The free strip has width $a := \epsilon w$ and height $b := h$ and thus an area of $\alpha^* := \epsilon hw = \epsilon\alpha$, and we now pack the squares of the small blocks (i.e. blocks with area less than $\epsilon\frac{\tilde{\alpha}}{KN}$) into this strip using NFDH. Let S be the set of these squares. The squares in S have a total area of at most $\epsilon\frac{\tilde{\alpha}}{N} \leq \epsilon\frac{\alpha}{N} = \alpha^*$.

Let now B' be one of the small blocks with width w' and height h', and let s be the size of a square in this block. We want to show that $s \leq \epsilon \min\{a, b\}$, and then argue that this gives us the desired result. We know from the definition of small blocks that $h'w' \leq \epsilon\frac{\alpha}{KN}$, and as $w', h' \geq 1$, we know that both h' and w' must be at most $\epsilon\frac{\alpha}{KN} \leq \epsilon\frac{\alpha}{N}$. We know $\min\{a, b\} \geq \epsilon\frac{\alpha}{N}$ (as again $\max\{a, b\} \leq N$ and $ab = \epsilon\alpha$) and $s \leq \epsilon w'$, and thus $s \leq \epsilon w' \leq \epsilon^2\frac{\alpha}{N} \leq \epsilon \min\{a, b\}$.

Now consider Lemma 4.2. In our case, the rectangular area in which we want to pack the squares from small blocks has side lengths $a = \epsilon w$ and $b = h$. These squares have size at most $\epsilon \min\{a, b\}$ and total area at most that of the free area, as established above. Hence, NFDH will pack a subset of S of profit at least $(1 - O(\epsilon))p(S)$ into the largest block. □

Using that our input squares are squares and that no cell can contain more than n squares we can prove the following lemma.

LEMMA 4.5. *For each block we can assume that $h_{C,\ell,j}, w_{C,\ell,j} \in [\sqrt{\frac{\epsilon\tilde{\alpha}}{nKN}}, \sqrt{(1 + \epsilon)\tilde{\alpha}n}]$.*

Proof. Consider a block B of height and width h, w. Let $\hat{s}_i$ be the size of the largest square in this block, then we have that $h, w \geq \hat{s}_i$ (as this square must fit in the box). We can assume that $h \leq n\hat{s}_i \leq nh$ and $w \leq n\hat{s}_i \leq nw$ (since at most n squares can be packed in this box and each of them has size at most $\hat{s}_i$ by definition of $\hat{s}_i$), and hence $w \leq nh$ and $h \leq nw \Leftrightarrow w \geq h/n$, which finally

gives $\frac{h^2}{n} \leq w \cdot h \leq (1+\epsilon)\tilde{\alpha} \Leftrightarrow h \leq \sqrt{(1+\epsilon)\tilde{\alpha}n}$ and $\epsilon\frac{\tilde{\alpha}}{KN} \leq wh \leq nh^2 \Leftrightarrow h \geq \sqrt{\tilde{\alpha}\frac{\epsilon}{nKN}}$ and therefore we get $w, h \in [\sqrt{\frac{\tilde{\alpha}\epsilon}{nKN}}, \sqrt{(1+\epsilon)\tilde{\alpha}n}]$. □

This ensures that there is a polynomial range for the sizes of the cells. Next, we show that we can round down the side lengths of the blocks.

LEMMA 4.6. *By losing a factor $1+O(\epsilon)$ we can round down the height and width of each block such that it becomes a power of $1+\epsilon$.*

Proof. We down round each block height and width to the nearest power of $1+\epsilon$. This results in $O(\log_{1+\epsilon} n)$ different values. Now, to show that we can still pack the squares into the blocks without losing too much profit, we do the following: Let B be some block of width w_B and height h_B and assume w.l.o.g. that $w_B \leq h_B$; denote by w'_B and h'_B the rounded side lengths of B. Assume $w'_B = (1+\epsilon)^j$ and $h'_B = (1+\epsilon)^i$, then we know that $w_B \leq (1+\epsilon)^{j+1}$ and $h_B \leq (1+\epsilon)^{i+1}$. We divide B into $\frac{1}{2\epsilon}$ many strips of equal width $2\epsilon h_B$. Each square touches at most two strips, so there is a strip s.t. all squares who touch this strip have total profit at most $O(\epsilon p_B)$ (where p_B is the total profit of squares in block B). We remove the squares from this strip and hence gain some free space of area $2\epsilon h_B w_B$. The squares touching the area that we lose due to shrinking have total area at most $(h_B - h'_B)w_B + (w_B - w'_B)h_B \leq ((1+\epsilon)^{i+1} - (1+\epsilon)^i)w_B + ((1+\epsilon)^{j+1} - (1+\epsilon)^j)h_B \leq \epsilon h'_B w_B + \epsilon w'_B h_B \leq 2\epsilon w_B h_B$. Hence, we can pack a subset of these squares into the free area using NFDH (see Lemma 4.2) and lose only an ϵ fraction of the profit. □

After applying Lemma 4.6 for each of the rounded values there are only $O\left(\frac{\log n}{\log(1+\epsilon)}\right)$ possibilities left. Thus, we can guess *all* these values in time $(\frac{\log n}{\log(1+\epsilon)})^{O(1/\epsilon^4)}$.

In the special case studied in this section all squares i with $s_i \leq \epsilon^4 \cdot w_C$ are packed into the blocks. We call them the *small squares*. All squares i with $s_i > \epsilon^4 \cdot w_C$ are packed into the rows, we call them the *row squares*. Each edge of each block is by a factor $\Omega(1/\epsilon^2)$ larger than the size of any small square by the definition of KR-packings. Thus, we can find a $(1+\epsilon)$-approximation for the small squares using greedy algorithms for selecting and packing the squares.

LEMMA 4.7. *There is a $(1+\epsilon)$-approximation algorithm with a running time of $n^{O(1)}$ for computing the most profitable set of small squares that can be packed into the blocks.*

Proof. According to Lemma 4.2, it suffices to select some set of small squares S that has area at most that of the blocks. The condition that squares are by a factor ϵ^2 smaller than the blocks is already established by the definition of KR-packings (the squares have $s_i \leq \epsilon^4 h_C$ while the width and height of the block are at least $\epsilon^2 h_C$). We do the selection of squares by solving a simple Knapsack problem that selects the most-profitable set of squares whose total area does not exceed the blocks' total area. □

4.2 Packing row subframes via indirect guessing. The packing decision for the row squares is more challenging. First, for each subframe $F_{C,\ell,j}$ we guess the value $f(C,\ell,j)$. For each row subframe $F_{C,\ell,j}$ denote by $\bar{w}_{C,\ell,j}$ the number of squares packed in $F_{C,\ell,j}$. Observe that they all fit into $F_{C,\ell,j}$ even if each of them has a width of up to $k_{f(C,\ell,j)}$ since $w_{C,\ell,j} \geq \bar{w}_{C,\ell,j} \cdot k_{f(C,\ell,j)}$. As the next lemma shows we can guess the values $\bar{w}_{C,\ell,j}$ approximately by, intuitively, allowing only powers of $1+\epsilon$ for them.

LEMMA 4.8. *There is a set L with $|L| = O(\frac{\log n}{\log(1+\epsilon)})$ such that by losing a factor $1+O(\epsilon)$ we can assume that $\bar{w}_{C,\ell,j} \in L$ for each subframe $F_{C,\ell,j}$.*

Proof. We can assume that for each C, ℓ, j we have that $1 \leq \bar{w}_{C,\ell,j} \leq n$. We define $L := \{\lfloor (1+\epsilon)^i \rfloor | \forall i \geq 0, (1+\epsilon)^i \leq n\}$ and thus get $|L| = O(\log_{1+\epsilon} n) = O(\frac{\log n}{\log(1+\epsilon)})$.

Now, we round down each value $\bar{w}_{C,\ell,j}$ to the nearest value in L. Consider one such value $\bar{w}_{C,\ell,j}$ and let $\tilde{w}_{C,\ell,j} = (1+\epsilon)^i$ be the new (smaller) value. When we do this, we might lose at most $\bar{w}_{C,\ell,j} - \tilde{w}_{C,\ell,j} \leq (1+\epsilon)^{i+1} - (1+\epsilon)^i = \epsilon(1+\epsilon)^i = \epsilon\tilde{w}_{C,\ell,j} \leq \epsilon\bar{w}_{C,\ell,j}$ many squares. For these, we pick the ones with smallest profit in $F_{C,\ell,j}$, so we lose profit of at most an ϵ fraction of the total profit of the squares packed in $F_{C,\ell,j}$. □

We guess all values $\bar{w}_{C,\ell,j}$ in time $(\frac{\log n}{\log(1+\epsilon)})^{O(F(\epsilon))} = O_\epsilon(1) \cdot n$. Ideally, we would like to guess the values k_j. Then it would be very easy to compute the packing: for each index r the squares i with $s_i \in (k_{r-1}, k_r]$ form a group I_r. In the packing that we aim at (see Lemma 2.1) each row of height k_r contains only squares from I_r. We already guessed the total number of squares in each row and this yields the total number of squares from each group I_r. Denote by n_r this value, i.e., $n_r := \sum_\ell \sum_{j:f(C,\ell,j)=r} \bar{w}_{C,\ell,j}$. We simply select the n_r most profitable squares from I_r and pack them greedily into the rows of height k_r.

Unfortunately, we cannot guess the values k_j directly. Instead, we guess approximations for them *indirectly*. Recall that $k_0 = \epsilon^4 \cdot w_C$. We want to guess a value k'_1 with $k'_1 \leq k_1$ such that if we define the first group to be $I'_1 := \{i | s_i \in (k_0, k'_1]\}$ (instead of $I_1 := \{i | s_i \in (k_0, k_1]\}$) then the greedy packing of a

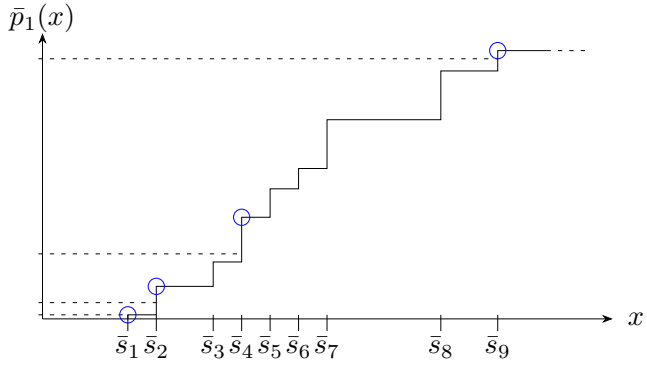


Figure 3: Example how the function $\bar{p}_1(x)$ might look like. $\bar{s}_1, \ldots, \bar{s}_9$ denote the distinct square sizes in increasing order. The graph shows the function $\bar{p}_1(x)$. Dashed lines denote powers of $(1+\epsilon)$, and the points highlighted with circles denote which values (i.e. which $\bar{s}_i$'s) are under consideration for k'_1.

subset of the squares in this group yields a profit that is almost as large as $p(\text{OPT} \cap I_1)$. To this end we ask ourselves: how much profit could we obtain from group I'_1 if we defined I'_1 to contain all squares in the size range $(k_0, x]$? We know that in the KR-packing the squares from group I_1 are packed only into the rows of height k_1 and we know already that there are n_1 slots for them.

