Algorithmica (2023) 85:2735–2778
https://doi.org/10.1007/s00453-023-01116-0

Tight Approximation Algorithms for Geometric Bin Packing with Skewed Items

Arindam Khan[1] · **Eklavya Sharma**[1,2]

Received: 18 January 2022 / Accepted: 7 March 2023 / Published online: 28 March 2023

Abstract

In *Two-dimensional Bin Packing* (*2BP*), we are given n rectangles as input and our goal is to find an axis-aligned nonoverlapping packing of these rectangles into the minimum number of unit square bins. 2BP admits no APTAS and the current best approximation ratio is 1.406 by Bansal and Khan (ACM-SIAM symposium on discrete algorithms (SODA), pp 13–25, 2014. https://doi.org/10.1137/1.9781611973402.2). A well-studied variant of 2BP is *Guillotine Two-dimensional Bin Packing* (*G2BP*), where rectangles must be packed in such a way that every rectangle in the packing can be obtained by applying a sequence of end-to-end axis-parallel cuts, also called *guillotine cuts*. Bansal et al. (Symposium on foundations of computer science (FOCS). IEEE, pp 657–666, 2005. https://doi.org/10.1109/SFCS.2005.10) gave an APTAS for G2BP. Let λ be the smallest constant such that for every set I of items, the number of bins in the optimal solution to G2BP for I is upper bounded by $\lambda \operatorname{opt}(I) + c$, where $\operatorname{opt}(I)$ is the number of bins in the optimal solution to 2BP for I and c is a constant. It is known that $4/3 \leq \lambda \leq 1.692$. Bansal and Khan (2014) conjectured that $\lambda = 4/3$. The conjecture, if true, will imply a $(4/3 + \varepsilon)$-approximation algorithm for 2BP. Given a small constant $\delta > 0$, a rectangle is called *large* if both its height and width are at least δ, else it is called *skewed*. We make progress towards the conjecture by showing that $\lambda = 4/3$ when all input rectangles are skewed. We also give an APTAS for 2BP for skewed items, though general 2BP does not admit an APTAS.

Arindam Khan is supported by Pratiksha Trust Young Investigator Award and Google ExploreCSR grant. Eklavya Sharma did this work when he was a student at the Indian Institute of Science. A preliminary version of this paper appeared in the proceedings of the International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) 2021 [1].

✉ Eklavya Sharma
eklavya2@illinois.edu

Arindam Khan
arindamkhan@iisc.ac.in

1 Indian Institute of Science, Bangalore, India

2 University of Illinois at Urbana-Champaign, Champaign, IL, USA

Keywords Geometric bin packing · Guillotine cuts · Approximation algorithms

1 Introduction

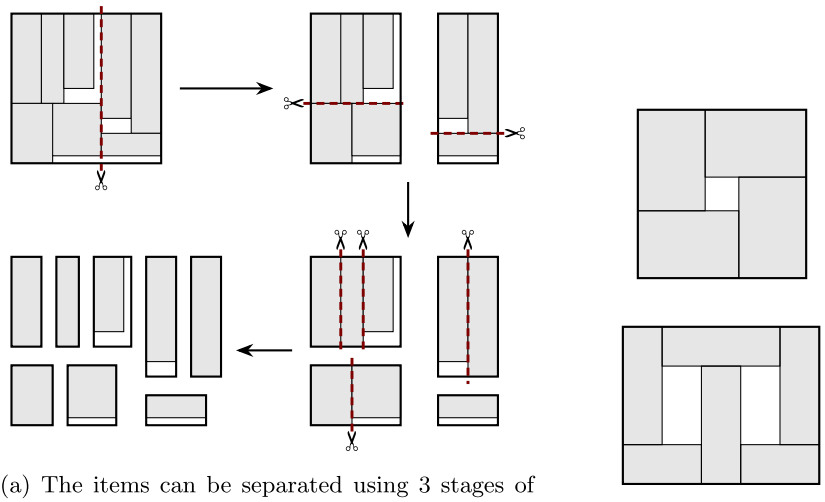
Two-dimensional Bin Packing (2BP) is a well-studied problem in combinatorial optimization. It finds numerous applications in logistics, databases, and cutting stock. In 2BP, we are given a set of n rectangular items and square bins of side length 1. The ith item is characterized by its width $w(i) \in (0, 1]$ and height $h(i) \in (0, 1]$. Our goal is to find an axis-aligned nonoverlapping packing of these items into the minimum number of square bins of side length 1. There are two well-studied variants: (i) where the items cannot be rotated, and (ii) they can be rotated by 90°.

As is conventional in bin packing, we focus on asymptotic approximation algorithms. For any optimization problem, the asymptotic approximation ratio (AAR) of algorithm $\mathcal{A}$ is defined as $\lim_{m\to\infty} \sup_{I:\mathrm{opt}(I)=m}(\mathcal{A}(I)/\mathrm{opt}(I))$, where $\mathrm{opt}(I)$ is the optimal objective value and $\mathcal{A}(I)$ is the objective value of the solution output by algorithm $\mathcal{A}$, respectively, on input I. Intuitively, AAR captures the algorithm's behavior when $\mathrm{opt}(I)$ is large. We call a bin packing algorithm α-asymptotic-approximate iff its AAR is at most α. An Asymptotic Polynomial-Time Approximation Scheme (APTAS) is an algorithm that accepts a parameter ε and has an AAR of $(1 + \varepsilon)$.

2BP is a generalization of the classical 1-D bin packing problem [2, 3]. However, unlike 1-D bin packing, 2BP does not admit an APTAS unless P = NP [4]. In 1982, Chung et al. [5] gave an approximation algorithm with AAR 2.125 for 2BP. Caprara [6] obtained a $T_\infty(\approx 1.691)$-asymptotic-approximation algorithm. Bansal et al. [7] introduced the Round and Approx framework to obtain an AAR of $1 + \ln(T_\infty)$ ($\approx$ 1.525). Then Jansen and Praedel [8] obtained an AAR of 1.5. The present best AAR is $1 + \ln(1.5)$ ($\approx$ 1.405), due to Bansal and Khan [9], and works for both the cases with and without rotations. The best lower bounds on the AAR for 2BP are 1 + 1/3792 and 1 + 1/2196 [10], for the versions with and without rotations, respectively.

In the context of geometric packing, guillotine cuts are well-studied and heavily used in practice [11]. The notions of *guillotine cuts* and *k-stage packing* were introduced by Gilmore and Gomory in their seminal paper [12] on the cutting stock problem. A packing of items into a bin is called a k-stage packing if the items can be separated from each other using k stages of axis-parallel end-to-end cuts, also called guillotine cuts, where in each stage, either all cuts are vertical or all cuts are horizontal. In each stage, each rectangular region obtained in the previous stage is considered separately and can be cut again using guillotine cuts. Note that in the cutting process we change the orientation (vertical or horizontal) of the cuts $k - 1$ times. In the k-stage bin packing problem, we need to pack the items into the minimum number of bins such that the packing in each bin is a k-stage packing. 2-stage packing, also called *shelf packing*, has been studied extensively. A packing is called *guillotinable* iff it is a k-stage packing for some integer k. See Fig. 1 for examples. Caprara et al. [13] gave an APTAS for 2-stage 2BP. Bansal et al. [14] showed an APTAS for guillotine 2BP.

The presence of an APTAS for guillotine 2BP raises an important question: can the optimal solution to guillotine 2BP be used as a good approximate solution to 2BP? Formally, let $\mathrm{opt}(I)$ and $\mathrm{opt}_g(I)$ be the minimum number of bins and the minimum



(a) The items can be separated using 3 stages of guillotine cuts, so this is a guillotinable packing.

(b) Two non-guillotinable bins.

Fig. 1 Examples of guillotinable and non-guillotinable packing

number of guillotinable bins, respectively, needed to pack items I. Let λ be the smallest constant such that for some constant c and for every set I of items, we get $\text{opt}_g(I) \leq \lambda\,\text{opt}(I) + c$. Then λ is called the Asymptotic Price of Guillotinability (APoG). It is easy to show that APoG $\geq 4/3$.[1] Bansal and Khan [9] conjectured that APoG $= 4/3$. If true, this would imply a $(4/3+\varepsilon)$-asymptotic-approximation algorithm for 2BP [14]. However, the present upper bound on APoG is only T_∞ (≈ 1.691), due to Caprara's HDH algorithm [6] for 2BP, which produces a 2-stage packing.

Nearly-optimal algorithms are known for some special cases of 2BP, such as when all items are squares [4] or when all rectangles are very small in both dimensions [15] (see Lemma 40 in Appendix C). Another important class is *skewed* rectangles. We say that a rectangle is δ-large if, for some constant $\delta > 0$, its width and height are more than δ; otherwise, the rectangle is δ-skewed. We just say that a rectangle is large or skewed when δ is clear from the context. An instance of 2BP is skewed if all the rectangles in the input are skewed. Skewed instances are important in geometric packing (see Sect. 1.1). This special case is practically relevant [16]: e.g., in scheduling, it captures scenarios where no job can consume a significant amount of a shared resource (energy, memory space, etc.) for a significant amount of time. Even for skewed instance for 2BP, the best known AAR is 1.406 [9]. Also, for skewed instance, the best known upper bound on APoG is $T_\infty \approx 1.691$.

1.1 Related Works

Multidimensional packing and covering problems are fundamental in combinatorial optimization [17]. Vector packing (VP) is another variant of bin packing, where the

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

input is a set of vectors in $[0, 1]^d$ and the goal is to partition the vectors into the minimum number of parts (bins) such that in each part, the sum of vectors is at most 1 in every coordinate. The present best approximation algorithm attains an AAR of $(0.807 + \ln(d + 1))$ [18] and there is a matching $\Omega(\ln d)$-hardness [19]. Generalized multidimensional packing [20, 21] generalizes both geometric and vector packing.

In two-dimensional strip packing (2SP) [15, 22], we are given a set of rectangles and a bounded width strip. The goal is to obtain an axis-aligned nonoverlapping packing of all rectangles such that the height of the packing is minimized. The best-known approximation ratio for 2SP is $5/3 + \varepsilon$ [23] and it is NP-hard to obtain better than 3/2-approximation. However, there exist APTASes for the problem, for both the cases with and without rotations [24, 25]. In two-dimensional knapsack (2GK) [26], the rectangles have associated profits and our goal is to pack the maximum profit subset into a unit square knapsack. The present best polynomial-time (resp. pseudopolynomial-time) approximation ratio for 2GK is 1.809 [27] (resp. 4/3 [28]). These geometric packing problems have also been studied for d dimensions ($d \geq 2$) [29].

2SP and 2GK are also well-studied under guillotine packing. Seiden and Woeginger [30] gave an APTAS for guillotine 2SP. Khan et al. [31] recently gave a pseudopolynomial-time approximation scheme for guillotine 2GK.