We want to define k'_1, then $I'_1 := \{i | s_i \in (k_0, k'_1]\}$, and then pack the n_1 most profitable squares from I'_1 into the n_1 slots in the rows of height k_1 (strictly speaking we make the rows less high such that they have height $k'_1 \leq k_1$).

We want to allow only few candidate values for k'_1. To this end, we define a function $\bar{p}_1(x)$ where each value x is mapped to the total profit of the n_1 most profitable squares i with $s_i \in (k_0, x]$, i.e., the total profit of the first n_1 squares from group I'_1 if we defined $I'_1 := \{i | s_i \in (k_0, x]\}$. Note that $\bar{p}_1$ is a non-decreasing step-function with at most n steps. Due to the rounding of the square profits, we know that $\max_x \bar{p}_1(x) \leq n^2/\epsilon$. Now instead of allowing all possible values for k'_1, we allow only those values x where $\bar{p}_1(x)$ increases by a power of $1+\epsilon$, i.e., such that there is a $\kappa \in \mathbb{N}$ with $(1+\epsilon)^\kappa \in (\bar{p}_1(x-\delta), \bar{p}_1(x)]$ for each $\delta > 0$. For an illustration, see Figure 3. As a result, there are only $O(\log n / \log(1+\epsilon))$ such values and we define one of them to be k'_1 such that our profit from the resulting group $I'_1 := \{i | s_i \in (k_0, k'_1]\}$ will be at least $\frac{1}{1+\epsilon} \cdot p(\text{OPT} \cap I_1)$ and $k'_1 \leq k_1$. The latter inequality is important because our rows must fit into C, i.e., the rows and blocks intersected by each vertical line must not have a total height of more than h_C. Thus, we guess k'_1 in time $O(\log n / \log(1+\epsilon))$.

LEMMA 4.9. *Given $\bar{p}_1$ we can compute a set of $O(\log n / \log(1+\epsilon))$ values which contains a value $k'_1 \leq k_1$ such that $\bar{p}_1(k'_1) \geq \frac{1}{1+\epsilon}\bar{p}_1(k_1) = \frac{1}{1+\epsilon}p(\text{OPT} \cap I_1)$.*

Proof. Let $\bar{s}_1, \ldots, \bar{s}_{n'}$ denote the distinct sizes of large squares (i.e. squares with size $> k_0$). We compute $\bar{p}_1$. Note that $\bar{p}_1$ is a step function having all its steps at points x such that $x = \bar{s}_i$ for some square i, as only for these values of x new squares become part of the set I_1 (there does not have to be a step at such a point x; e.g., consider the case that the new squares of size x have very little profit and therefore they are not included in the subset of the n_1 most profitable squares of I_1). Intuitively, we define C to be those points where the function $\bar{p}_1$ "jumps" over a value $(1+\epsilon)^j$ for the first time. Formally, let $C := \{\bar{s}_i | \exists j : \bar{p}_1(\bar{s}_{i-1}) < (1+\epsilon)^j \leq \bar{p}_1(\bar{s}_i)\}$ where we define for convenience $\bar{p}_1(\bar{s}_0) = 0$. We have $|C| \leq \log_{1+\epsilon} \frac{n^2}{\epsilon} = O(\frac{\log n}{\log(1+\epsilon)})$.

Now, let k'_1 be the largest value in C that is not larger than k_1. Then there is a j s.t. $\bar{p}_1(k'_1) \geq (1+\epsilon)^j$ and $\bar{p}_1(k_1) < (1+\epsilon)^{j+1}$ and thus $\bar{p}_1(k'_1) \geq \frac{1}{1+\epsilon}\bar{p}_1(k_1)$. □

We iterate and we want to guess a value k'_2 (instead of k_2). Recall that there are n_2 slots for the group I_2 and the optimal packing we select the n_2 most profitable squares in I_2. We want to define k'_2 such that $k'_2 \leq k_2$ and such that if we pack the n_2 most profitable squares from the group $I'_2 := \{i | s_i \in (k'_1, k'_2]\}$ then we obtain almost the same profit as $p(\text{OPT} \cap I_2)$. We use the same approach as for defining k'_1. We define a function $\bar{p}_2(x)$ that maps each value x to the profit of the n_2 most profitable squares i with $s_i \in [k'_1, x)$. Observe that $\bar{p}_2(k_2) \geq p(\text{OPT} \cap I_2)$ because $k'_1 \leq k_1$. Like before, for defining k'_2 we allow only values x where $\bar{p}_2(x)$ increases by a power of $1+\epsilon$. There are only $O(\log n / \log(1+\epsilon))$ such values.

LEMMA 4.10. *Given $\bar{p}_2$ we can compute a set of $O(\log n / \log(1+\epsilon))$ values which contains a value $k'_2 \leq k_2$ such that $\bar{p}_2(k'_2) \geq \frac{1}{1+\epsilon}\bar{p}_2(k_2) \geq \frac{1}{1+\epsilon}p(\text{OPT} \cap I_2)$.*

Proof. The argumentation is basically analogous to the proof of the previous lemma, however, there is one difference. As $I'_2 = \{i | s_i \in (k'_1, k'_2]\}$, this set also depends on the previous guess k'_1. We define the set C as before and let k'_2 be the largest value in C that is not larger than k_2 as before. As $k'_1 \leq k_1$, we know that the set $\{i | s_i \in (k'_1, k'_2]\}$ is a superset of $\{i | s_i \in (k_1, k'_2]\}$, therefore the argumentation carries over. □

We guess the value k'_2 in time $O(\log n / \log(1+\epsilon))$ and we define $I'_2 := \{i | s_i \in (k'_1, k'_2]\}$. We continue in this way until for each j we guessed a value k'_j

and the corresponding set $I'_j := \{i | s_i \in (k'_{j-1}, k'_j]\}$. Since there are $1/\epsilon^4$ such values and for each of them there are $O(\log n / \log(1+\epsilon))$ possibilities, this yields $(\log n)^{O_\epsilon(1)}$ possibilities in total. Observe that each function $\bar{p}_j$ depends on the previous guess k'_{j-1}. There are $(\log n)^{O_\epsilon(1)}$ guesses overall and we can enumerate all of them in time $O_\epsilon(n) \cdot (\log n)^{O_\epsilon(1)}$.

The values k'_j imply the height of each row: for each row subframe $F_{C,\ell,j}$ we define its height $h_{C,\ell,j}$ to be $k'_{f(C,\ell,j)}$. In case that one frame $F_{C,\ell}$ is higher than h_C, i.e., $\sum_j h_{C,\ell,j} > h_C$, then we reject this set of guesses. In particular, then there must be some value k'_j with $k'_j > k_j$ and thus our guess was wrong. Among all remaining guesses, we select the one that yields the maximum total profit if we take the n_r most profitable squares from each group I'_r.

THEOREM 4.1. *There is a $(1+\epsilon)$-approximation algorithm for the two-dimensional knapsack problem for squares if the height and the width of the knapsack differ by more than a factor of $1/\epsilon^4$. The running time of the algorithm is $O_\epsilon(1) \cdot n^{O(1)}$.*

Proof. The algorithm proceeds as follows: First, we guess the large squares as described in Section 3 in time $(\frac{1}{\log(1+\epsilon)})^{O(F(\epsilon))} \cdot n = O_\epsilon(1) \cdot n$. Next, we guess the number of frames and subframes of each frame in time $1/\epsilon^{O(1/\epsilon^4)} = O_\epsilon(1)$. Then we guess $\tilde{\alpha}$ in time $\frac{n \log n}{\log(1+\epsilon)} = O_\epsilon(1) \cdot n^{O(1)}$ and use it to guess the rounded width and height of each of the $O(1/\epsilon^4)$ blocks in time $(\frac{\log n}{\log(1+\epsilon)})^{O(1/\epsilon^4)} = O_\epsilon(1) \cdot n$. Row squares can be packed into the corresponding row subframes in a greedy manner and are guaranteed to fit. We can use an FPTAS for the knapsack problem to select a set of small items whose total area does not exceed the area of the blocks while maximizing their total profit. Using Lemma 4.2, we can pack almost all of these into the blocks while losing at most an ϵ fraction of the profit. The total running time is $O_\epsilon(1) \cdot n^{O(1)}$. □

5 General case.

We present our EPTAS for the general case now. There are two new difficulties compared to the special case from Section 4:

- there can be several elongated cells now (rather than only one that spans the entire knapsack). For elongated cells C with $w_C \geq h_C$ we cannot guess the height in time $O_\epsilon(1) \cdot n^{O(1)}$ and for elongated cells C' with $h_{C'} \geq w_{C'}$ we cannot guess the width in time $O_\epsilon(1) \cdot n^{O(1)}$. Therefore, we incorporate the guessing of these values into the indirect guessing framework for the heights of the horizontal (widths of the vertical) row subframes.
- we can no longer partition the input squares into two groups such that the squares from one group are assigned only to the blocks (like the squares i with $s_i \leq \epsilon^4 \cdot h_C$ in the previous special case) and the others are only assigned to the rows (like the squares i with $s_i > \epsilon^4 \cdot h_C$). Therefore, for each group I_r we guess an estimate of either the total number of squares in each block or the total occupied area in each block in order to apply our framework for indirect guessing.

First, we guess the large squares I'_L as described in Section 3. Next, we guess the number of block and elongated cells, and for each elongated cell the number of frames and the number of subframes in each frame. We will use the term *block* for a block cell or a block subframe. We guess the sizes of all blocks like in Section 4, with the only difference that we can now have up to $O(F(\epsilon))$ many blocks instead of $O(1/\epsilon^4)$: we first guess the value $\tilde{\alpha}$ in time $O(\frac{n \log n}{\log(1+\epsilon)})$ and then the heights and widths of all blocks in time $(\frac{\log n}{\log(1+\epsilon)})^{O(F(\epsilon))}$.

LEMMA 5.1. *By losing a factor $1 + O(\epsilon)$ we can round the heights and widths of all block cells and all block subframes such that we can guess all of them in time $\frac{n \log n}{\log(1+\epsilon)} \cdot (\frac{\log n}{\log(1+\epsilon)})^{O(F(\epsilon))} \leq O_\epsilon(1) \cdot n^{O(1)}$.*

In the packing given by Lemma 2.1 each elongated cell is packed as a KR-packing. Therefore, for each horizontal elongated cell C there is a set of at most $1/\epsilon^4$ values $k_0, k_1, k_2, \ldots$ defined *locally* for C, such that for the height of each row subframe $F_{C,\ell,j}$ of C there is an integer $f(C,\ell,j) \geq 1$ such that $h_{C,\ell,j} = k_{f(C,\ell,j)}$ and $F_{C,\ell,j}$ contains only squares i with $s_i \in (k_{f(C,\ell,j)-1}, k_{f(C,\ell,j)}]$, while we assume that $k_0 = 0$. We establish now that we can assume that there are $O_\epsilon(1)$ *global* values with these properties.

LEMMA 5.2. *Let $k_0 = 0$. By losing a factor $1 + O(\epsilon)$ and by increasing the number of elongated cells within the knapsack and the total number of row subframes in all elongated cells to $O(1/\epsilon^{10} F(\epsilon)^3)$ each we can assume that there are at most $O(1/\epsilon^4 F(\epsilon)^2)$ (global) values $k_1 \leq k_2 \leq \ldots$ such that for each horizontal (or vertical) row subframe $F_{C,\ell,j}$ there is a value $f(C,\ell,j) \geq 1$ such that $h_{C,\ell,j} = k_{f(C,\ell,j)}$ (or $w_{C,\ell,j} = k_{f(C,\ell,j)}$) and $F_{C,\ell,j}$ contains only squares i with $s_i \in (k_{f(C,\ell,j)-1}, k_{f(C,\ell,j)}]$. Furthermore, for each block B of height h_B and width w_B and each r, we know that either $k_r \leq \epsilon^2 \min\{w_B, h_B\}$ or $k_{r-1} \geq \epsilon^2 \min\{w_B, h_B\}$.*

Proof. Let $C_1, C_2, \ldots$ be the elongated cells. Denote by $k_1^{(i)}, k_2^{(i)}, \ldots$ the local k_r-values for cell C_i. We define the set $K = \{k_j^{(i)} | \forall i, j\} \cup$

$\{\epsilon^2 \min\{w_B, h_B\} | \forall$ blocks B of width w_B and height $h_B\}$. Clearly, $|K| \leq 1/\epsilon^4 \cdot F(\epsilon) + F(\epsilon) = O(1/\epsilon^4 F(\epsilon)^2)$.

It remains to show that there is a KR-packing that corresponds to these global k-values. Consider the original KR-packing of one elongated cell, i.e. the one using local k_r-values (w.l.o.g. we consider a vertical elongated cell; the argument for horizontal elongated cells is analogous). We will show how to transform it into another KR-packing that adheres to the global values K. Consider a row subframe $F_{C,\ell,j}$ of height $k_r^{(i)}$. Assume that the largest value smaller than $k_r^{(i)}$ in K, k^*, is a value that does not belong to the local k_r-values of this cell (it might be some $k_{r'}^{(i')}$ for $i' \neq i$ or a value $\epsilon^2 \min\{w_B, h_B\}$); so in particular $k^* \in (k_{r-1}^{(i)}, k_r^{(i)})$. The subframe $F_{C,\ell,j}$ might now contain squares of side length in $(k_{r-1}^{(i)}, k_r^{(i)}]$, i.e. in particular squares with size smaller than k^* and also squares with size larger than k^*.

In a KR-packing that adheres to the values K, these squares must not be packed in the same row subframe. We therefore do the following: Assume w.l.o.g. that the squares packed in subframe $F_{C,\ell,j}$ are sorted in decreasing order of size from left to right. We split this one elongated cell into several smaller ones: the first one contains all frames above the "problematic" frame $F_{C,\ell}$, the second contains all frames below $F_{C,\ell}$, the third consists of one frame that has subframes $F_{C,\ell,1}, \ldots, F_{C,\ell,j-1}$, the fourth ones contains subframes $F_{C,\ell,j+1}, \ldots, F_{C,\ell,p}$ (where p is the number of subframes of $F_{C,\ell}$) contained in one single frame and the fifth consists only of the subframe $F_{C,\ell,j}$ (see Figure 4 for illustration). Note that all these are valid KR-packings. Now, in the new elongated cell that only contains subframe $F_{C,\ell,j}$, we can split the frame into two frames, each having one subframe: one subframe has height $k_r^{(i)}$ and contains only squares of size in $(k^*, k_r^{(i)}]$, the other one has height k^* and contains squares of size in $(k_{r-1}^{(i)}, k^*]$.

The number of row subframes before this splitting of elongated cells was at most $O(F(\epsilon)) \cdot O(1/\epsilon^6)$. In the worst case, we splitted each row subframe into $O(1/\epsilon^4 F(\epsilon)^2)$ row subframes, hence after this process we have at most $O(1/\epsilon^{10} F(\epsilon)^3)$ many row subframes. As each elongated cells must contain at least one row subframe or block subframe and the number of block subframes stayed unchanged, we have at most $O(1/\epsilon^{10} F(\epsilon)^3)$ many elongated cells as well.

By introducing k_r-values of the form $\epsilon^2 \min\{w_B, h_B\}$ for each block B, we also made sure that for each B and $k_r \in K$, either $k_r \leq \epsilon^2 \min\{w_B, h_B\}$ or $k_{r-1} \geq \epsilon^2 \min\{w_B, h_B\}$. □

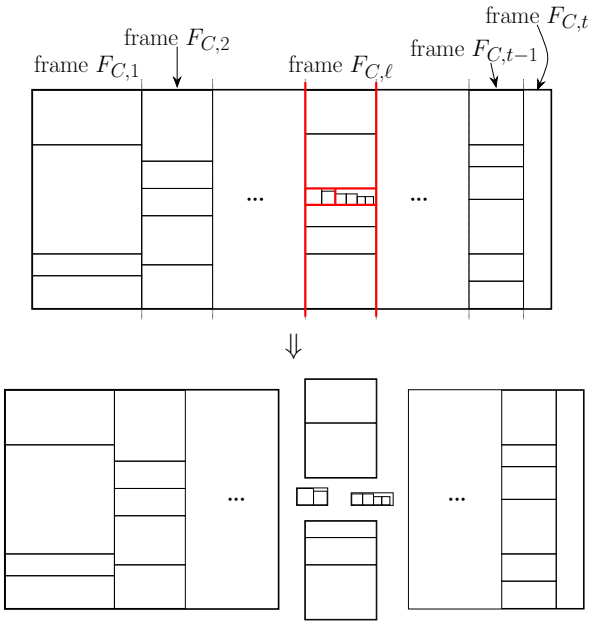


Figure 4: Splitting one elongated cell into five.

Then we guess some quantities of the row subframes (similar to Section 4).

LEMMA 5.3. *By losing a factor* $1+O(\epsilon)$ *in time* $O_\epsilon(1) \cdot n$ *we can guess for each row subframe* $F_{C,\ell,j}$

- *the number* $\bar{w}_{C,\ell,j}$ *of squares packed in* $F_{C,\ell,j}$,
- *the value* $f(C,\ell,j)$ *indicating that if* $F_{C,\ell,j}$ *is a horizontal row then for its height* $h_{C,\ell,j}$ *it holds that* $h_{C,\ell,j} = k_{f(C,\ell,j)}$, *and if* $F_{C,\ell,j}$ *is a vertical row then for its width* $w_{C,\ell,j}$ *it holds that* $w_{C,\ell,j} = k_{f(C,\ell,j)}$

Proof. First, we want to guess the values $\bar{w}_{C,\ell,j}$ for each row subframe $F_{C,\ell,j}$. Similar to Lemma 4.8, we round down $\bar{w}_{C,\ell,j}$ to the nearest value in L (as defined in the proof of Lemma 4.8), and by throwing away the least profitable squares we lose at most an ϵ fraction of the total profit in $F_{C,\ell,j}$. For each row subframe, we have $O(\frac{\log n}{\log(1+\epsilon)})$ possible values, i.e., we can guess the $\bar{w}_{C,\ell,j}$ for all row subframes in time $(\frac{\log n}{\log(1+\epsilon)})^{O(1/\epsilon^{10} F(\epsilon)^3)} = (\log n)^{O_\epsilon(1)} \leq O_\epsilon(1) \cdot n$.

Second, we need to guess $f(C,\ell,j)$ for each of the at most $O(1/\epsilon^{10} F(\epsilon)^3)$ row subframes. There are $O(1/\epsilon^4 F(\epsilon)^2)$ many possibilities per subframe, so $(1/\epsilon^4 \cdot F(\epsilon)^2)^{O(1/\epsilon^{10} F(\epsilon)^3)}$ possibilities in total, which is a constant only depending on ϵ.

Hence, in total we need time $O_\epsilon(1) \cdot n$ to guess the values $\bar{w}_{C,\ell,j}$ and $f(C,\ell,j)$ for all row subframes. □

As before, for each r denote by n_r the total number of squares in rows of height k_r, i.e., $n_r :=$

$\sum_C \sum_\ell \sum_{j:f(C,\ell,j)=r} \bar{w}_{C,\ell,j}$.

Note that we did not guess the heights and widths for the row subframes. Those are implied once we guess the values $k_1, k_2, ...$ since the height/width of each vertical/horizontal row subframe $F_{C,\ell,j}$ is at least $\bar{w}_{C,\ell,j} \cdot k_{f(C,\ell,j)}$ and a larger height/width is not necessary to fit $\bar{w}_{C,\ell,j}$ squares of size at most $k_{f(C,\ell,j)}$. In contrast to Section 4 a square i with size $i \in (k_r, k_{r+1}]$ might be assigned to a row (of height k_{r+1}) *or* to a block. In our indirect guessing framework, we need in particular a function $\bar{p}_1(x)$ that maps each x to the profit we would obtain from group I_1 if the group I_1 contained all squares i with $s_i \in (k_0, x]$. This profit depends now not only on the number of slots in the rows for the squares in I_1 but also in the space that we allocate to I_1 in each block. In order to handle this, we first guess for each k_r-value for $r \geq 2$, whether $\frac{k_r}{k_{r-1}} > 1+\epsilon$ or $\frac{k_r}{k_{r-1}} \leq 1+\epsilon$. We denote by K_L the union $\{k_r | \frac{k_r}{k_{r-1}} > 1+\epsilon\}$ of all k_r-values for which the first inequality holds and by K_S all values with the latter property. If $k_r \in K_L$ then we guess for the set I_r and each block B the area that squares from I_r occupy in B. Otherwise, if $k_r \in K_S$ then for each block B we guess the number of squares from I_r packed in B. Recall that for each block B we already guessed its height and width.

LEMMA 5.4. *We can compute for each k_r sets L'_r, L''_r with $|L'_r|, |L''_r| = O(\log n / \log(1+\epsilon))$ such that by losing a factor $1 + O(\epsilon)$, we can assume that for each block B and each group $I_r = \{i | s_i \in (k_{r-1}, k_r]\}$ the following holds:*

- *If $k_r \in K_L$, then there is a value $a_{B,r} \in L'_r$ such that the total area used by squares of group I_r in B is bounded by $a_{B,r}$ and either $a_{B,r} = 0$ or $a_{B,r} \geq \frac{1}{\epsilon} k_r^2$.*
- *If $k_r \in K_S$, then there is a value $n_{B,r} \in L''_r$ such that the number of squares of group I_r packed in B is bounded by $n_{B,r}$.*

Moreover, for each block B we have $\sum_{r \in K_L} a_{B,r} + \sum_{r \in K_S} n_{B,r} k_r^2 \leq (1+\epsilon) a_B$ where a_B denotes the area of B.

Proof. We will prove the two statements of the lemma separately.