Recently, guillotine cuts [32] have received attention due to their connection with the maximum independent set of rectangles (MISR) problem [33]. In MISR, we are given a set of possibly overlapping rectangles and the goal is to find the maximum cardinality set of rectangles so that there is no pairwise overlap. It was noted in [34, 35] that for any set of n non-overlapping axis-parallel rectangles, if there is a guillotine cutting sequence separating αn of them, then it implies a $1/\alpha$-approximation for MISR. Recently there has been progress on MISR [36, 37] based on geometric decomposition using cuts that can be considered a generalization of guillotine cuts. Guillotine cuts have only one segment and divide a rectangle into two sub-rectangles, whereas the cuts in [36, 37] divide a rectilinear region into $O(1)$ sub-regions, each with $O(1)$ number of edges.

Skewed instance is an important special case in these problems. In some problems, such as MISR and 2GK, if all items are δ-large then we can solve them exactly in polynomial time. So, the inherent difficulty of these problems lies in instances containing skewed items. For VP, hard instances are again skewed, e.g., Bansal et al. [18] showed that hard instances for 2-D VP (for a class of algorithms called *rounding based algorithms*) are skewed instances, where one dimension is $1 - \varepsilon$ and the other dimension is ε. Galvez el al. [16] recently studied strip packing when all items are skewed. For skewed instances, they showed $(3/2 - \varepsilon)$ hardness of approximation and a matching $(3/2 + \varepsilon)$-approximation algorithm. For 2GK, when the height of each item is at most ε^3, a $(1 - 72\varepsilon)^{-1}$-approximation algorithm is known [38].

1.2 Our Contributions

We study 2BP for the special case of δ-skewed rectangles, where $\delta \in (0, 1/2]$ is a constant.

First, we make progress towards the conjecture [9] that APoG $= 4/3$. Even for skewed rectangles, we only knew APoG $\leq T_\infty (\approx 1.691)$. We resolve the conjecture for skewed rectangles, by giving lower and upper bounds of roughly $4/3$ when δ is a small constant.

Specifically, in Sect. 3, we give an algorithm for 2BP, called $\texttt{skewed4Pack}_\varepsilon$, that takes a parameter $\varepsilon \in (0, 1/2]$ as input. For a set I of δ-skewed rectangles, we show that when δ and ε are close to 0, $\texttt{skewed4Pack}_\varepsilon(I)$ outputs a 4-stage packing of I into roughly $4\,\mathrm{opt}(I)/3 + O(1)$ bins.

Theorem 1 *Let I be a set of δ-skewed items, where $\delta \in (0, 1/2]$. Then* $\texttt{skewed4Pack}_\varepsilon(I)$ *outputs a 4-stage packing of I in time $O(n(\log n + \delta/\varepsilon)) + O_\varepsilon(1)$, where $n := |I|$, and the number of bins used is less than $(4/3)(1 + 8\delta)(1 + 6\varepsilon)\,\mathrm{opt}(I) + 12\delta/\varepsilon^2 + 50$.*

A tighter analysis (Lemma 16) shows that when $\delta \leq 1/16$ and $\varepsilon \leq 10^{-4}$, then $\texttt{skewed4Pack}$ has AAR $(76/45)(1 + 6\varepsilon) < T_\infty$, which improves upon the best-known bound on APoG for the general case.

In Sect. 4, we show that the lower bound of $4/3$ on APoG can be extended to skewed items.

Theorem 2 *Let m and k be positive integers and $\varepsilon \in (0, 1)$. Let I be a set of $4mk$ rectangular items, where $2mk$ items have width $(1 + \varepsilon)/2$ and height $(1 - \varepsilon)/2k$, and $2mk$ items have height $(1 + \varepsilon)/2$ and width $(1 - \varepsilon)/2k$. Let $\mathrm{opt}(I)$ be the number of bins in the optimal packing of I and $\mathrm{opt}_g(I)$ be the number of bins in the optimal guillotinable packing of I. Then*

$$\frac{\mathrm{opt}_g(I)}{\mathrm{opt}(I)} \geq \frac{4}{3}(1 - \varepsilon).$$

This holds true even if items in I are allowed to be rotated by $90°$.

Hence, our bounds on APoG are almost tight for skewed items. Our result indicates that to improve the bounds for APoG in the general case, we should focus on δ-large items.

Our other main result is something like an APTAS for 2BP for skewed items. Formally, in Sect. 5, we describe a polynomial-time algorithm for 2BP, called $\texttt{skewedCPack}$ (abbreviates *skewed compartmental packing*), that is parametrized by a constant $\varepsilon \in (0, 1/2]$. We show that for some constant $\delta \in (0, \varepsilon)$, $\texttt{skewedCPack}$ has an AAR of $1 + 14\varepsilon$ when all items in the input are δ-skewed rectangles.

Theorem 3 *Let $\varepsilon \in (0, 1/2]$. Let $f(x) := \varepsilon x/(104(1+1/(\varepsilon x))^{2/x-2})$. (Note that $f(x)$ increases with x, and $f(x) \leq x\ \forall x \leq 1/2$.) For any non-negative integer j, let $f^{(j)}(x)$ be x if $j = 0$ and $f(f^{(j-1)}(x))$ otherwise. Let $\eta := f^{\lceil 2/\varepsilon \rceil}(\varepsilon)$ and $\gamma := 1 + 1/(\varepsilon\eta)$.*

Let I be a set of η-skewed rectangular items. Then the number of bins used by $\texttt{skewedCPack}_\varepsilon(I)$ *is less than*

$$(1 + 14\varepsilon)\,\mathrm{opt}(I) + \frac{1}{20}\gamma^{2/\eta-3} + 17.$$

The running time of $\texttt{skewedCPack}_\varepsilon(I)$ *is* $O_\varepsilon(|I|^{2+\gamma^{\gamma^{2/\eta+1}}})$.

The best-known AAR for 2BP is $1+\ln(1.5)+\varepsilon$. Our result indicates that to improve upon algorithms for 2BP, one should focus on δ-large items.

In Appendices A and B, we show that our results also hold when items can be rotated by 90°.

2 Preliminaries

Let $[n] := \{1, 2, \ldots, n\}$, for $n \in \mathbb{N}$. For a rectangle i, its area $a(i) := w(i)h(i)$. For a set I of rectangles, let $a(I) := \sum_{i \in I} a(i)$. An *axis-aligned packing* of an item i in a bin is specified by a pair $(x(i), y(i))$, where $x(i) \in [0, 1 - w(i)]$ and $y(i) \in [0, 1-h(i)]$, so that i is placed in the region $[x(i), x(i)+w(i)] \times [y(i), y(i)+h(i)]$. A packing of rectangles in a bin is called *non-overlapping* iff for any two distinct items i and j, the rectangles $(x(i), x(i) + w(i)) \times (y(i), y(i) + h(i))$ and $(x(j), x(j) + w(j)) \times (y(j), y(j) + h(j))$ are disjoint. Equivalently, items may only intersect at their boundaries.

For a set I of rectangular items, define $\mathrm{opt}(I)$ as the minimum number of bins needed to pack I, and define $\mathrm{opt}_g(I)$ as the minimum number of guillotine-separable bins needed to pack I.

When analyzing running time of algorithms, we assume that arithmetic operations take constant time. For any function f and parameter ε, define $O_\varepsilon(f(n))$ as the set of functions of n that are upper-bounded by $Cf(n)$ for all sufficiently-large n and some value C that only depends on ε.

2.1 Next-Fit Decreasing Height

The NFDH algorithm [15] is a simple algorithm for 2SP and 2BP. We use the following results on NFDH. See Appendix C for the description of NFDH and proofs of these results.

Lemma 4 *Let I be a set of items where each item i has $w(i) \le \delta_W$ and $h(i) \le \delta_H$. NFDH can pack I into a bin of width W and height H if $a(I) \le (W - \delta_W)(H - \delta_H)$.*

Lemma 5 *NFDH uses less than $(2a(I) + 1)/(1 - \delta)$ bins to pack I when $h(i) \le \delta$ for each item i and less than $2a(I)/(1 - \delta) + 3$ bins when $w(i) \le \delta$ for each item i.*

If we swap the coordinate axes in NFDH, we get the Next-Fit Decreasing Width (NFDW) algorithm. Analogs of the above results hold for NFDW.

2.2 Slicing Items

We consider variants of 2BP where some items can be *sliced*. Formally, slicing a rectangular item i using a horizontal cut is the operation of replacing i by two items i_1 and i_2 such that $w(i) = w(i_1) = w(i_2)$ and $h(i) = h(i_1) + h(i_2)$. Slicing using vertical cut is defined analogously. Allowing some items to be sliced may reduce the number of bins required to pack them. See Fig. 2 for an example.

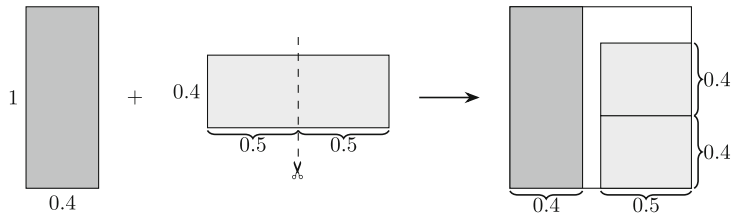


Fig. 2 Packing two items into a bin, where one item is sliced using a vertical cut. If slicing were forbidden, two bins would be required

Variants of 2SP where items can be sliced using vertical cuts find applications in resource allocation problems [39–41]. Many packing algorithms [8, 14, 24] solve the sliceable version of the problem as a subroutine.

3 Guillotinable Packing of Skewed Rectangles

An item is called (δ_W, δ_H)-skewed iff its width is at most δ_W or its height is at most δ_H. In this section, we consider the problem of obtaining good upper and lower bounds on APoG for (δ_W, δ_H)-skewed items. We describe the `skewed4Pack` algorithm and prove Theorem 1.

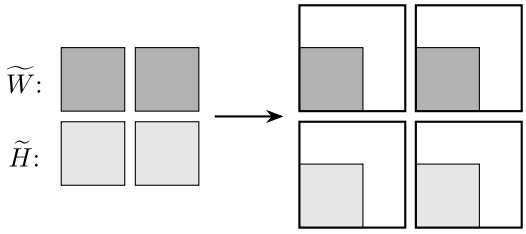
3.1 Packing With Slicing

Before describing `skewed4Pack`, let us first look at a closely-related variant of this problem, called the *sliceable 2D bin packing problem*, denoted as S2BP. In this problem, we are given two sets of rectangular items, $\widetilde{W}$ and $\widetilde{H}$, where items in $\widetilde{W}$ have width more than $1/2$, and items in $\widetilde{H}$ have height more than $1/2$. $\widetilde{W}$ is called the set of wide items and $\widetilde{H}$ is called the set of tall items. We are allowed to slice items in $\widetilde{W}$ using horizontal cuts and slice items in $\widetilde{H}$ using vertical cuts, and our task is to pack $\widetilde{W} \cup \widetilde{H}$ into the minimum number of bins without rotating the items. See Fig. 3 for an example that illustrates the difference between 2BP and S2BP.