Case 1: $k_r \in K_L$**.** Let $L'_r = \{(1+\epsilon)^i | 1/\epsilon k_r^2 \leq (1+\epsilon)^i \leq n k_r^2\} \cup \{0\}$. Clearly, $|L'_r| \leq \log_{1+\epsilon}(n) + 1 = O(\frac{\log n}{\log(1+\epsilon)})$. Consider a block B and a group I_r; let $a^*_{B,r}$ be the area of squares of I_r in B in the solution under consideration. We want to guess a value $a_{B,r} \leq a^*_{B,r}$ such that most of the squares of I_r still fit in this area (most of the squares means enough squares so we do not lose too much profit). Let $a_{B,r}$ be the largest value in L'_r that is at most $a^*_{B,r}$.

First, assume that $a_{B,r} \geq 1/\epsilon k_r^2$. Say $a_{B,r} = (1+\epsilon)^i$, thus we know $a^*_{B,r} < (1+\epsilon)^{i+1}$ and therefore $a^*_{B,r} - a_{B,r} < (1+\epsilon)^{i+1} - (1+\epsilon)^i = \epsilon(1+\epsilon)^i = \epsilon a_{B,r}$. That is, the total area of squares of I_r that were packed in B before but now cannot be packed in B ("lost" squares) is at most an ϵ fraction of the guessed area. Hence, we can partition these squares into $1/\epsilon$ many buckets, then delete squares in one of these buckets (losing profit at most an ϵ fraction of the whole profit packed in this area) and pack the lost squares there.

Now assume that $a^*_{B,r} < 1/\epsilon k_r^2$. That means, if we define as before $a_{B,r} = 0$, we will not pack any I_r squares into B, but these squares might carry a significant amount of profit. In this case, we want to *over*estimate the area instead, i.e. we want to guess $a_{B,r} = \min(L'_r \setminus \{0\}) \leq (1+\epsilon)(1/\epsilon k_r^2)$. However, now we need to argue that this area still fits into B, especially as it could happen that many groups I_r have $a^*_{B,r} < 1/\epsilon k_r^2$ for one block B (i.e. B contains squares of many different groups, and of each group only few squares are packed in B). Therefore, we want to bound the total area by which we overload block B for all these groups. To formalize this, let S denote the set of all indices r s.t. $k_r \in K_L$ and group r has $a^*_{B,r} < 1/\epsilon k_r^2$.

Given any group I_r that has squares in B, we have the upper bound $(1+\epsilon)(1/\epsilon k_r^2)$ on $a_{B,r}$, which might be large compared to the area of B. Here, we use the second property that we have proven in Lemma 5.2: either $k_r \leq \epsilon^2 \min\{w_B, h_B\}$, or $k_{r-1} \geq \epsilon^2 \min\{w_B, h_B\}$, which means that if any square of a certain group I_r is packed into B, then the upper bound for square sizes in this group, k_r, is at most $\epsilon^2 \min\{w_B, h_B\}$. Thus we get $a_{B,r} \leq (1+\epsilon)(1/\epsilon k_r^2) \leq (1+\epsilon) 1/\epsilon \cdot \epsilon^4 \min\{w_B, h_B\}^2 = O(\epsilon^3) a_B$.

Sort the indices in S in decreasing order of the size of the corresponding squares; let $r_1, r_2, \ldots$ be this sorted sequence. We know that two of the corresponding k_r-values of S are always at least a factor $1+\epsilon$ apart (as we are talking about values in K_L). Using a geometric sum argument, we have that $\sum_{r_i \in S} a_{B,r_i} = a_{B,r_1} + a_{B,r_2} + a_{B,r_3} + \ldots \leq a_{B,r_1} + \frac{1}{1+\epsilon} a_{B,r_1} + \frac{1}{(1+\epsilon)^2} a_{B,r_1} + \ldots = a_{B,r_1} \sum_{i=0}^{|S|-1} (\frac{1}{1+\epsilon})^i = a_{B,r_1} \frac{1-(\frac{1}{1+\epsilon})^{|S|}}{1-\frac{1}{1+\epsilon}} \leq O(\epsilon^3) a_B \frac{1+\epsilon}{\epsilon} = O(\epsilon) a_B$, i.e. the total area of all these excess squares is at most an ϵ fraction of the total area of B. Thus, the lemma holds.

Case 2: $k_r \in K_S$**.** Now, we consider a k_r-value that is in K_S, i.e. we need to guess the number of squares of this group in B. Let $L''_r = \{\lfloor (1+\epsilon)^i \rfloor | 1 \leq (1+\epsilon)^i \leq n\} \cup \{0\}$, and clearly $|L''_r| \leq O(\log_{1+\epsilon} n)$. Let $n^*_{B,r}$ denote the number of squares of group I_r that are

packed in B, and let S be the set of these squares; we want to guess a value $n_{B,r} \leq n^*_{B,r}, n_{B,r} \in L''_r$, such that the $n_{B,r}$ most profitable squares from S still give profit at least a $\frac{1}{1+\epsilon}$ fraction of the profit of S.

Let $n_{B,r}$ be the largest value in L''_r that is smaller than $n^*_{B,r}$, i.e. $n_{B,r} = (1+\epsilon)^i$ for some i and $n^*_{B,r} < (1+\epsilon)^{i+1}$. The difference between the two numbers is $n^*_{B,r} - n_{B,r} < \epsilon n_{B,r} \leq \epsilon n^*_{B,r}$ similar as before. Therefore, if we select that $n_{B,r}$ most profitable squares from S, they have profit at least $1-\epsilon$ times the profit of S. It could now however happen that these squares selected by the algorithm have larger area than the ones selected by the optimal solution and hence do not fit into the block. However, as these squares belong to groups I_r with $r \in K_S$, we know, that they are by at most a factor $1+\epsilon$ larger than the squares selected by the optimal solution. Thus the lemma is proven. □

Using Lemma 5.4 we guess the values $a_{B,r}$ and $n_{B,r}$ for each of the $O_\epsilon(1)$ blocks B and each of the $O_\epsilon(1)$ values r in time $(\log n)^{O_\epsilon(1)}$.

5.1 Indirect guessing. We perform the indirect guessing of the values $k_1, k_2, ...$ similarly as in Section 4. Recall that $k_0 = 0$. We define a function $\bar{p}_1(x)$ that *in an approximative sense* maps each value x to the maximum profit that we can obtain from the group I_1 if we defined I_1 to contain all squares i with $s_i \in (0, x]$. Formally, we define $\bar{p}_1(x)$ to be the maximum profit of a subset $\tilde{I} \subseteq \{i | s_i \in (0,x]\}$ such that we can assign n_1 squares from $\tilde{I}$ to the rows $F_{C,\ell,j}$ with $f(C,\ell,j) = 1$ and the remaining squares of $\tilde{I}$ can be assigned to the blocks such that for each block B either the squares of $\tilde{I}$ assigned to B have a total area of at most $a_{B,1}$ (since $k_1 \in K_L$; later, for $k_r \in K_S$ we replace the latter condition by requiring that at most $n_{B,r}$ squares of the respective set $\tilde{I}$ are assigned to B if $k_1 \in K_S$). We say that such a set $\tilde{I}$ *fits into the space for group* I_1. Unfortunately, in contrast to Section 4 it is NP-hard to compute $\bar{p}_1(x)$ since it is a generalization of the multi-dimensional knapsack problem. However, in polynomial time we can compute an approximation for it, using that $a_{B,r} \in \{0\} \cup [\frac{1}{\epsilon}k_r, \infty)$ for each B, r with $k_r \in K_L$.

LEMMA 5.5. *In time $O_\epsilon(1) \cdot n^{O(1)}$ we can compute a function $\tilde{p}_1(x)$ such that $\frac{1}{1+\epsilon}\bar{p}_1(x) \leq \tilde{p}_1(x) \leq \bar{p}_1(x)$ for each x. The function $\tilde{p}_1(x)$ is a step-function with $O(n)$ steps. Moreover, for each x in time $O_\epsilon(1) \cdot n^{O(1)}$ we can compute a set $\tilde{I} \subseteq \{i | s_i \in (0,x]\}$ with $p(\tilde{I}) = \tilde{p}_1(x)$ such that $\tilde{I}$ fits into the space for group I_1.*

The proof for this uses LP-rounding similarly as in [11].

Proof. As before, we can argue that $\bar{p}_1(x)$ is a step-function with at most n steps: Let $\bar{s}_1, \ldots, \bar{s}_{n'}$ be the distinct sizes of squares in I_1. The value of the function $\bar{p}_1(x)$ will only change when the set $\bar{I}_1 = \{i | s_i \in (0,x]\}$ changes, which is only the case when x is one of these $\bar{s}_i$-values. For the function $\tilde{p}_1(x)$, we define the values at points $\bar{s}_i$ to be the values of $\bar{p}_1(x)$ at these points; in between two such points, $\tilde{p}_1(x)$ is defined to be constant.

In order to compute $\bar{p}_1$, we therefore need to evaluate for each $\bar{s}_i$, what is the maximum profit we can gain from a set $\tilde{I}_1 \subseteq \bar{I}_1$ such that $\tilde{I}_1$ fits into the space for I_1. Solving this exactly is NP-hard, thus we compute an approximation for this function, $\tilde{p}_1(x)$. We formulate the problem as an LP and then perform a rounding similar to [11] (which is based on a method for the generalized assignment problem [20]). We do all that follows for each r separately.

Notation and LP formulation. When considering a certain value r, let $\mathcal{R}_r$ be the set of all row subframes of height (for horizontal row subframes) or width (for vertical row subframes) k_r. For a square i, let $\mathcal{B}_i$ be the set of all blocks B such that $s_i \leq \epsilon^2 \min\{w_B, h_B\}$, where w_B, h_B are the width and height of B (which we guessed already). We introduce binary variables $x_{i,R}$ for each square $i \in I_r$ and $R \in \mathcal{R}_r$ that indicates whether square i is packed in row subframe C, and variables $x_{i,B}$ for each square $i \in I_r$ and block $B \in \mathcal{B}_i$ (i.e. we only introduce these variables for pairs of squares and blocks s.t. the square is small enough to be packed in the block). Now, consider the following LP, which assumes that $k_r \in K_L$:

$$\text{maximize} \quad \textstyle\sum_{i \in I_r} \left(p_i \cdot \sum_{R,B} (x_{i,R} + x_{i,B}) \right)$$

$$\text{(5.1)} \quad \text{s.t.} \quad \sum_{i \in I_r} x_{i,R} \leq \bar{w}_R \qquad \forall R \in \mathcal{R}_r$$

$$\text{(5.2)} \quad \sum_{i \in I_r} x_{i,B} \cdot s_i^2 \leq a_{B,r} \quad \forall \text{ blocks } B$$

$$\text{(5.3)} \quad \sum_{R} x_{i,R} + \sum_{B} x_{i,B} \leq 1 \qquad \forall i \in I_r$$

$$\text{(5.4)} \quad x_{i,B} \geq 0$$

$$\text{(5.5)} \quad x_{i,R} \geq 0$$

The constraints (5.1) ensure that the number of squares packed into each row subframe is at most the number of squares that should be packed in this row subframe according to our guess. The constraints (5.2) ensure that the total area of squares from this group packed into any block B does not exceed the guessed area $a_{B,r}$. The constraints (5.3) ensure that each square is packed at most once. If instead $k_r \in K_S$, we replace the constraints (5.2) by $\sum_{i \in I_r} x_{i,B} \leq n_{B,r}$. This LP gives an upper bound for OPT, given that our guesses for $\bar{w}_r, a_{B,r}, n_{B,r}$ are correct.

Note that the number of variables is at most $n \cdot$

$O(1/\epsilon^{10}F(\epsilon)^3)$, and the number of constraints is also at most $n \cdot O(1/\epsilon^{10}F(\epsilon)^3)$. Therefore, we can solve this LP in EPTAS-time $n^{O(1)} \cdot O_\epsilon(1)$.

Rounding the fractional LP solution. We can compute a vertex solution to this LP and we will now discuss how to round this solution x^* to a feasible integral one. We create a bipartite graph such that the fractional LP-solution corresponds to a fractional matching in this bipartite graph. The vertices on the left side of the graph correspond to the squares in I_r; call these vertices $v_1, \dots, v_{|I_r|}$. The vertices on the right side of the graph correspond to the row subframes and blocks in the following way: for each row subframe R of height k_r (or width k_r, if it is a vertical one), we create $\bar{w}_R$ many vertices $v_1^R, \dots, v_{\bar{h}_C}^R$, for each block B that can get squares from I_r, we create $t_B = \lceil \sum_{i \in I_r} x^*_{i,B} \rceil \leq n$ many vertices $v_1^B, \dots, v_{t_B}^B$. Now, we will define edges and at the same time specify the fractional matching M^{fr} by defining values $y_e \in (0,1]$ for each edge e. Consider the left hand side vertices in some order $v_1, \dots, v_{|I_r|}$; for vertex v_i, denote by f_i the remaining fraction of edges, initially $x^*_{i,R}$, and for a vertex v_k^R, denote by f_k^R the sum over y_e for all incident edges e (initially this is zero). Let the current vertex be v_i, let v_k^R be the vertex for R with the smallest index k such that $f_k^R < 1$. Create an edge between v_i and v_k^R, and let $y_{v_i,v_k^R} = \max\{f_i, 1 - f_k^R\} =: p$. Set $f_i := f_i - p$. Repeat this with the next v_{k+1}^R until $f_i = 0$, then go on to the next vertex v_{i+1}. Now, we define the fractional matching M^{fr} to be the set of all edges, each edge e having weight y_e. Note that the weights y_e assigned to the edges incident to a certain left side vertex v_i sum to $x^*_{i,R}$.

Now, for a block B, we consider the squares fractionally assigned to B by the LP solution in decreasing order of their area. For the B-vertices we do the same procedure as for the C-vertices and consider the squares in this order to create edges and augment our fractional matching (here, initially $f_i = x^*_{i,B}$). Finally, to each edge incident to a left side vertex v_i, we assign profit p_i. This way, the total profit of M^{fr}, $\sum_{e=(v_i,u) \in G} y_e p_i$, is the same as the objective value of the LP for x^*, which is $\sum_{i,R,B}(x^*_{i,R} + x^*_{i,B})p_i$. The number of nodes in this graph is n on the left side and $O_\epsilon(1) \cdot n$ on the right side.

We can now find an integral matching in this graph with at least the same profit as M^{fr} (this is a standard result in matching theory, see e.g. [18]; the running time can be bounded by $O(|V|^4)$, where $|V|$ is the number of nodes, i.e., running time $O_\epsilon(1) \cdot n^4$ in our case). It remains to show how to construct a solution for our original packing problem from this integral matching. First of all, if an edge between v_i and v_j^C is taken by the integral matching M^{int}, we assign this square i to cell C. By construction of the graph, we cannot assign more than $\bar{w}_C$ squares to a specific row subframe, and these squares surely fit into the row subframe.

Now, consider squares matched to block-vertices. First assume that $k_r \in K_L$ and thus, we used the area-based block constraints in the LP. It might now be the case that the total area assigned to this block by the LP (in a fractional way) is less than the total area of the integral assignment, hence we cannot assign all squares that are matched to this block in the integral solution to this block without violating the constraint. We will now argue that it is nevertheless possible to assign squares to B that fulfill the constraint and have profit at least $\frac{1}{1+\epsilon}$ of the profit of all squares matched to B in the integral matching.

Consider a specific block B, and let $x_{i,B,k}$ denote the fraction with which the edge (v_i, v_k^B) was selected in M^{fr}. We call the area that is assigned to B in the fractional solution due to vertex v_k^B the *load of this vertex*. This load is defined as $L_k^B := \sum_{i \in I_r} x_{i,B,k} s_i^2$. Now, denote by v_{i_k} the left hand side vertex of the graph that is matched to the right hand side vertex v_k^B in M^{int}. Due to the sorting of the squares in decreasing order of size when creating edges and M^{fr}, and due to the fact that $\sum_i x_{i,B,k} = 1$ for $k < t_B$, we know that the load of vertex v_k^B is at least $s_{i_{k+1}}^2$ for $k < t_B$: all vertices having edges to v_k^B have area at least as large as square i_{k+1}, as this square has an edge to v_{k+1}^B. Now, it follows that the total load of all vertices from the LP solution is $LP_B := \sum_{k=1}^{t_B} L_k^B \geq \sum_{k=1}^{t_B - 1} s_{i_{k+1}}^2 = \sum_{k=2}^{t_B} s_{i_k}^2$, which is the total area assigned to B by the integral matching for vertices $v_2^B, \dots, v_{t_B}^B$. Hence, these squares must fit into the area in B reserved for I_r squares, due to constraint (5.2) in the LP. Thus, the integral solution might exceed this area only due to the square i_1, which is matched to vertex v_1^B. Now it comes into play that we guaranteed in Lemma 5.4 that if $a_{B,r} > 0$, then we have $a_{B,r} \geq 1/\epsilon k_r^2$. Partition the squares into buckets in the following way: Consider squares in any order, add squares to one bucket until the total area of squares in this bucket is at least k_r^2; then start filling the next bucket. The area of squares within one bucket is at most $2k_r^2$ as each square has area at most k_r^2, thus we have at least $2/\epsilon$ many buckets. The profit in the least-profitable bucket is hence at most an $O(\epsilon)$ fraction of the total profit of these squares, and we can remove them. The free area is at least k_r^2, which fits the excess square i_1.

Now, assume that $k_r \in K_S$. In this case, we replace constraints (5.2) by the simpler ones $\sum_{i \in I_r} x_{i,B} \leq n_{B,r}$. We need to argue that again for each block B, the squares assigned by the integral solution do not violate

this constraint. In this case, $t_B \leq n_{B,r}$, thus we have at most that many squares matched to this block, hence the constraint is trivially satisfied by the integral solution. □

Like before, we guess a value $k_1' \leq k_1$ and for k_1' we allow only those values where $\tilde{p}_1(x)$ increases by a factor of $1+\epsilon$. Therefore, there are only $O_\epsilon(\log n)$ possibilities. We define $I_1' := \{i | s_i \in (0, k_1']\}$. We iterate as in Section 4. We next guess a value k_2' and we define the function $\bar{p}_2(x)$ similarly as above. Note that $\bar{p}_2(k_2) \geq p(\mathrm{OPT} \cap I_2)$ since $k_1' \leq k_1$ and our condition for when a set $\tilde{I}$ fits into the space for group I_1 is a relaxation.

Again, we can compute an approximation $\tilde{p}_2(x)$ for $\bar{p}_2(x)$. Overall, there are $(\log n)^{O_\epsilon(1)}$ guesses and we can enumerate all of them in time $O_\epsilon(n) \cdot (\log n)^{O_\epsilon(1)}$.

LEMMA 5.6. *In time $O_\epsilon(n) \cdot (\log n)^{O_\epsilon(1)} = O_\epsilon(1) \cdot n^{O(1)}$ we can enumerate guesses for all values k_j such that for one such guess $k_1', k_2', \ldots$ we have for each j that $k_j' \leq k_j$ and there is a set $\tilde{I}_j \subseteq I_j' = \{i | s_i \in (k_{j-1}', k_j']\}$ with $p(\mathrm{OPT} \cap I_j) \leq (1 + O(\epsilon)) p(\tilde{I}_j)$.*

Proof. We iterate the process described in Lemma 5.5 for all k_r-values: First, compute the function $\tilde{p}_1(x)$ (i.e., compute for all square sizes s_i the value $\tilde{p}_1(s_i)$) in time $O_\epsilon(1) \cdot n^{O(1)}$ by solving the LP and then finding an integral matching in the bipartite graph defined by this LP solution as described in the proof of Lemma 5.5). We get, as before, $O(\frac{\log n}{\log(1+\epsilon)})$ many candidate values for k_1 at those points where the function "jumps" over a power of $1+\epsilon$, and guess the correct one, k_1'. So far, this takes time $O_\epsilon(1) \cdot n^{O(1)} + O(\frac{\log n}{\log(1+\epsilon)})$. For each of the k_1'-guesses, we now compute in the same manner the function $\tilde{p}_2(x)$, which gives $O(\frac{\log n}{\log(1+\epsilon)})$ possible choices for k_2'. Note here that $\tilde{p}_2(k_2') \geq p_2(k_2)$, as $k_1' \leq k_1, k_2' \leq k_2$, and again the corresponding LP gives an upper bound on the optimal solution. We continue like that until we have guessed all k_r'-values. The total running time becomes $O_\epsilon(1) \cdot n^{O(1)} + O(\frac{\log n}{\log(1+\epsilon)}) \cdot (O_\epsilon(1) \cdot n^{O(1)} + O(\frac{\log n}{\log(1+\epsilon)}) \cdot (\ldots)) = \left(\frac{\log n}{\log(1+\epsilon)}\right)^{O(1/\epsilon^{10} F(\epsilon)^3)} \cdot O_\epsilon(1) \cdot n^{O(1)} = O_\epsilon(1) \cdot n^{O(1)}$. □

5.2 Computing the packing. The guessed values k_j' imply the heights and widths of the elongated cells in the following way. Consider an elongated cell C and assume w.l.o.g. that $h_C \leq w_C$. Then h_C is the maximum height of a frame of C, i.e., $h_C := \max_\ell \sum_j h_{C,\ell,j}$ where for each row subframe $F_{C,\ell,j}$ we have that $h_{C,\ell,j} := k'_{f(C,\ell,j)}$ (for block subframes we already guessed the height and the width). The width w_C of C is the total width of all its frames, i.e., $w_C := \sum_\ell w_{C,\ell}$ where for each frame $F_{C,\ell}$ its width is the maximum width of a subframe, i.e., $w_{C,\ell} := \max_j w_{C,\ell,j}$ and for each row subframe $F_{C,\ell,j}$ we define $w_{C,\ell,j} := \bar{w}_{C,\ell,j} \cdot k_{f(C,\ell,j)}$ (again, for block subframes we already guessed the height and the width).

We obtained $O(F(\epsilon))$ large squares and $O(F(\epsilon))$ cells for which we now know all heights and widths. We verify in $O_\epsilon(1)$ time that there exists a feasible packing for them (if not then we reject our guess). Observe that we *under*estimated all guessed quantities and therefore for the correct guesses a feasible packing must exist.