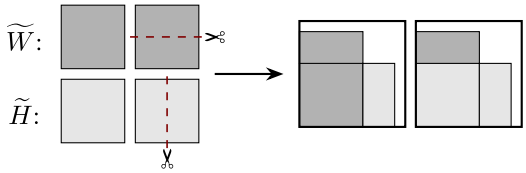
The S2BP problem can be viewed as a special case of the (δ_W, δ_H)-skewed 2BP problem when δ_W and δ_H are infinitesimally small. (This perspective is not used in this paper, so we omit a proof of equivalence.)

We first describe a simple $4/3$-asymptotic-approximation algorithm for S2BP, called `greedyPack`, that outputs a 2-stage packing. (Note that the asymptotic approximation ratio is with respect to the optimal solution to S2BP, i.e., items can be sliced in the optimal solution.) Later, we show how to use `greedyPack` to design `skewed4Pack`.

We assume that the bin is a square of side length 1. Since we can slice items, we allow items in $\widetilde{W}$ to have height more than 1 and items in $\widetilde{H}$ to have width more than 1.



(a) Packing items into 4 bins without slicing.



(b) Packing items into 2 bins by horizontally slicing an item in $\widetilde{W}$ and vertically slicing an item in $\widetilde{H}$.

Fig. 3 Example illustrating 2BP vs. S2BP. There are 2 wide items ($\widetilde{W}$) and 2 tall items ($\widetilde{H}$). The items are squares of side length 0.6 and the bins are squares of side length 1

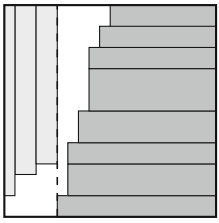
For $X \subseteq \widetilde{W}$, $Y \subseteq \widetilde{H}$, define

$$\text{hsum}(X) := \sum_{i \in X} h(i) \qquad \text{wmax}(X) := \begin{cases} \max_{i \in X} w(i) & \text{if } X \neq \emptyset \\ 0 & \text{if } X = \emptyset \end{cases}$$

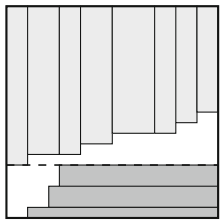
$$\text{wsum}(Y) := \sum_{i \in Y} w(i) \qquad \text{hmax}(Y) := \begin{cases} \max_{i \in Y} h(i) & \text{if } Y \neq \emptyset \\ 0 & \text{if } Y = \emptyset \end{cases}.$$

In the algorithm `greedyPack`$(\widetilde{W}, \widetilde{H})$, we first sort items $\widetilde{W}$ in decreasing order of width and sort items $\widetilde{H}$ in decreasing order of height. Suppose $\text{hsum}(\widetilde{W}) \geq \text{wsum}(\widetilde{H})$. Let X be the largest prefix of $\widetilde{W}$ of total height at most 1, i.e., if $\text{hsum}(\widetilde{W}) > 1$, then X is a prefix of $\widetilde{W}$ such that $\text{hsum}(X) = 1$ (slice items if needed), and $X = \widetilde{W}$ otherwise. Pack X into a bin such that the items touch the right edge of the bin. Then we pack the largest possible prefix of $\widetilde{H}$ into the empty rectangular region of width $1 - \text{wmax}(X)$ in the left side of the bin. We call this a type-1 bin. See Fig. 4a for an example. If $\text{hsum}(\widetilde{W}) < \text{wsum}(\widetilde{H})$, we proceed analogously in a coordinate-swapped way, i.e., we first pack tall items in the bin and then pack wide items in the remaining space. Call this bin a type-2 bin. See Fig. 4b for an example. We pack the rest of the items into bins in the same way. See Algorithm 1 for a more precise description of `greedyPack`.

Claim 6 `greedyPack`$(\widetilde{W}, \widetilde{H})$ *outputs a 2-stage packing of* $\widetilde{W} \cup \widetilde{H}$ *in* $O(m + |\widetilde{W}| \log |\widetilde{W}| + |\widetilde{H}| \log |\widetilde{H}|)$ *time, where* m *is the number of bins used. Furthermore, it slices items in* $\widetilde{W}$ *by making at most* $m - 1$ *horizontal cuts and slices items in* $\widetilde{H}$ *by making at most* $m - 1$ *vertical cuts.*



(a) A type-1 bin produced by `greedyPack`. Wide items are packed on the right. Tall items are packed on the left.



(b) A type-2 bin produced by `greedyPack`. Tall items are packed above. Wide items are packed below.

Fig. 4 Examples of type-1 and type-2 bins produced by `greedyPack`

Algorithm 1 `greedyPack`$(\widetilde{W}, \widetilde{H})$: Packs items $\widetilde{W} \cup \widetilde{H}$ into bins. The items $\widetilde{W}$ have width more than $1/2$ and can be sliced using horizontal cuts. The items $\widetilde{H}$ have height more than $1/2$ and can be sliced using vertical cuts.

Sort the items in $\widetilde{W}$ in decreasing order of width.
Sort the items in $\widetilde{H}$ in decreasing order of height.
while $\widetilde{W} \neq \emptyset$ or $\widetilde{H} \neq \emptyset$ **do**
  Create an empty bin.
  if $\text{hsum}(\widetilde{W}) \geq \text{wsum}(\widetilde{H})$ **then**
    Let X be the largest prefix of $\widetilde{W}$ of height at most 1.
    // *Hence,* $\text{hsum}(X) = 1$ *if* $\text{hsum}(\widetilde{W}) > 1$*; else* $X = \widetilde{W}$.
    Pack X in a region of width $\text{wmax}(X)$ on the right side of the bin.
    Remove X from $\widetilde{W}$.
    Let Y be the largest prefix of $\widetilde{H}$ of width at most $1 - \text{wmax}(X)$.
    Pack Y in a region of width $1 - \text{wmax}(X)$ on the left side of the bin.
    Remove Y from $\widetilde{H}$.
    Label the bin as a type-1 bin.
  else
    Proceed analogous to the previous case, i.e., X is a prefix of $\widetilde{H}$ of width at most 1 and Y is a prefix of $\widetilde{W}$ of height at most $1 - \text{hmax}(X)$.
    Label the bin as a type-2 bin.
  end if
end while

Since items in $\widetilde{W}$ have width more than $1/2$, no two items can be placed side-by-side. Hence, $\lceil \text{hsum}(\widetilde{W}) \rceil = \text{opt}(\widetilde{W}) \leq \text{opt}(\widetilde{W} \cup \widetilde{H})$, where $\text{opt}(X)$ is the minimum number of bins needed to pack X when we can slice wide items in X using horizontal cuts and slice tall items in X using vertical cuts. Similarly, $\lceil \text{wsum}(\widetilde{H}) \rceil \leq \text{opt}(\widetilde{W} \cup \widetilde{H})$. So, if all bins have the same type, `greedyPack` uses $\max(\lceil \text{hsum}(\widetilde{W}) \rceil, \lceil \text{wsum}(\widetilde{H}) \rceil) = \text{opt}(\widetilde{W} \cup \widetilde{H})$ bins. We now focus on the case where some bins have type 1 and some have type 2.

Definition 1 In a type-1 bin, let X and Y be the wide and tall items, respectively. The bin is called *full* iff $\text{hsum}(X) = 1$ and $\text{wsum}(Y) = 1 - \text{wmax}(X)$. Define fullness for type-2 bins analogously.

We first show that the total area of items packed in a full bin is large, and then show that if some bins have type 1 and some bins have type 2, then there are at most 2 non-full bins. This helps us upper-bound the number of bins used by `greedyPack`$(\widetilde{W}, \widetilde{H})$ in terms of $a(\widetilde{W} \cup \widetilde{H})$.

Lemma 7 *Let there be m_1 type-1 full bins. Let J_1 be the items in them. Then $m_1 \leq 4a(J_1)/3 + 1/3$.*

Proof In the jth full bin of type 1, let X_j be the items from $\widetilde{W}$ and Y_j be the items from $\widetilde{H}$. Let $\ell_j := \text{wmax}(X_j)$ if $j \leq m_1$ and $\ell_{m_1+1} := 1/2$. Since all items have their larger dimension more than $1/2$, $\ell_j \geq 1/2$ and $\text{hmax}(Y_j) > 1/2$, for any $j \in [m_1]$.

$a(X_j) \geq \ell_{j+1}$, since X_j has height 1 and width at least ℓ_{j+1}. $a(Y_j) \geq (1 - \ell_j)/2$, since Y_j has width $1 - \ell_j$ and height more than $1/2$. So, $a(J_1) = \sum_{j=1}^{m_1}(a(X_j) + a(Y_j)) \geq \sum_{j=1}^{m_1}(\ell_{j+1}+(1-\ell_j)/2) \geq \sum_{j=1}^{m_1}\big((\ell_{j+1}/2) + (1/4) + (1/2) - (\ell_j/2)\big) = (3m_1/4) + (1/4) - (\ell_1/2) \geq (3m_1 - 1)/4$. In the above inequalities, we used $\ell_{j+1} \geq 1/2$ and $\ell_1 \leq 1$.

Therefore, $m_1 \leq 4a(J_1)/3 + 1/3$. □

An analog of Lemma 7 can be proven for type-2 bins. Lemma 7 implies that very few full bins can have items of total area significantly less than 3/4.

Let m be the number of bins used by `greedyPack`$(\widetilde{W}, \widetilde{H})$. After j bins have been packed, let A_j be the height of the remaining items in $\widetilde{W}$ and B_j be the width of the remaining items in $\widetilde{H}$. Let t_j be the type of the jth bin (1 for type-1 bin, 2 for type-2 bin). So $t_j = 1 \iff A_{j-1} \geq B_{j-1}$.

We first show that $|A_{j-1} - B_{j-1}| \leq 1 \implies |A_j - B_j| \leq 1$, i.e., once $|\,\text{hsum}(\widetilde{W}) - \text{wsum}(\widetilde{H})|$ becomes at most 1 during `greedyPack`, it continues to stay at most 1. Next, we show that $t_j \neq t_{j+1} \implies |A_{j-1} - B_{j-1}| \leq 1$, i.e., if all bins do not have the same type, then $|\,\text{hsum}(\widetilde{W}) - \text{wsum}(\widetilde{H})|$ eventually becomes at most 1 during `greedyPack`. In the first non-full bin, we use up all the wide items or all the tall items. We now show that the remaining items have total height or total width at most 1, so we have at most 2 non-full bins.

In the jth bin, let a_j be the height of items from $\widetilde{W}$ and b_j be the width of items from $\widetilde{H}$. Hence, for all $j \in [m]$, $A_{j-1} = A_j + a_j$ and $B_{j-1} = B_j + b_j$.

Lemma 8 $|A_{j-1} - B_{j-1}| \leq 1 \implies |A_j - B_j| \leq 1$.

Proof W.l.o.g., assume $A_{j-1} \geq B_{j-1}$. So, $t_j = 1$. Suppose $a_j < b_j$. Then $a_j < 1$, so we used up $\widetilde{W}$ in the jth bin. Therefore, $A_j = 0 \implies A_{j-1} = a_j < b_j \leq b_j + B_j = B_{j-1}$, which is a contradiction. Hence, $a_j \geq b_j$. As $0 \leq (A_{j-1} - B_{j-1}), (a_j - b_j) \leq 1$, we get $A_j - B_j = (A_{j-1} - B_{j-1}) - (a_j - b_j) \in [-1, 1]$. □

Lemma 9 $t_j \neq t_{j+1} \implies |A_{j-1} - B_{j-1}| \leq 1$.

Proof W.l.o.g., assume $t_j = 1$ and $t_{j+1} = 2$. Then $A_{j-1} \geq B_{j-1}$ and $A_j < B_j \implies B_{j-1} \leq A_{j-1} < B_{j-1} + a_j - b_j \implies A_{j-1} - B_{j-1} \in [0, 1)$. □

Lemma 10 *If all bins do not have the same type, then there can be at most 2 non-full bins.*

Proof Let there be p full bins. Assume w.l.o.g. that in the $(p+1)$th bin, we used up all items from $\widetilde{W}$ but not $\widetilde{H}$. Hence, $A_{p+1} = 0$ and $\forall i \geq p+2, t_i = 2$. Since all bins do not have the same type, $\exists k \leq p+1$ such that $t_k = 1$ and $t_{k+1} = 2$. By Lemmas 9 and 8, we get $|A_{p+1} - B_{p+1}| \leq 1$, implying $B_{p+1} \leq 1$. Hence, the $(p+2)$th bin will use up all tall items, implying at most 2 non-full bins. □

Theorem 11 *The number of bins m used by* `greedyPack`$(\widetilde{W}, \widetilde{H})$ *is at most* $\max\left(\lceil \text{hsum}(\widetilde{W})\rceil, \lceil \text{wsum}(\widetilde{H})\rceil, \frac{4}{3}a(\widetilde{W} \cup \widetilde{H}) + \frac{8}{3}\right)$.

Proof If all bins have the same type, then $m \leq \max(\lceil \text{hsum}(\widetilde{W})\rceil, \lceil \text{wsum}(\widetilde{H})\rceil)$.

Let there be m_1 (resp. m_2) full bins of type 1 (resp. type 2) and let J_1 (resp. J_2) be the items inside those bins. Then by Lemma 7, we get $m_1 \leq 4a(J_1)/3 + 1/3$ and $m_2 \leq 4a(J_2)/3 + 1/3$. Hence, $m_1 + m_2 \leq 4a(\widetilde{W} \cup \widetilde{H})/3 + 2/3$. If all bins do not have the same type, then by Lemma 10, there can be at most 2 non-full bins, so `greedyPack`$(\widetilde{W}, \widetilde{H})$ uses at most $4a(\widetilde{W} \cup \widetilde{H})/3 + 8/3$ bins. □

Corollary 12 `greedyPack`$(\widetilde{W}, \widetilde{H}) \leq (4/3)\,\text{opt}(\widetilde{W} \cup \widetilde{H}) + (8/3)$. *(Hence,* `greedyPack` *is 4/3-asymptotic-approximate for S2BP).*

Proof Follows from Theorem 11, $\lceil \text{hsum}(\widetilde{W})\rceil = \text{opt}(\widetilde{W}) \leq \text{opt}(\widetilde{W} \cup \widetilde{H})$, $\lceil \text{wsum}(\widetilde{H})\rceil = \text{opt}(\widetilde{H}) \leq \text{opt}(\widetilde{W} \cup \widetilde{H})$, and $a(\widetilde{W} \cup \widetilde{H}) \leq \text{opt}(\widetilde{W} \cup \widetilde{H})$. □

3.2 The `skewed4Pack` Algorithm

We now return to the 2BP problem. `skewed4Pack` is an algorithm for 2BP that takes as input a set I of rectangular items and a parameter $\varepsilon \in (0, 1/2]$ where $\varepsilon^{-1} \in \mathbb{Z}$. It outputs a 4-stage bin packing of I. `skewed4Pack` has the following outline:

1. **Item Classification and Rounding**: Use linear grouping [3, 24] to round up the width or height of each item in I. This gives us a new instance $\widehat{I}$.
2. **Creating Shelves**: Pack $\widehat{I}$ into at most $2(1/\varepsilon^2 + 1)$ shelves, after possibly *slicing* some items. A shelf is a rectangular region with width or height more than $1/2$ and is fully packed, i.e., the total area of items in a shelf equals the area of the shelf. If we treat each shelf as an item, we get a new instance $\widetilde{I}$.
3. **Packing Shelves into Bins**: Compute a packing of $\widetilde{I}$ into bins, after possibly slicing some items, using `greedyPack`.
4. **Packing Items into Shelves**: Pack most of the items of I into the shelves in the bins. We prove that the remaining items have very small area, so they can be packed separately using NFDH.

3.2.1 Item Classification and Rounding

Define $W := \{i \in I : h(i) \leq \delta_H\}$ and $H := I - W$. Items in W are called *wide* and items in H are called *tall*. Let $W^{(L)} := \{i \in W : w(i) > \varepsilon\}$ and $W^{(S)} := W - W^{(L)}$. Similarly, let $H^{(L)} := \{i \in H : h(i) > \varepsilon\}$ and $H^{(S)} := H - H^{(L)}$.

We use *linear grouping* [3, 24] to round up the widths of items $W^{(L)}$ and the heights of items $H^{(L)}$ to get items $\widehat{W}^{(L)}$ and $\widehat{H}^{(L)}$, respectively. Since linear grouping is a standard technique (which we also use in Sect. 5.1), we describe it in Appendix D. Formally,

$\widehat{W}^{(L)} := \texttt{lingroupWide}(W^{(L)}, \varepsilon, \varepsilon)$ and $\widehat{H}^{(L)} := \texttt{lingroupTall}(H^{(L)}, \varepsilon, \varepsilon)$, where `lingroupWide` and `lingroupTall` are algorithms that we define in Appendix D. In Appendix D, we show that $\widehat{W}^{(L)}$ and $\widehat{H}^{(L)}$ satisfy the following properties:

- $|\widehat{W}^{(L)}| = |W|$, and each item in W is smaller than the corresponding item in $\widehat{W}^{(L)}$. Formally, there is a bijection $\pi : W \to \widehat{W}^{(L)}$ such that $\forall i \in W, w(i) \le w(\pi(i))$ and $h(i) = h(\pi(i))$.
- $|\widehat{H}^{(L)}| = |H|$, and each item in H is smaller than the corresponding item in $\widehat{H}^{(L)}$. Formally, there is a bijection $\pi : H \to \widehat{H}^{(L)}$ such that $\forall i \in H, h(i) \le h(\pi(i))$ and $w(i) = w(\pi(i))$.
- Items in $\widehat{W}^{(L)}$ have at most $1/\varepsilon^2$ distinct widths.
- Items in $\widehat{H}^{(L)}$ have at most $1/\varepsilon^2$ distinct heights.

Let $\widehat{W} := \widehat{W}^{(L)} \cup W^{(S)}$, $\widehat{H} := \widehat{H}^{(L)} \cup H^{(S)}$, and $\widehat{I} := \widehat{W} \cup \widehat{H}$. For $\widehat{X} \subseteq \widehat{I}$, let $\text{fopt}(\widehat{X})$ be the minimum number of bins needed to pack $\widehat{X}$ when items in $\widehat{X} \cap \widehat{W}^{(L)}$ can be sliced using horizontal cuts, items in $\widehat{X} \cap \widehat{H}^{(L)}$ can be sliced using vertical cuts, and items in $\widehat{X} \cap (W^{(S)} \cup H^{(S)})$ can be sliced both vertically and horizontally. Then the following lemma follows from Lemma 43 in Appendix D.

Lemma 13 $\text{fopt}(\widehat{I}) < (1+\varepsilon)\,\text{opt}(I) + 2$.

Linear grouping takes $O(n \log n + 1/\varepsilon^2)$ time, where $n := |I|$ (c.f. Lemma 41).

3.2.2 Creating Shelves

We use ideas from Kenyon and Rémila's 2SP algorithm [24] to pack $\widehat{I}$ into *shelves*. Roughly, we solve a linear program in $O(n) + O_\varepsilon(1)$ time to compute an optimal strip packing of $\widehat{W}$, where the packing is 3-stage. In this packing, we define *shelves* to be the rectangular regions given by the first stage of cuts, and we define *containers* to be the regions given by the second stage of cuts. From each shelf, we trim off space that doesn't belong to any container. We defer the details of shelf-creation to Sect. 3.2.5.

Let $\widetilde{W}$ be the shelves thus obtained. Analogously, we can pack items $\widehat{H}$ into shelves $\widetilde{H}$. Shelves in $\widetilde{W}$ are called *wide shelves* and shelves in $\widetilde{H}$ are called *tall shelves*. Let $\widetilde{I} := \widetilde{W} \cup \widetilde{H}$. We can interpret each shelf in $\widetilde{I}$ as a rectangular item. We allow slicing $\widetilde{W}$ and $\widetilde{H}$ using horizontal cuts and vertical cuts, respectively. In Sect. 3.2.5, we prove the following facts.