For each group I_j' we obtained a set $\tilde{I}_j$ that fits into the space of group I_j. We take the union of all these sets $\tilde{I}_j$. This is not yet a feasible solution: even though each block B gets squares assigned whose total size does not exceed the size of B, a feasible packing is not guaranteed to exist. However, we can remove some of the squares such that we lose at most a factor of $1+\epsilon$ and the squares assigned to the blocks can be packed in a greedy manner. Thus, we obtain a globally feasible solution $\tilde{I}$ with $p(\tilde{I}) \geq (1+O(\epsilon))^{-1} \sum_j p(\tilde{I}_j) \geq (1+O(\epsilon))^{-1} \mathrm{OPT}$.

THEOREM 5.1. *There is a $(1+\epsilon)$-approximation algorithm for the two-dimensional knapsack problem for squares with a running time of $2^{2^{2^{O(1/\epsilon^4)}}} \cdot n^{O(1)} = O_\epsilon(1) \cdot n^{O(1)}$.*

Proof. We will now discuss step by step how our algorithm proceeds and which running time each step incurs.

Step 1: Guessing large squares and number of cells, frames and subframes. As described in Section 3, we guess B in time $O(F(\epsilon))$, then we guess the at most B large squares in time $\left(\frac{1}{\log(1+\epsilon)}\right)^{O(F(\epsilon))} \cdot n$.

Guessing the number of cells (block cells and elongated cells) takes time $O(1/\epsilon^{10} F(\epsilon)^3)$, guessing the number of frames for each elongated cell takes total time $(1/\epsilon)^{O(1/\epsilon^{10} F(\epsilon)^3)}$, and guessing the number of subframes for each frame takes total time $(1/\epsilon)^{O(1/\epsilon^{14} F(\epsilon)^3)}$. This gives a total running time of $\left(\frac{1}{\epsilon \log(1+\epsilon)}\right)^{O(1/\epsilon^{14} F(\epsilon)^3)} \cdot n$ for this step.

Step 2: Guessing dimensions of blocks. As described in Section 4, we first guess $\tilde{\alpha}$ in time $\frac{n \log n}{\log(1+\epsilon)}$. Guessing the height and width of one block then takes time $O(\frac{\log n}{\log(1+\epsilon)})$. The number of blocks in the whole knapsack is the initial number of elongated cells in the knapsack (i.e. before application of Lemma 5.2, as these do not create new block subframes although they increase the number of elongated cells) times $O(1/\epsilon^2)$ (as in each frame we might have at most one block subframe) plus the number of block cells, i.e., in total we have at most $O(1/\epsilon^2 F(\epsilon) + F(\epsilon)) = O(1/\epsilon^2 F(\epsilon))$

blocks. Therefore, the total time for step 2 is $\frac{n \log n}{\log(1+\epsilon)} \cdot (\frac{\log n}{\log(1+\epsilon)})^{O(1/\epsilon^2 F(\epsilon))} = n \cdot (\frac{\log n}{\log(1+\epsilon)})^{O(1/\epsilon^2 F(\epsilon))} = n^2 \cdot (\frac{1}{\log(1+\epsilon)})^{O(1/\epsilon^2 F(\epsilon))}$.

Step 3: Guessing $\bar{w}_{C,\ell,j}$ and $f(C,\ell,j)$ for row subframes. After Lemma 5.2, the number of row subframes is at most $O(1/\epsilon^{10} F(\epsilon)^3)$. For $\bar{w}_{C,\ell,j}$ there are $\log_{1+\epsilon} n$ many possibilities, for $f(C,\ell,j)$ there are $O(1/\epsilon^4 F(\epsilon)^2)$ many possibilities as described in the proof of Lemma 5.3, and hence we need time $\left(\frac{\log n}{\log(1+\epsilon)} O(1/\epsilon^4 F(\epsilon)^2)\right)^{O(1/\epsilon^{10} F(\epsilon)^3)} = n \cdot (\frac{1/\epsilon F(\epsilon)}{\log(1+\epsilon)})^{O(1/\epsilon^{10} F(\epsilon)^3)}$ for this step.

Step 4: Guessing area of square groups and number of squares in blocks. In the next step, we want to guess the quantities $a_{B,r}$ and $n_{B,r}$ as described in Lemma 5.4 for all blocks B and all groups I_r (number of blocks is at most $O(1/\epsilon^2 F(\epsilon))$ and number of groups is at most $O(1/\epsilon^4 F(\epsilon)^2)$). For each block B and group I_r, we have $O(\frac{\log n}{\log(1+\epsilon)})$ many possibilities, hence the total time for step 4 is $(\frac{\log n}{\log(1+\epsilon)})^{O(1/\epsilon^6 F(\epsilon)^3)} = n \cdot (\frac{1}{\log(1+\epsilon)})^{O(1/\epsilon^6 F(\epsilon)^3)}$.

Step 5: Guessing the values k_r. There are $O(1/\epsilon^4 F(\epsilon)^2)$ values to guess. In order to guess each value, we first need to compute the function $\bar{p}_r(x)$, or, to be more precise, for each square size we need to evaluate the function at this point. Evaluating the function at one point takes time $(n \cdot 1/\epsilon \cdot F(\epsilon))^{O(1)}$ for solving the LP and time $(n \cdot 1/\epsilon \cdot F(\epsilon))^{O(1)}$ for finding the integral matching (as the graph has at most $O(1/\epsilon^{10} F(\epsilon)^3 \cdot n)$ many nodes). Thus guessing all k_r-values can be done in time $n^{O(1)} \cdot O(F(\epsilon)/\epsilon)^{O(1)} \cdot (\frac{\log n}{\log(1+\epsilon)})^{O(1/\epsilon^4 F(\epsilon)^2)} = n^{O(1)} \cdot (\frac{F(\epsilon)}{\epsilon \log(1+\epsilon)})^{O(1/\epsilon^4 F(\epsilon)^3)}$.

Step 6: Checking whether cells and large squares can be packed into the knapsack. We need to enumerate all possible packings of these rectangular regions into the knapsack. When packing L rectangles into one rectangle, we need running time $L^{O(L)}$, hence in this case time $(F(\epsilon) + 1/\epsilon^{10} F(\epsilon)^3)^{O(F(\epsilon)+1/\epsilon^{10} F(\epsilon)^3)} = (F(\epsilon)/\epsilon)^{O(1/\epsilon^{10} F(\epsilon)^3)}$.

Step 7: Packing squares into blocks and row subframes. Notice that, in contrast to Lemma 4.7, we do not know beforehand which squares can go into which blocks, as some squares might be small compared to one block (i.e., the square's size is at most an ϵ^2 fraction of the block's side lengths) but not compared to another block. However, a feasible assignment of squares to blocks is given by the integral matching computed as described in the proof of Lemma 5.5, as well as an assignment of squares to the row subframes. We simply pack squares into the corresponding row subframes, and squares assigned to blocks are packed using NFDH (running time $n^{O(1)}$). We know from the matching solution and Lemma 5.4 that the area of the selected squares exceeds the area of the block by at most a factor $1+\epsilon$. We can apply Lemma 4.2 to ensure that we pack a subset of these squares of profit at least $1 - O(\epsilon)$ times the total profit of assigned squares into each block. □

6 Resource Augmentation in Both Dimensions.

In this section we present a technique for two-dimensional knapsack in the setting of $(1+\epsilon)$-resource augmentation in both dimensions. In contrast to the previous sections, here we allow the input objects even to be axis-parallel rectangles, rather than only squares. For each input rectangle i denote by $h_i \in \mathbb{N}$ its height and by $w_i \in \mathbb{N}$ its width. We reduce the running time of the known PTAS [9] of $\Omega\left(n^{1/\epsilon^{1/\epsilon}}\right)$ to $O_\epsilon(1) \cdot n^{O(1)}$ and compute even a solution whose profit is at least $p(OPT)$, instead of a $(1+\epsilon)$-approximative solution. We investigate first the case without the possibility to rotate the rectangles.

6.1 Rectangle classification.

We classify rectangles into large, medium, horizontal, vertical, and small rectangles according to their side lengths, using constants $\mu, \delta > 0$ defined below. We will use a separate routine for the medium rectangles. Using the next lemma we ensure that they have small total area in the optimal solution.

LEMMA 6.1. *There is a universal set $D = \{(\mu_0, \delta_0), (\mu_1, \delta_1), ..., (\mu_{1/\epsilon}, \delta_{1/\epsilon})\}$ with $\mu_i = \epsilon^2 \cdot \delta_i^3 < \epsilon$ and $\mu_i, \delta_i \in \Omega_\epsilon(1)$ for each $(\mu_i, \delta_i) \in D$ such that for each input there exists a pair $(\mu, \delta) \in D$ and the total area of rectangles $i \in OPT$ with $\mu \cdot N < w_i < \delta \cdot N$ or $\mu \cdot N < h_i < \delta \cdot N$ is bounded by $2\epsilon \cdot N^2$.*

Proof. Our construction is similar to [9]. Define $\delta_j := \epsilon^{2 \cdot 3^j - 1}$ and $\mu_j := \delta_{j+1} = \epsilon^{2 \cdot 3^{j+1} - 1}$ for $0 \le j \le 1/\epsilon$. Let $M_j^V := \{i \in OPT | \mu_j N < h_i < \delta_j N\}$ and $M_j^H := \{i \in OPT | \mu_j N < w_i < \delta N\}$. It holds that $M_j^V \cap M_k^V = \emptyset$ and $M_j^H \cap M_k^H = \emptyset$ for $j \neq k$, as the intervals (μ_j, δ_j) are disjoint. Therefore, each rectangle from the optimal solution occurs in at most one of the sets M_j^H and at most one of the sets M_j^V, and thus $\sum_{j=1}^{1/\epsilon} a(M_j^H \cup M_j^V) \le 2N^2$ as the total area of all rectangles in OPT is at most N^2 (we denote by $a(S)$ the total area of the rectangles in set S). Therefore, there is one value j such that $a(M_j^H \cup M_j^V) \le 2\epsilon N^2$. □

We guess the correct pair $(\mu, \delta) \in D$ due to Lemma 6.1. Note that μ might be as small as $\epsilon^{O(1/\epsilon)}$. We denote by L the set of all rectangles i with $w_i > \delta \cdot N$ and $h_i > \delta \cdot N$, by V all rectangles i with $w_i < \mu \cdot N$ and

$h_i > \delta \cdot N$, by H all rectangles i with $w_i > \delta \cdot N$ and $h_i < \mu \cdot N$, and by S all rectangles i with $w_i < \mu \cdot N$ and $h_i < \mu \cdot N$. We define $M := I \setminus (L \cup V \cup H \cup S)$, i.e., all rectangles i with $\mu \cdot N < w_i < \delta \cdot N$ or $\mu \cdot N < h_i < \delta \cdot N$.

We treat the rectangles in M separately using the following lemma and pack them into the additional space gained by increasing the size of the knapsack (a similar argumentation was used in [2]). The intuition is that increasing the size of the knapsack by a factor $1 + O(\epsilon)$ gains free space that is by a constant factor larger than the total space needed for the medium rectangles. This makes the packing easy.