Lemma 14 *$\widetilde{I}$ has the following properties: (a) $|\widetilde{W}| \le 1 + 1/\varepsilon^2$ and $|\widetilde{H}| \le 1 + 1/\varepsilon^2$; (b) Each item in $\widetilde{W}$ has width more than $1/2$ and each item in $\widetilde{H}$ has height more than $1/2$; (c) $a(\widetilde{I}) = a(\widehat{I})$; (d) $\max(\lceil \text{hsum}(\widetilde{W}) \rceil, \lceil \text{wsum}(\widetilde{H}) \rceil) \le \text{fopt}(\widehat{I})$.*

3.2.3 Packing Shelves Into Bins

So far, we have packed $\widehat{I}$ into shelves $\widetilde{W}$ and $\widetilde{H}$. We now use $\texttt{greedyPack}(\widetilde{W}, \widetilde{H})$ to pack the shelves into bins. By Claim 6, we get a 2-stage packing of $\widetilde{W} \cup \widetilde{H}$ into m bins, where we make at most $m-1$ horizontal cuts in $\widetilde{W}$ and at most $m-1$ vertical cuts in $\widetilde{H}$. The horizontal cuts (resp. vertical cuts) increase the number of wide shelves

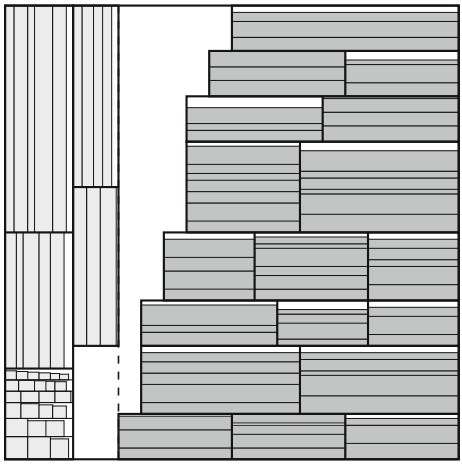


Fig. 5 A type-1 bin in the packing of $\widehat{I}$ computed by `skewed4Pack`. The packing contains 5 tall containers in 2 tall shelves and 18 wide containers in 8 wide shelves

(resp. tall shelves) from at most $1+1/\varepsilon^2$ to at most $m+1/\varepsilon^2$. By Theorem 11, Lemma 14(d) and Lemma 14(c), we get $m \le \max\left(\lceil \mathrm{hsum}(\widetilde{W})\rceil, \lceil \mathrm{wsum}(\widetilde{H})\rceil, \frac{4}{3}a(\widetilde{I}) + \frac{8}{3}\right) \le \frac{4}{3}\,\mathrm{fopt}(\widehat{I}) + \frac{8}{3}$. This step takes $O(m + (1/\varepsilon^2)\log(1/\varepsilon))$ time (c.f. Claim 6).

3.2.4 Packing Items Into Shelves

So far, we have a packing of shelves into m bins, where the shelves contain slices of items $\widehat{I}$. We now repack a large subset of the items $\widehat{I}$ into the shelves without slicing $\widehat{I}$. See Fig. 5 for an example output. We do this using a standard greedy algorithm. See Sect. 3.2.6 for details of the algorithm and proof of the following lemma.

Lemma 15 *Let P be a packing of $\widetilde{I}$ into m bins, where we made at most $m-1$ horizontal cuts in wide shelves and at most $m-1$ vertical cuts in tall shelves. Then we can (without slicing) pack a large subset of items $\widehat{I}$ into the shelves in P in $O(|\widehat{I}|\log|\widehat{I}| + m/\varepsilon + 1/\varepsilon^3)$ time such that the unpacked items (also called* discarded items*) from $\widehat{W}$ have total area less than $\varepsilon\,\mathrm{hsum}(\widetilde{W}) + \delta_H(1+\varepsilon)(m+1/\varepsilon^2)$, and the unpacked items from $\widehat{H}$ have total area less than $\varepsilon\,\mathrm{wsum}(\widetilde{H}) + \delta_W(1+\varepsilon)(m+1/\varepsilon^2)$.*

We pack wide discarded items into new bins using NFDH and pack tall discarded items into new bins using NFDW.

Finally, we prove the performance guarantee of $\texttt{skewed4Pack}_\varepsilon(I)$.

Lemma 16 *Let I be a set of (δ_W, δ_H)-skewed items. Then $\texttt{skewed4Pack}_\varepsilon(I)$ outputs a 4-stage packing of I in time $O(n(\log n + \max(\delta_H, \delta_W)/\varepsilon)) + O_\varepsilon(1)$, where $n = |I|$, and uses less than $\alpha\,\mathrm{opt}(I) + 2\beta$ bins, where $\Delta := \frac{1}{2}\left(\frac{\delta_H}{1-\delta_H} + \frac{\delta_W}{1-\delta_W}\right)$, $\alpha := (4/3)(1+4\Delta)(1+3\varepsilon)(1+\varepsilon)$, $\beta := (1/3)(6\Delta(1+\varepsilon)/\varepsilon^2 + 11 + 35\Delta + 12\varepsilon + 44\Delta\varepsilon)$.*

Proof The discarded items are packed using NFDH or NFDW, which output a 2-stage packing. Since `greedyPack` outputs a 2-stage packing of the shelves and the packing of items into the shelves is a 2-stage packing, the bin packing of non-discarded items is a 4-stage packing.

Suppose greedyPack uses at most m bins. Then by Theorem 11, $m \leq 4\,\mathrm{fopt}(\widehat{I})/3+8/3$. Let W^d and H^d be the items discarded from W and H, respectively. By Lemma 15 and Lemma 14(d), $a(W^d) < \varepsilon\,\mathrm{fopt}(\widehat{I}) + \delta_H(1+\varepsilon)(m+1/\varepsilon^2)$ and $a(H^d) < \varepsilon\,\mathrm{fopt}(\widehat{I}) + \delta_W(1+\varepsilon)(m+1/\varepsilon^2)$.

By Lemmas 5 and 13, the number of bins used by $\texttt{skewed4Pack}_\varepsilon(I)$ is less than

$$\begin{aligned}
&m + \frac{2a(W^d)+1}{1-\delta_H} + \frac{2a(H^d)+1}{1-\delta_W} \\
&\leq (1+4\Delta(1+\varepsilon))m + 4\varepsilon(1+\Delta)\,\mathrm{fopt}(\widehat{I}) + 2(1+\Delta) + 4\Delta(1+\varepsilon)/\varepsilon^2 \\
&\leq \frac{4}{3}(1+4\Delta+3\varepsilon+7\Delta\varepsilon)\,\mathrm{fopt}(\widehat{I}) + \frac{2}{3}\left(\frac{6\Delta(1+\varepsilon)}{\varepsilon^2} + 7 + 19\Delta + 16\Delta\varepsilon\right) \\
&\leq \alpha\,\mathrm{opt}(I) + 2\beta.
\end{aligned}$$

The last inequality follows from Lemma 13.

skewed4Pack runs in $O(n(\log n + \max(\delta_H, \delta_W)/\varepsilon)) + O_\varepsilon(1)$ time, since

- linear grouping takes $O(n \log n + 1/\varepsilon^2)$ time.
- shelf creation takes $O(n) + O_\varepsilon(1)$ time.
- packing shelves into bins using greedyPack takes $O(m+(1/\varepsilon^2)\log(1/\varepsilon))$ time, where m is the number of bins used.
- packing $\widehat{I}$ into shelves takes $O(n \log n + m/\varepsilon + 1/\varepsilon^3)$ time.
- packing discarded items using NFDH and NFDW takes $O(n \log n)$ time.

Furthermore,

$$\begin{aligned}
m &\leq \max(\lceil \mathrm{hsum}(\widetilde{W})\rceil, \lceil \mathrm{wsum}(\widetilde{H})\rceil, (4/3)a(\widetilde{I}) + (8/3)) \\
&\leq \max(2a(\widetilde{W}) + 1, 2a(\widetilde{H}) + 1, (4/3)a(\widetilde{I}) + (8/3)) \\
&\quad \text{(since wide shelves have width } > 1/2) \\
&\leq 2a(\widetilde{I}) + 8/3 \leq 2n\max(\delta_H, \delta_W) + 8/3 \\
&\in O(n\max(\delta_H, \delta_W) + 1).
\end{aligned}$$

□

Now we conclude with the proof of Theorem 1.

Theorem 1 *Let I be a set of δ-skewed items, where $\delta \in (0, 1/2]$. Then* $\texttt{skewed4Pack}_\varepsilon(I)$ *outputs a 4-stage packing of I in time $O(n(\log n + \delta/\varepsilon)) + O_\varepsilon(1)$, where $n := |I|$, and the number of bins used is less than $(4/3)(1+8\delta)(1+6\varepsilon)\,\mathrm{opt}(I) + 12\delta/\varepsilon^2 + 50$.*

Proof This is a simple corollary of Lemma 16, where $\delta \leq 1/2$ and $\varepsilon \leq 1/2$ give us $\Delta \leq 2\delta$, $\alpha \leq (4/3)(1+8\delta)(1+6\varepsilon)$, and $\beta < 6\delta/\varepsilon^2 + 25$. □

3.2.5 Details on Creating Shelves

Here we describe how to obtain shelves $\widetilde{W}$ and $\widetilde{H}$ from items $\widehat{W}$ and $\widehat{H}$, respectively.

Since we allow horizontally slicing items in $\widehat{W}$, a packing of $\widehat{W}$ into m bins gives us a packing of $\widehat{W}$ into a strip of height m, and a packing of $\widehat{W}$ into a strip of height h' gives us a packing of $\widehat{W}$ into $\lceil h' \rceil$ bins. Hence, define $\mathrm{fopt}_{\mathrm{SP}}(\widehat{W})$ as the height of the optimal strip packing of $\widehat{W}$ where we are allowed to slice items in $\widehat{W}$ using horizontal cuts. Then $\mathrm{fopt}(\widehat{W}) = \lceil \mathrm{fopt}_{\mathrm{SP}}(\widehat{W}) \rceil$. We now try to compute a near-optimal strip packing of $\widehat{W}$.

Recall the definition of $W^{(S)}$ and $\widehat{W}^{(L)}$ from Sect. 3.2. Let $\widehat{W}_j^{(L)}$ be the jth linear group of $\widehat{W}^{(L)}$, where $j \leq 1/\varepsilon^2$ (cf. Appendix D). Let w_j be the width of items in $\widehat{W}_j^{(L)}$.

Define a horizontal configuration S as a tuple $(S_0, S_1, S_2, \ldots)$ of $1/\varepsilon^2 + 1$ non-negative integers, where $S_0 \in \{0, 1\}$ and $\sum_{j=1}^{1/\varepsilon^2} S_j w_j \leq 1$. For any horizontal line at height y in a strip packing of $\widehat{W}$, the multiset of items intersecting the line corresponds to a configuration. S_0 indicates whether the line intersects items from $W^{(S)}$, and S_j is the number of items from $\widehat{W}_j^{(L)}$ that the line intersects. Let $\mathcal{S}$ be the set of all horizontal configurations. Let $N := |\mathcal{S}|$. Then $N \leq 2(1/\varepsilon)^{1/\varepsilon^2}$.

To obtain an optimal packing, we need to determine the height of each configuration. This can be done with the following linear program.

$$
\begin{aligned}
&\min_{x \in \mathbb{R}^N} \sum_{S \in \mathcal{S}} x_S \\
&\text{where} \quad \sum_{S \in \mathcal{S}} S_j x_S = h(\widehat{W}_j^{(L)}) && \forall j \in [1/\varepsilon^2] \\
&\text{and} \quad \sum_{S : S_0 = 1} \left(1 - \sum_{j=1}^{1/\varepsilon^2} S_j w_j \right) x_S = a(W^{(S)}) \\
&\text{and} \quad x_S \geq 0 && \forall S \in \mathcal{S}
\end{aligned}
$$

Here $h(\widehat{W}_j^{(L)}) := \sum_{i \in \widehat{W}_j^{(L)}} h(i)$. Let x^* be an optimal extreme-point solution to the above LP. This gives us a packing where the strip is divided into rectangular regions called *shelves* that are stacked on top of each other. Each shelf has a configuration S associated with it and has height $h(S) := x_S^*$. Each shelf can be divided into rectangular regions called *containers* that are stacked side-by-side. Each container has an integer j associated with it, called the *type* of the container, where $0 \leq j \leq 1/\varepsilon^2$. A shelf having configuration S contains S_j containers of type j. For $j \geq 1$, a type-j container has width w_j and only contains items from $\widehat{W}_j^{(L)}$. A type-0 container has width $w_0 := 1 - \sum_{j=1}^{1/\varepsilon^2} S_j w_j$ and only contains items from $W^{(S)}$. Each container is fully filled with items, i.e., the area of the container equals the sum of areas of items inside the container. Let $w(S)$ denote the width of shelf S. Then the sum of widths of all containers in S is $w(S)$. Hence, if $S_0 = 1$, then $w(S) = 1$; otherwise $w(S) = \sum_{j=1}^{1/\varepsilon^2} S_j w_j$.

Lemma 17 *x^* contains at most $1/\varepsilon^2 + 1$ positive entries.*

Proof sketch Follows by applying Rank Lemma[2] to the linear program. □

Lemma 18 $x^*_S > 0 \implies w(S) > 1/2$.

Proof Suppose $w(S) \leq 1/2$. Then we could have split S into two parts by making a horizontal cut in the middle and packed the parts side-by-side, reducing the height of the strip by $x^*_S/2$. But that would contradict the fact that x^* is optimal. □

Treat each shelf S as an item of width $w(S)$ and height $h(S)$. Allow each such item to be sliced using horizontal cuts. This gives us a new set $\widetilde{W}$ of items such that $\widehat{W}$ can be packed inside $\widetilde{W}$.

By applying an analogous approach to $\widehat{H}$, we get a new set $\widetilde{H}$ of items. Let $\widetilde{I} := \widetilde{W} \cup \widetilde{H}$. We call the shelves of $\widetilde{W}$ *wide shelves* and the shelves of $\widetilde{H}$ *tall shelves*. The containers in wide shelves are called *wide containers* and the containers in tall shelves are called *tall containers*.

Lemma 14 *$\widetilde{I}$ has the following properties: (a) $|\widetilde{W}| \leq 1 + 1/\varepsilon^2$ and $|\widetilde{H}| \leq 1 + 1/\varepsilon^2$; (b) Each item in $\widetilde{W}$ has width more than $1/2$ and each item in $\widetilde{H}$ has height more than $1/2$; (c) $a(\widetilde{I}) = a(\widehat{I})$; (d)* $\max(\lceil \text{hsum}(\widetilde{W})\rceil, \lceil \text{wsum}(\widetilde{H})\rceil) \leq \text{fopt}(\widehat{I})$.

Proof Lemma 17 implies (a) and Lemma 18 implies (b). $a(\widehat{I}) = a(\widetilde{I})$ as the shelves are tightly packed. Since x^* is an optimal solution to the linear program, $\lceil \text{hsum}(\widetilde{W})\rceil = \lceil \sum_{S\in\mathcal{S}} x^*_S\rceil = \lceil \text{fopt}_{\text{SP}}(\widehat{W})\rceil = \text{fopt}(\widehat{W}) \leq \text{fopt}(\widehat{I})$. Similarly, $\lceil \text{wsum}(\widetilde{H})\rceil = \text{fopt}(\widehat{H}) \leq \text{fopt}(\widehat{I})$. □

In $O(n + 1/\varepsilon^2)$ time, we can compute $h(\widehat{W}^{(L)}_j)$ for all j and $a(W^{(S)})$. In $O((1/\varepsilon)^{1/\varepsilon^2})$ time, we can compute $\mathcal{S}$. The LP has $1/\varepsilon^2 + 1$ non-trivial constraints and N variables. Hence, we can find the optimal extreme-point solution to it using the simplex algorithm [44], and the running time would be a constant (though it will be a very large constant that is a function of ε), provided an appropriate pivoting rule is used [45].

(Although there are polynomial-time algorithms for solving linear programs, they are not strongly polynomial-time. Since we want to get an extreme point solution and we do not want to worry about the bit-length of the input, we use the simplex algorithm).

3.2.6 Details on Packing Items Into Shelves

Lemma 15 *Let P be a packing of $\widetilde{I}$ into m bins, where we made at most $m-1$ horizontal cuts in wide shelves and at most $m-1$ vertical cuts in tall shelves. Then we can (without slicing) pack a large subset of items $\widehat{I}$ into the shelves in P in $O(|\widehat{I}|\log|\widehat{I}| + m/\varepsilon + 1/\varepsilon^3)$ time such that the unpacked items (also called* discarded items*) from $\widehat{W}$ have total area less than $\varepsilon\,\text{hsum}(\widetilde{W}) + \delta_H(1+\varepsilon)(m + 1/\varepsilon^2)$, and the unpacked items from $\widehat{H}$ have total area less than $\varepsilon\,\text{wsum}(\widetilde{H}) + \delta_W(1+\varepsilon)(m + 1/\varepsilon^2)$.*

[2] Rank Lemma: the number of non-zero variables in an extreme-point solution to a linear program is at most the number of non-trivial constraints [42, Lemma 2.1.4]..

Proof For each $j \in [1/\varepsilon^2]$, number the type-j wide containers arbitrarily, and number the items in $\widehat{W}_j^{(L)}$ arbitrarily. Now greedily assign items from $\widehat{W}_j^{(L)}$ to the first container C until the total height of the items exceeds $h(C)$. Then move to the next container and repeat. As per the constraints of the linear program, all items in $\widehat{W}_j^{(L)}$ will get assigned to some type-j wide container. Similarly, number the type-0 wide containers arbitrarily and number the items in $W^{(S)}$ arbitrarily. Greedily assign items from $W^{(S)}$ to the first container C until the total area of the items exceeds $a(C)$. Then move to the next container and repeat. As per the constraints of the linear program, all items in $W^{(S)}$ will get assigned to some type-0 wide container. Similarly, assign all items from $\widehat{H}$ to tall containers.

Let C be a type-j wide container and $\widehat{J}$ be the items assigned to it. If we discard the last item from $\widehat{J}$, then the items can be packed into C. The area of the discarded item is at most $w(C)\delta_H$. Let C be a type-0 wide container and $\widehat{J}$ be the items assigned to it. Arrange the items in $\widehat{J}$ in decreasing order of width and pack the largest prefix $\widehat{J}' \subseteq \widehat{J}$ into C using NFDW (Next-Fit Decreasing Width), which is an analog of NFDH with the coordinate axes swapped.

Discard the items $\widehat{J} - \widehat{J}'$. By Lemma 4, $a(\widehat{J} - \widehat{J}') < \varepsilon h(C) + \delta_H w(C) + \varepsilon\delta_H$. Therefore, for a wide shelf S, the total area of discarded items is less than $\varepsilon h(S) + \delta_H(1+\varepsilon)$.

After slicing the shelves in $\widetilde{I}$ to get P, we get at most $m + 1/\varepsilon^2$ wide shelves and at most $m + 1/\varepsilon^2$ tall shelves. Therefore, the total area of discarded items from W is less than

$$\varepsilon \operatorname{hsum}(\widetilde{W}) + \delta_H(1+\varepsilon)(m + 1/\varepsilon^2),$$

and the total area of discarded items from H is less than

$$\varepsilon \operatorname{wsum}(\widetilde{H}) + \delta_W(1+\varepsilon)(m + 1/\varepsilon^2).$$

In the above subroutine for packing $\widehat{I}$, for $n = |\widehat{I}|$, it takes $O(n \log n)$ time to sort $W^{(S)}$ by width, and $O(n + m_{\text{cont}})$ time to pack $\widehat{I}$ into containers, where m_{cont} is the number of containers in P. Since there are $2(m + 1/\varepsilon^2)$ shelves and each shelf has at most $1/\varepsilon$ containers, we get that $m_{\text{cont}} \le (2/\varepsilon)(m + 1/\varepsilon^2)$. □

4 Lower Bound on APoG

In this section, we prove a lower bound of roughly 4/3 on the APoG for skewed rectangles.

Lemma 19 *Let k be a positive integer and $\varepsilon \in (0, 1)$ be a real number. Let J be a set of items packed into a bin, where each item has the longer dimension equal to $(1+\varepsilon)/2$ and the shorter dimension equal to $(1-\varepsilon)/2k$. If the bin is guillotine-separable, then $a(J) \le 3/4 + \varepsilon/2 - \varepsilon^2/4$.*

Proof For an item packed in the bin, if the height is $(1-\varepsilon)/2k$, call it a wide item, and if the width is $(1-\varepsilon)/2k$, call it a tall item. Let W be the set of wide items in J.

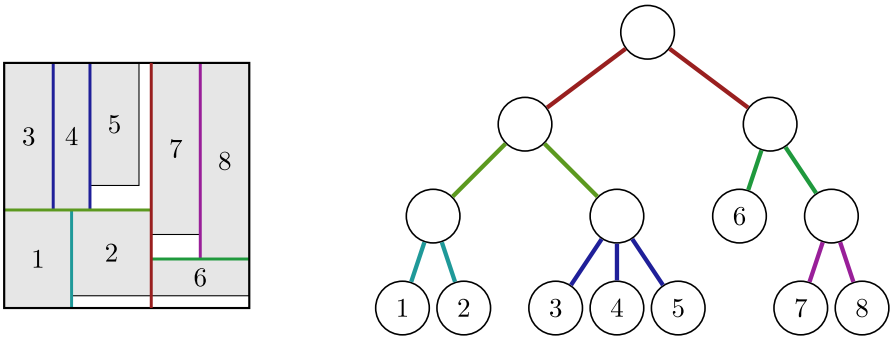


Fig. 6 A guillotinable packing of items into a bin and the corresponding guillotine tree

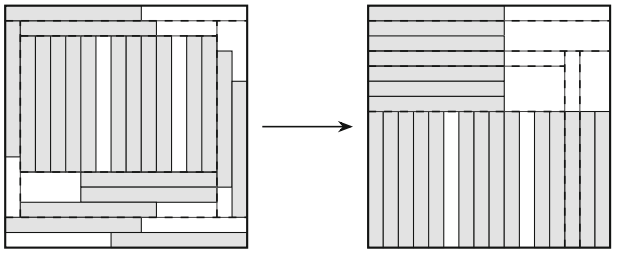


Fig. 7 Structuring a guillotine-separable packing

The packing of items in the bin can be represented as a tree, called the *guillotine tree* of the bin, where each node u represents a rectangular region of the bin (the root node represents the entire bin) and the child nodes $v_1, v_2, \ldots, v_p$ of node u represent the sub-regions obtained by parallel guillotine cuts. The ordering of the children has a significance here: if the guillotine cuts are vertical, children are ordered by increasing x-coordinate, and if the cuts are horizontal, children are ordered by increasing y-coordinate. See Fig. 6 for an example.