LEMMA 6.2. *There is a polynomial time algorithm that computes a set $M' \subseteq M$ with $p(M') \geq p(M \cap OPT)$ and a packing for M' into two boxes of sizes $N \cdot O(\epsilon) \times N$ and $N \times O(\epsilon) \cdot N$, respectively.*

Proof. We know from Lemma 6.1 that $a(M \cap OPT) \leq 2\epsilon N^2$ and for all rectangles in M we have $\mu N < w_i < \delta N$ or $\mu N < h_i < \delta N$. Let $M_V := \{i \in M | \mu N < w_i < \delta N\}$ and $M_H := \{i \in M | \mu N < h_i < \delta N\}$. We want to select subsets $M'_V \subseteq M_V$ and $M'_H \subseteq M_H$ s.t. $a(M'_V) \leq 4\epsilon N^2$ and $a(M'_H) \leq 4\epsilon N^2$. Then, we can use the algorithm NFDH from [7], which packs M'_H into an area of width N and height $2a(M'_H)/N + h_{max}$ where h_{max} denotes the maximum height of rectangles in M'_H. As rectangles in M'_H have height at most $\delta N \leq \epsilon N$, we need height at most $2 \cdot 4 \cdot \epsilon N^2/N + \epsilon N = O(\epsilon)N$. Analogously we can pack the rectangles M'_V into an area of height N and width $O(\epsilon)N$.

For selecting the rectangles M'_H, we consider a knapsack problem: For each rectangle $i \in M_H$, we create an item with size equal to the area of i and profit equal to the profit of i. The knapsack has capacity $2\epsilon N^2$. We use the simple greedy Knapsack algorithm with one modification: the last item (that is not taken into the Knapsack because it would exceed the capacity) is also selected. Since this item has size at most $2\epsilon N^2$, we get that all selected items have total size at most $4\epsilon N^2$ (and thus all rectangles from M_H corresponding to these items have total area at most $4\epsilon N^2$). Moreover, the selected rectangles have profit at least $p(M_H \cap OPT)$. The same selection procedure can be applied to rectangles in M_V. □

Lemma 6.2 ensures that $p(M') \geq p(M \cap OPT)$ and thus our medium rectangles are as profitable as the medium rectangles in the optimal solution. In contrast, the algorithm in [9] loses a factor of $1 + \epsilon$ in the approximation ratio in this step. From now on we consider only the rectangles in $L \cup H \cup V \cup S$.

6.2 Placing a grid. Using the argumentation in [9], by enlarging the knapsack by a factor $1+\epsilon$ we can round up the heights of the rectangles in $L \cup V$ and the widths of the rectangles in $L \cup H$ to integral multiples of $\epsilon \cdot \delta \cdot N$. Even more, we can ensure that they are aligned with a grid G of granularity $\epsilon \cdot \delta \cdot N$, see Figure 5. Formally, we define $G := \{\mathbb{R} \times k \cdot \epsilon \cdot \delta \cdot N | k \in \mathbb{N}\} \cup \{k \cdot \epsilon \cdot \delta \cdot N \times \mathbb{R} | k \in \mathbb{N}\}$.

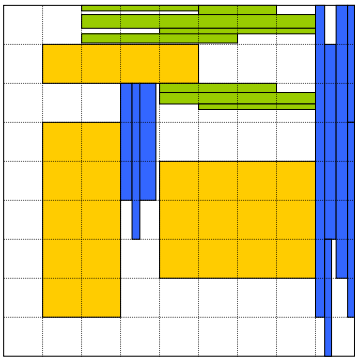


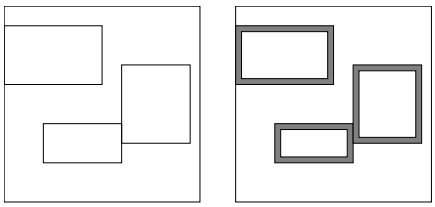
Figure 5: The grid G and large (yellow), horizontal (green) and vertical (blue) rectangles aligned with it.

LEMMA 6.3. ([9]) *By enlarging the knapsack by a factor $1 + \epsilon$ we can assume for the input and the optimal solution that*

- *for each $i \in L \cup H$ we have that w_i and the x-coordinates of all corners of i are integral multiples of $\epsilon \cdot \delta \cdot N$,*
- *for each $i \in L \cup V$ we have that h_i and the y-coordinates of all corners of i are integral multiples of $\epsilon \cdot \delta \cdot N$,*
- *the height and width of the knapsack is an integral multiple of $\epsilon \cdot \delta \cdot N$.*

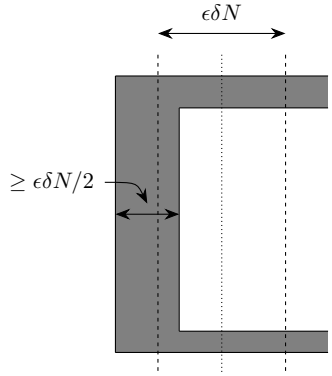
Proof. We follow the argumentation from [9]. Enlarge the knapsack and rectangles in $L \cup H \cup V$ by a factor of $1 + \epsilon$ (the packing is still feasible). Define the *induced space* of some rectangle $i \in L \cup H \cup V$ to be the space that i occupies in this enlarged packing. Reduce the rectangles back to their original size. Consider some rectangle i in $L \cup H$; we will shift i horizontally inside its induced space in order to align it with the grid (the argumentation is analogous for rectangles in $L \cup V$ to align them vertically with the grid). For illustration see Figure 6b. The distance between a vertical boundary of i and the corresponding vertical boundary of its induced space is at least $\epsilon\delta N/2$ as the rectangle has width at least δN. As the grid lines are $\epsilon\delta N$ apart and we can shift i by $\epsilon\delta N/2$ to the left or to the right, we can shift it in either direction s.t. one of its vertical

boundaries is aligned with one of the grid lines (see Figure 6c). Now, we can enlarge the rectangle again by a factor of at most $1+\epsilon$ until its other vertical boundary is aligned with another grid line. In order to do this (without knowing the optimal solution), we simply round all side lengths to the next larger multiple of $\epsilon\delta N$. The enlarged rectangle will still reside inside its induced space. Finally, by enlarging the knapsack by a factor $1+\epsilon$ we can easily ensure that its height and width are integral multiples of $\epsilon \cdot \delta \cdot N$. □



(a) Initial instance.

(b) Instance after enlarging and then shrinking the rectangles and enlarging the knapsack. Gray areas denote the induced spaces.



(c) Shifting of a large or horizontal rectangle. The dashed lines denote grid lines.

Figure 6: For aligning large, horizontal and vertical rectangles with the grid, we first enlarge them, and then shrink them again. The third picture illustrates that we have enough space inside the induced space to align the rectangle with a grid line.

6.3 Packing the rectangles. The grid G divides the knapsack into $1/\delta^2 \leq \left(\frac{1}{\epsilon}\right)^{O(1/\epsilon)} = O_\epsilon(1)$ grid cells. In the packing due to Lemma 6.3, each such cell is either fully covered by a large rectangle or does not intersect a large rectangle at all. At this point, the algorithm in [9] guesses the large rectangles directly which leads to a running time of $\binom{n}{1/\delta^2}$ which might be as large $\Omega\left(n^{1/\epsilon^{1/\epsilon}}\right)$ since δ can be as small as $\epsilon^{1/\epsilon}$. Instead, we now guess the boundaries of the large rectangles in this packing, without guessing the rectangles themselves. Since there are $O_\epsilon(1)$ grid cells, this can be done in time $O_\epsilon(1)$. Given the boundaries, we compute the best choice for the large rectangles using a greedy algorithm.

LEMMA 6.4. *Given the boundaries of the large rectangles in the optimal solution, we can compute a set $L' \subseteq L$ with $p(L') \geq p(L \cap OPT)$ in time $O_\epsilon(n)$ that fits into the space given by these boundaries.*

Proof. We consider the boundaries one by one in arbitrary order. For each boundary, search for the most profitable rectangle in L that fits into this space, pack it there, and remove it from the list of candidates for the next boundaries. Clearly, the optimal solution cannot pack any more profitable rectangles into these boundaries. □

Each remaining grid cell (i.e., that is not covered by a large rectangle) is either intersected by a horizontal rectangle, or by a vertical rectangle, or by no rectangle in $H \cup V$ at all. First, we guess the total area of the selected horizontal and vertical rectangles in the optimal solution. Then, as in [8, 9], we use techniques from strip packing to compute a profitable packing for the horizontal and vertical rectangles while enlarging the knapsack slightly.

LEMMA 6.5. *When enlarging the knapsack by a factor $1+O(\epsilon)$ there is an algorithm with running time $O_\epsilon(1) \cdot n^{O(1)}$ that computes sets $H' \subseteq H$ and $V' \subseteq V$ and a packing for them in the space of the knapsack that is not occupied by rectangles in L' such that $p(H') \geq p(H \cap OPT)$, $p(V') \geq p(V \cap OPT)$, $a(H' \cup V') \leq a((H \cup V) \cap OPT) + O(\epsilon) \cdot N$ and the space of the knapsack not used by rectangles in $L' \cup V' \cup H'$ can be partitioned into $O\left(\left(\frac{1}{\epsilon\delta}\right)^2\right) = O_\epsilon(1)$ rectangular boxes.*

Proof. Our reasoning essentially reiterates the argumentation in [8, 9], we refer to these papers for more details. Consider the horizontal rectangles H. They have up to $\frac{1}{\epsilon \cdot \delta}$ many widths and their minimum width is $\epsilon \cdot \delta$. For each width class we guess the total height of the rectangles in OPT of this width in integral multiples of $\epsilon^2 \cdot \delta \cdot N$, i.e., for each width class we guess an integer $k \in O(\frac{1}{\epsilon^3 \cdot \delta^2})$ such that the guessed height equals $k \cdot \epsilon^2 \delta \cdot N$. By increasing the size of the knapsack by a factor $1+\epsilon$ this discretization is justified since then we gain additional space of width N and height $\epsilon \cdot N$ which is enough to accommodate one rectangle of height $\epsilon^2 \delta \cdot N$ of each width class. Also, we can restrict to values of k

with $k \in O(\frac{1}{\epsilon^3 \cdot \delta^2})$ since for larger values the total area of the rectangles in this width class would be larger than the size of the knapsack. For each i we denote by A_i the guessed total height of the rectangles of width $i \cdot \epsilon\delta$. We do a similar operation for the rectangles in V. Note that for the guesses there are only $O_\epsilon(1)$ possibilities.

Then we guess which of the $O(\epsilon^2 \cdot \delta^2)$ grid cells are used by horizontal rectangles and which are used by vertical rectangles. Note that a grid cell can be used by only one of these two types of rectangles so the guessing can be done in time $2^{O(\epsilon^2 \cdot \delta^2)} = O_\epsilon(1)$. Consider the cells that we guessed to be used by horizontal rectangles. For each grid row we partition them into sets of connected components. We call these connected components *blocks*. Then we use a configuration-LP to pack horizontal rectangles into the blocks. Each configuration is a vector $(b_1, ..., b_{1/(\epsilon\delta)})$ with $\sum_i b_i \cdot i \leq 1/(\epsilon\delta)$ that specifies b_i slots for rectangles of width $b_i \cdot i$ for each i. For each block there are $(\frac{1}{\epsilon \cdot \delta})^{\frac{1}{\epsilon \cdot \delta}}$ many configurations with at most $1/(\epsilon\delta)$ slots each. Based on this we formulate a configuration-LP that ensures that

- in each block the total height of assigned configurations is at most the height of the block, i.e., $N \cdot \epsilon \cdot \delta$, and
- in all horizontal blocks B for each rectangle width $i \cdot \epsilon\delta$ there are slots whose total height is at least A_i.