We now show how to rearrange the items in the bin so that the packing remains guillotine-separable but becomes more structured. We exploit this structure to show that the packing has a large unpacked area. See Fig. 7 for an example.

In the guillotine tree, suppose there is a node u that has children $v_1, v_2, \ldots, v_p$. W.l.o.g., assume that the children are obtained by making vertical cuts. At most one of these children can contain items from W. We can assume w.l.o.g. that the other children contain only one item, because otherwise we can separate them by vertical cuts. We can reorder the children (which is equivalent to repacking the guillotine partitions) so that the child containing items from W (if any) is the first child. Therefore, we can assume w.l.o.g. that at any level in the guillotine tree, only the first node has children.

Based on the argument above, we can see that the first node in each level touches the bottom-left corner of the bin. All the other nodes either contain a single wide item and touch the left edge of the bin but not the bottom edge, or they contain a single tall item and touch the bottom edge of the bin but not the left edge. In each node containing a wide item, shift the item leftwards, and in each node containing a tall item, shift the

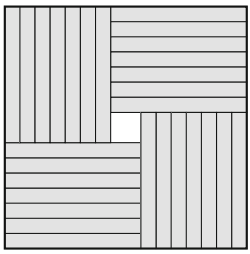


Fig. 8 Packing $4k$ items in one bin. Here $k = 7$

item downwards. Then each wide item touches the left edge of the bin and each tall item touches the bottom edge of the bin.

Therefore, the square region of side length $(1 - \varepsilon)/2$ at the top-right corner of the bin is empty. Hence, the area occupied in each bin is at most $3/4 + \varepsilon/2 - \varepsilon^2/4$. □

Theorem 2 *Let m and k be positive integers and $\varepsilon \in (0, 1)$. Let I be a set of $4mk$ rectangular items, where $2mk$ items have width $(1+\varepsilon)/2$ and height $(1-\varepsilon)/2k$, and $2mk$ items have height $(1+\varepsilon)/2$ and width $(1-\varepsilon)/2k$. Let $\mathrm{opt}(I)$ be the number of bins in the optimal packing of I and $\mathrm{opt}_g(I)$ be the number of bins in the optimal guillotinable packing of I. Then*

$$\frac{\mathrm{opt}_g(I)}{\mathrm{opt}(I)} \geq \frac{4}{3}(1-\varepsilon).$$

This holds true even if items in I are allowed to be rotated by 90°.

Proof For an item $i \in I$, if $h(i) = (1-\varepsilon)/2k$, call it a wide item, and if $w(i) = (1-\varepsilon)/2k$, call it a tall item. Let W be the set of wide items and H be the set of tall items. We show that $\mathrm{opt}_g(I)/\mathrm{opt}(I)$ is sufficiently large, which will give us a lower-bound on APoG.

Partition W into groups of k elements. In each group, stack items one-over-the-other. This gives us $2m$ containers of width $(1+\varepsilon)/2$ and height $(1-\varepsilon)/2$. Similarly, get $2m$ containers of height $(1+\varepsilon)/2$ and height $(1-\varepsilon)/2$ by stacking items from H side-by-side. We can pack 4 containers in one bin, so I can be packed into m bins. See Fig. 8 for an example. Therefore, $\mathrm{opt}(I) \leq m$.

We now show a lower-bound on $\mathrm{opt}_g(I)$. In any guillotine-separable packing of I, the area occupied by each bin is at most $3/4 + \varepsilon/2 - \varepsilon^2/4$ (by Lemma 19). Note that $a(I) = m(1-\varepsilon^2)$. Therefore,

$$\mathrm{opt}_g(I) \geq \frac{m(1-\varepsilon^2)}{3/4 + \varepsilon/2 - \varepsilon^2/4}$$
$$\implies \frac{\mathrm{opt}_g(I)}{\mathrm{opt}(I)} \geq \frac{4}{3} \times \frac{1-\varepsilon^2}{1 + 2\varepsilon/3 - \varepsilon^2/3} = \frac{4}{3} \times \frac{1-\varepsilon}{1-\varepsilon/3} \geq \frac{4}{3}(1-\varepsilon).$$

□

5 Almost-Optimal Bin Packing of Skewed Rectangles

In this section, we describe the algorithm `skewedCPack`. `skewedCPack` takes as input a set I of δ-skewed items and a parameter $\varepsilon \in (0, 1/2]$, where $\varepsilon^{-1} \in \mathbb{Z}$. We prove that `skewedCPack` has AAR $1 + 14\varepsilon$ when δ is sufficiently small. `skewedCPack` works roughly as follows:

1. Invoke the subroutine `round`(I) (cf. Sect. 5.1). `round`(I) returns a pair $(\widetilde{I}, I_{\text{med}})$. Here I_{med}, called the set of *medium items*, has low total area, so we can pack it in a small number of bins. $\widetilde{I}$, called the set of *rounded items*, is obtained by rounding up the width or height of each item in $I - I_{\text{med}}$, so that $\widetilde{I}$ has special properties that help us pack it easily.
2. Compute the optimal *fractional compartmental* bin packing of $\widetilde{I}$ (we will define *fractional* and *compartmental* later).
3. Use this packing of $\widetilde{I}$ to obtain a packing of I that uses slightly more number of bins.

To bound the AAR of `skewedCPack`, we prove a structural theorem (Sect. 5.2), which says that the optimal fractional compartmental packing of $\widetilde{I}$ uses close to $\text{opt}(I)$ bins.

We first describe `round` and the properties of its output in Sect. 5.1. We then define compartmental packing in Sect. 5.2. In Sect. 5.3, we give an overview of the `skewedCPack` algorithm. We provide the remaining details in Sects. 5.4, 5.5 and 5.6.

5.1 Classifying and Rounding Items

We now describe the algorithm `round`. It takes as input the set I of items and a parameter $\varepsilon \in (0, 1/2]$, and outputs the pair $(\widetilde{I}, I_{\text{med}})$ where $\widetilde{I}$ and I_{med} are sets of items that satisfy useful properties.

5.1.1 Removing Medium Items

First, we find a set $I_{\text{med}} \subseteq I$ (called the set of *medium* items) and positive constants ε_1 and ε_2 such that $a(I_{\text{med}}) \leq \varepsilon a(I)$ and $I - I_{\text{med}}$ is $(\varepsilon_2, \varepsilon_1]$-free, i.e., no item in $I - I_{\text{med}}$ has its width or height in the interval $(\varepsilon_2, \varepsilon_1]$. Then we can remove I_{med} from I and pack it separately into a small number of bins using NFDH. We will see that when $\varepsilon_2 \ll \varepsilon_1$, the $(\varepsilon_2, \varepsilon_1]$-freeness of $I - I_{\text{med}}$ will help us pack $I - I_{\text{med}}$ efficiently.

Additionally, we impose the conditions $\varepsilon_1 \leq \varepsilon$, $\varepsilon_1^{-1} \in \mathbb{Z}$, and $\varepsilon_2 = f(\varepsilon_1)$, where $f(x) := \varepsilon x / \left(104(1 + 1/(\varepsilon x))^{2/x-2}\right)$. We explain this choice of f in Sect. 5.3. Intuitively, such an f ensures that $\varepsilon_2 \ll \varepsilon_1$ (because $f(x) \ll x$ for small x) and $\varepsilon_2^{-1} \in \mathbb{Z}$. For `skewedCPack` to work, we require $\delta \leq \varepsilon_2$. Finding such an I_{med} and ε_1 is a standard technique [8, 9], which we describe now.

Let $T := \lceil 2/\varepsilon \rceil$. Let $\mu_0 = \varepsilon$. For $t \in [T]$, define $\mu_t := f(\mu_{t-1})$ and define

$$J_t := \{i \in I : w(i) \in (\mu_t, \mu_{t-1}] \text{ or } h(i) \in (\mu_t, \mu_{t-1}]\}.$$

Define $r := \operatorname{argmin}_{t=1}^{T} a(J_t)$, $I_{\text{med}} := J_r$ and $\varepsilon_1 := \mu_{r-1}$. Each item belongs to at most 2 sets J_t, so

$$a(I_{\text{med}}) = \min_{t=1}^{T} a(J_t) \le \frac{1}{T}\sum_{t=1}^{T} a(J_t) \le \frac{2}{\lceil 2/\varepsilon \rceil} a(I) \le \varepsilon a(I).$$

We can compute I_{med} and ε_1 in $O(|\widetilde{I}| \log(1/\varepsilon) + 1/\varepsilon)$ time, since we can compute all μ_t in $O(1/\varepsilon)$ time, and for each item i, we can find all t such that $i \in J_t$ in $O(\log(1/\varepsilon))$ time by binary-searching for $w(i)$ and $h(i)$ in the list $[\mu_1, \mu_2, \ldots]$.

5.1.2 Classifying Items

Next, we classify the items in $I - I_{\text{med}}$ into three disjoint classes:

- Wide items: $W := \{i \in I : w(i) > \varepsilon_1 \text{ and } h(i) \le \varepsilon_2\}$.
- Tall items: $H := \{i \in I : w(i) \le \varepsilon_2 \text{ and } h(i) > \varepsilon_1\}$.
- Small items: $S := \{i \in I : w(i) \le \varepsilon_2 \text{ and } h(i) \le \varepsilon_2\}$.

Note that there is no item $i \in I$ for which both $w(i)$ and $h(i)$ are more than ε_1, since I is δ-skewed, so $\min(w(i), h(i)) \le \delta \le \varepsilon_2$.

5.1.3 Linear Grouping

We now use *linear grouping* [3, 24] to round up the widths of items W and the heights of items H to get items $\widetilde{W}$ and $\widetilde{H}$, respectively. Since linear grouping is a standard technique (which we also use in Sect. 3.2), we describe it in Appendix D. Formally, let $\widetilde{W} := \texttt{lingroupWide}(W, \varepsilon, \varepsilon_1)$ and $\widetilde{H} := \texttt{lingroupTall}(H, \varepsilon, \varepsilon_1)$, where `lingroupWide` and `lingroupTall` are algorithms that we define in Appendix D. In Appendix D, we show that $\widetilde{W}$ and $\widetilde{H}$ satisfy the following properties:

- $|\widetilde{W}| = |W|$, and each item in W is smaller than the corresponding item in $\widetilde{W}$. Formally, there is a bijection $\pi : W \to \widetilde{W}$ such that $\forall i \in W, w(i) \le w(\pi(i))$ and $h(i) = h(\pi(i))$.
- $|\widetilde{H}| = |H|$, and each item in H is smaller than the corresponding item in $\widetilde{H}$. Formally, there is a bijection $\pi : H \to \widetilde{H}$ such that $\forall i \in H, h(i) \le h(\pi(i))$ and $w(i) = w(\pi(i))$.
- Items in $\widetilde{W}$ have at most $1/(\varepsilon\varepsilon_1)$ distinct widths.
- Items in $\widetilde{H}$ have at most $1/(\varepsilon\varepsilon_1)$ distinct heights.