The resulting LP has $O_\epsilon(1)$ variables and constraints and we compute an extreme point solution for it. The number of constraints in the LP is bounded by the number of blocks plus the number of rectangle widths, i.e., by $2\left(\frac{1}{\epsilon\delta}\right)^2$, and thus the number of non-zero entries in our solution is bounded by the same value. This yields a partition of the blocks for horizontal rectangles into at most $2\left(\frac{1}{\epsilon\delta}\right)^3$ slots, each slot corresponding to one rectangle width. We fill now the rectangles in H fractionally into these slots. We do this in a greedy manner, i.e., for each rectangle width we order the rectangles non-increasingly by density p_i/h_i and assign them greedily in this order into the slots (the slots are ordered arbitrarily). If a rectangle does not fit entirely into the remaining space in a considered slot, we split the rectangle horizontally and assign one part of it to the current slot and the remainder to the next slot (we iterate this procedure if the remaining part does not fit into the next slot). The obtained profit of the rectangles assigned in this way is at least $p(H \cap OPT)$ (also counting the whole profit of the last rectangle that might be only partially assigned). At the end, there are at most $2\left(\frac{1}{\epsilon\delta}\right)^3$ rectangles that are split. Their total height is bounded by $\mu N \cdot 2\left(\frac{1}{\epsilon\delta}\right)^3 \leq O(\epsilon) \cdot N$ and thus we can place them into an additional block of height $O(\epsilon) \cdot N$ and width N. By enlarging the size of the knapsack by a factor $1+\epsilon$ we gain additional free space of this size. By construction, the empty space in the so far considered blocks can be partitioned into at most $O\left(\left(\frac{1}{\epsilon\delta}\right)^2\right)$ rectangular boxes. We do the same operation for the vertical rectangles in V. □

Finally, we add the small rectangles. We select a set of small rectangles whose total area is at most the area of the remaining space which yields an instance of one-dimensional Knapsack. Using that the rectangles are all very small and that we can increase the capacity of our knapsack, we can select small rectangles that are as profitable as the small rectangles in OPT and we can find a packing for all the selected rectangles. In contrast, in this step the algorithm in [9] loses a factor $1+\epsilon$ in the approximation ratio (instead of increasing the size of the knapsack).

Lemma 6.6. *When enlarging the knapsack by a factor $1+\epsilon$ there is a polynomial time algorithm that computes a set of rectangles $S' \subseteq S$ with $p(S') \geq p(S \cap OPT)$ and a packing for S' into the remaining space that is not used by the rectangles in $L' \cup H' \cup V'$.*

Proof. Consider the free space in the knapsack that is not used by $L' \cup H' \cup V'$. According to Lemma 6.5, this space can be partitioned into $O\left(\left(\frac{1}{\epsilon\delta}\right)^2\right)$ rectangular areas; from now on we will call them *boxes*.

Our goal is to find a set of rectangles $S' \subseteq S$ which we can pack into the free space and the area that we gain by resource augmentation, and we want to have $p(S') \geq p(S \cap OPT)$.

To this end we define an instance of the one-dimensional knapsack problem: For each rectangle $i \in S$, create an item i' for the knapsack instance with profit equal to the profit of i and size equal to the area of i. The capacity of the knapsack is the total area of all boxes; denote this quantity by A. For solving this problem we use the simple greedy algorithm for knapsack. This algorithm sorts the items in non-increasing order of their profit-to-size-ratios, and then greedily packs items into the knapsack in this order until an item does not fit. We put all rectangles corresponding to items selected by this algorithm into S' and we also add into S' the rectangle corresponding to the first item that is not taken into the knapsack by the greedy algorithm because it would exceed the capacity A. It has size at most $\mu^2 N^2$. The profit of S' is now at least that of the optimal solution (which is only allowed to use capacity A). This is the case since our profit is at

least the profit of the optimal solution to the canonical LP-relaxation for our knapsack instance.

From now on, we use a standard argumentation to place the items in S' greedily into the remaining space (see e.g., [19]): We ignore all boxes that are too small, i.e. boxes that have one side length $\leq \mu N$. Let k be the number of boxes that are large enough and $\overline{k}$ be the number of boxes that are too small, $k, \overline{k} \leq O\left(\left(\frac{1}{\epsilon\delta}\right)^2\right)$. The area of a box that is too small is at most μN^2 (its larger side might be as large as N), and thus we lose in total an area of at most $\overline{k}\mu N^2$ by this.

We fill the rectangles S' into all other boxes using NFDH. The total area that is available for this is at least $A - k\mu N^2$. Let $B_1, \ldots, B_k$ be the boxes in the order they are used by NFDH. Let $\overline{w}_j, \overline{h}_j$ be the side lengths of B_j and let i_j be the first rectangle packed into B_j (by our constraint on the size of the boxes used we know that at least one rectangle fits into each box). If all rectangles were packed into the boxes, we are done, so assume this is not the case, i.e. some rectangles remain unpacked. For box B_j, the NFDH-algorithm packs rectangles into a sub-box $\hat{B}_j$ of size $\overline{w}_j \times \hat{h}_j$, i.e. a sub-box of height $\overline{h}_j - \hat{h}_j$ and width $\overline{w}_j$ is unused by the algorithm (see Figure 7 for illustration). This means that $\overline{h}_j - \hat{h}_j \leq \mu N$ (otherwise it would be possible to pack another rectangle there). In [7] it is proven that the wasted space inside $\hat{B}_j$ is at most $h_{i_j} \cdot \overline{w}_j$. Thus, the total unused space inside B_j is at most $h_{i_j} \cdot \overline{w}_j + \mu N \cdot \overline{w}_j \leq 2\mu N^2$ and the unused space in all boxes is at most $2\mu N^2 \cdot k$. The total unused space in the whole knapsack (i.e. the space not used within $B_1, \ldots, B_k$ and the space in the small boxes) is therefore at most $2k\mu N^2 + \overline{k}\mu N^2 = O\left(\frac{1}{\epsilon^2\delta^2}\right)\mu N^2 = O(\epsilon)N^2$. In addition, we need additional space because the rectangles in S' have area larger than A, but this additional space is again in $O(\epsilon)N^2$. By again using NFDH and the bounds derived in [7], we can pack all remaining rectangles into an area of width N and height $\mu N + 2 \cdot O(\epsilon)N^2/N = O(\epsilon)N$ gained by enlarging the size of the knapsack by a factor $1 + \epsilon$. □

Overall, we obtain a solution whose profit is at least $p(OPT)$ and which is packed into a knapsack of size $(1 + O(\epsilon))N \times (1 + O(\epsilon))N$. If it is allowed to rotate rectangles by 90 degrees then it is straight forward to adjust Lemmas 6.4 and 6.5 accordingly. Also, Lemma 6.6 still holds, even if we do not rotate any rectangle in S (while OPT might still do this).

THEOREM 6.1. *For any $\epsilon > 0$ there is an algorithm with a running time of $O_\epsilon(1) \cdot n^{O(1)}$ for the two-dimensional knapsack problem for rectangles under $(1 + \epsilon)$-resource augmentation in both dimensions that computes a solution whose profit is at least $p(OPT)$. This holds for the settings with and without rotation.*

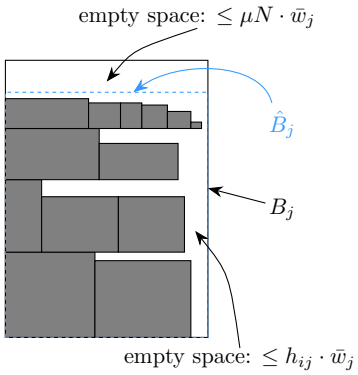


Figure 7: Packing produced by NFDH inside one box.

References

[1] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 1681–1697, 2015.

[2] Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the two-dimensional geometric knapsack problem. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 1491–1505. SIAM, 2015.

[3] Nikhil Bansal, Alberto Caprara, Klaus Jansen, Lars Prädel, and Maxim Sviridenko. A structural lemma in 2-dimensional packing, and its implications on approximability. In *Algorithms and Computation (ISAAC 2009)*, volume 5878 of *LNCS*, pages 77–86. Springer, 2009.

[4] Nikhil Bansal, Jose R Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31:31–49, 2006.

[5] Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 13–25. SIAM, 2014.

[6] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97. IEEE Computer Society, 2015.

[7] Edward G. Coffman, Jr, Michael R. Garey, David S. Johnson, and Robert E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9:808–826, 1980.

[8] José R Correa and Claire Kenyon. Approximation schemes for multidimensional packing. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 186–195. Society for Industrial and Applied Mathematics, 2004.

[9] Aleksei Fishkin, Olga Gerber, Klaus Jansen, and Roberto Solis-Oba. On packing rectangles with resource augmentation: Maximizing the profit. *Algorithmic Operations Research*, 3, 2008.

[10] Rolf Harren, Klaus Jansen, Lars Prädel, and Rob van Stee. A $(5/3 + \epsilon)$-approximation for strip packing. *Comput. Geom.*, 47(2):248–267, 2014.

[11] Wiebke Höhn, Julián Mestre, and Andreas Wiese. How unsplittable-flow-covering helps scheduling with job-dependent cost functions. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 625–636. Springer, 2014.

[12] Klaus Jansen and Roberto Solis-Oba. New approximability results for 2-dimensional packing problems. In *Mathematical Foundations of Computer Science (MFCS 2007)*, volume 4708 of *LNCS*, pages 103–114. Springer, 2007.

[13] Klaus Jansen and Roberto Solis-Oba. A polynomial time approximation scheme for the square packing problem. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Integer Programming and Combinatorial Optimization, 13th International Conference, IPCO 2008, Bertinoro, Italy, May 26-28, 2008, Proceedings*, volume 5035 of *Lecture Notes in Computer Science*, pages 184–198. Springer, 2008.

[14] Klaus Jansen and Roberto Solis-Oba. Packing squares with profits. *SIAM J. Discrete Math.*, 26(1):263–279, 2012.

[15] Klaus Jansen and Guochuan Zhang. Maximizing the number of packed rectangles. In *Algorithm Theory (SWAT 2004)*, volume 3111 of *LNCS*, pages 362–371. Springer, 2004.

[16] Klaus Jansen and Guochuan Zhang. On rectangle packing: maximizing benefits. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, pages 204–213. SIAM, 2004.

[17] Claire Kenyon and Eric Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25(4):645–656, 2000.

[18] László Lovász and Michael D. Plummer. *Matching Theory*. Akadémiai Kiadó - North Holland, Budapest, 1986.

[19] Giorgi Nadiradze and Andreas Wiese. On approximating strip packing with a better ratio than 3/2. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1491–1510. SIAM, 2016.

[20] David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1-3):461–474, 1993.