Let $\widetilde{I} := \widetilde{W} \cup \widetilde{H} \cup S$.

Definition 2 (Fractional packing) Suppose we are allowed to slice wide items in $\widetilde{I}$ using horizontal cuts, slice tall items in $\widetilde{I}$ using vertical cuts and slice small items in $\widetilde{I}$ using both horizontal and vertical cuts. For any $\widetilde{X} \subseteq \widetilde{I}$, a bin packing of the slices of $\widetilde{X}$ is called a *fractional packing* of $\widetilde{X}$. The optimal fractional packing of $\widetilde{X}$ is denoted by $\text{fopt}(\widetilde{X})$.

Lemma 20 (Proved as Lemma 43 in Appendix D) $\text{fopt}(\widetilde{I}) < (1+\varepsilon)\,\text{opt}(I) + 2$.

Linear grouping takes $O(n \log n + 1/\varepsilon\varepsilon_1)$ time, where $n := |I|$ (c.f. Lemma 41). Hence, $\texttt{round}(I)$ takes $O(n \log(n/\varepsilon) + 1/\varepsilon\varepsilon_1)$ time.

5.2 Structural Theorem

We now define compartmental packing and prove the structural theorem, which says that the number of bins in the optimal fractional compartmental packing of $\widetilde{I}$ is roughly equal to $\text{fopt}(\widetilde{I})$.

We first show how to *discretize* a packing, i.e., we show that given a fractional packing of items in a bin, we can remove a small fraction of tall and small items and shift the remaining items leftwards so that the x-coordinate of left and right edges of each wide item belong to a constant-sized set $\mathcal{T}$, where $|\mathcal{T}| \le (1+1/\varepsilon\varepsilon_1)^{2/\varepsilon_1-2}$. Next, we define *compartmental* packing and show how to convert a discretized packing to a compartmental packing.

5.2.1 Defining $\mathcal{T}$

For any sets $A, B \subseteq [0,1]$, define the *restricted Minkowski sum* $A \oplus B$ as $\{x+y : x \in A, y \in B, x+y \le 1\}$. For any $\alpha \in (0,1]$ such that $\alpha^{-1} \in \mathbb{Z}$, define $\text{unif}(\alpha)$ as $\{k\alpha : k \in \mathbb{Z}, 0 \le k < 1/\alpha\}$.

Let R be the set of distinct widths of items in $\widetilde{W}$. Given the way we rounded items, $|R| \le 1/\varepsilon\varepsilon_1$. For $j \ge 0$, define $R^{(j)}$ as the restricted Minkowski sum of j copies of $R \cup \{0\}$. Define μ and $\mathcal{T}$ as

$$\mu := \frac{\varepsilon\varepsilon_1}{(1+1/\varepsilon\varepsilon_1)^{2/\varepsilon_1-4}}, \qquad \mathcal{T} := \text{unif}(\mu) \oplus R^{(1/\varepsilon_1-1)}$$

Recall that $\varepsilon_1 \le \varepsilon \le 1/2$ and $\varepsilon_1^{-1}, \varepsilon^{-1} \in \mathbb{Z}$, so $\mu^{-1} \in \mathbb{Z}$.

Lemma 21 $4 \le 1/\varepsilon\varepsilon_1 \le |\mathcal{T}| \le (1+1/\varepsilon\varepsilon_1)^{2/\varepsilon_1-2}$.

Proof $|\mathcal{T}| \ge |\text{unif}(\mu)| = 1/\mu \ge 1/\varepsilon\varepsilon_1 \ge 4$. Let $\gamma := 1+1/\varepsilon\varepsilon_1$. Then

$$\begin{aligned}
|\mathcal{T}| &\le \frac{1}{\mu} + |R \cup \{0\}|^{1/\varepsilon_1-1} = \frac{\gamma^{2/\varepsilon_1-4}}{\varepsilon\varepsilon_1} + \gamma^{1/\varepsilon_1-1} \\
&\le \frac{\gamma^{2/\varepsilon_1-3}}{\varepsilon\varepsilon_1} + \gamma^{2/\varepsilon_1-3} \qquad (\varepsilon_1 \le 1/2 \implies 1/\varepsilon_1 - 1 \le 2/\varepsilon_1 - 3) \\
&= \gamma^{2/\varepsilon_1-2}.
\end{aligned}$$

□

5.2.2 Discretization

One of our main results regarding `skewedCPack` is the Discretization Theorem, which we prove in Sect. 5.4.

For any rectangle i packed in a bin, let $x_1(i)$ and $x_2(i)$ denote the x-coordinates of its left and right edges, respectively, and let $y_1(i)$ and $y_2(i)$ denote the y-coordinates of its bottom and top edges, respectively.

Theorem 22 (Discretization) *Given a fractional packing of items $\widetilde{J} \subseteq \widetilde{I}$ into a bin, we can remove tall and small items of total area less than ε and shift some of the remaining items to the left such that for every wide item i, we get $x_1(i), x_2(i) \in \mathcal{T}$.*

5.2.3 Compartmental Packing

Definition 3 (Compartmental packing) Consider a packing of some items into a bin. A *compartment* C is defined as a rectangular region in the bin satisfying the following properties:

- $x_1(C) \in \mathcal{T}, x_2(C) \in \mathcal{T} \cup \{1\} - \{0\}$.
- $y_1(C), y_2(C)$ are multiples of $\varepsilon_{\text{cont}} := \varepsilon\varepsilon_1/6|\mathcal{T}|$.
- Each item either lies inside C or outside C, i.e., no item crosses C's boundary.
- C does not contain both wide items and tall items.
- If C contains tall items, then $x_1(C)$ and $x_2(C)$ are consecutive values in $\mathcal{T}$.

If a compartment contains a wide item, it is called a *wide compartment*. Otherwise it is called a *tall compartment*. A packing of items into a bin is called *compartmental* iff there is a set of non-overlapping *compartments* in the bin such that all of the following hold:

- each wide or tall item lies completely inside some compartment,
- each tall compartment's top and bottom edges either touch a wide compartment or the edge of the bin.
- there are at most $n_W := 3(1/\varepsilon_1 - 1)|\mathcal{T}| + 1$ wide compartments in the bin.
- there are at most $n_H := (1/\varepsilon_1 - 1)|\mathcal{T}|$ tall compartments in the bin.

A packing of items into multiple bins is called compartmental iff each bin is compartmental.

Note that small items can be packed both inside and outside compartments.

A *fractional compartmental* packing is a fractional packing that is also compartmental, i.e., it is a compartmental packing of slices of items (we can slice wide items using horizontal cuts, tall items using vertical cuts, and small items both horizontally and vertically). Note that it's possible for different slices of the same item to be in different compartments.

The following lemma states that a discretized packing can be converted to a fractional compartmental packing.

Lemma 23 *If $x_1(i), x_2(i) \in \mathcal{T}$ for each wide item i in a bin, then by removing wide and small items of area less than ε, we can get a fractional compartmental packing of the remaining items into a bin.*

Lemma 23 can be proved using standard techniques (e.g., Section 3.2.3 in [43]). We provide a formal proof in Sect. 5.6.

Theorem 24 (Structural Theorem) *For a set $\widetilde{I}$ of δ-skewed rounded items, define* $\text{fcopt}(\widetilde{I})$ *as the number of bins in the optimal fractional compartmental packing of $\widetilde{I}$. Then* $\text{fcopt}(\widetilde{I}) < (1 + 4\varepsilon)\,\text{fopt}(\widetilde{I}) + 2$.

Proof Consider a fractional packing of $\widetilde{I}$ into $m := \text{fopt}(\widetilde{I})$ bins. From each bin, we can discard items of area less than 2ε and get a fractional compartmental packing of the remaining items by Theorem 22 and Lemma 23.

Let X be the set of wide and small discarded items and let Y be the set of tall discarded items. For each item $i \in X$, if $w(i) \leq 1/2$, slice it using a horizontal cut in the middle and place the pieces horizontally next to each other to get a new item of width $2w(i)$ and height $h(i)/2$. Repeat until $w(i) > 1/2$. Now pack the items in bins by stacking them one-over-the-other so that for each item $i \in X$, $x_1(i) = 0$. This will require less than $2a(X) + 1$ bins, and the packing will be compartmental, where each bin has a single wide compartment of width and height 1.

Similarly, we can get a fractional compartmental packing of Y into less than $2a(Y)+1$ bins, where each bin has $|\mathcal{T}|$ tall compartments of height 1 each. Since $a(X \cup Y) < 2\varepsilon m$, we require less than $4\varepsilon m + 2$ bins. Therefore, the total number of compartmental bins used to fractionally pack $\widetilde{I}$ is less than $(1 + 4\varepsilon)m + 2$. □

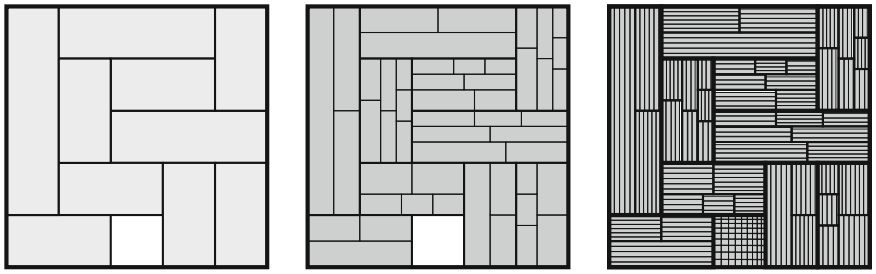
5.3 Overview of Packing Algorithm

We now give an overview of the `skewedCPack` algorithm for packing a set I of δ-skewed items. It consists of the following steps:

1. `round`(I): **classifies and rounds items** (see Sect. 5.1):
   `round`(I) returns a pair $(\widetilde{I}, I_{\text{med}})$, where I_{med}, called the set of *medium items*, has low total area, and $\widetilde{I}$, called the set of *rounded items*, is obtained by rounding up the width or height of each item in $I - I_{\text{med}}$.
2. `iterPackings`($\widetilde{I}$): **enumerates packing of compartments**:
   The subroutine `iterPackings`($\widetilde{I}$) computes all possible packings of empty compartments into at least $\lceil a(\widetilde{I}) \rceil$ bins and at most $|\widetilde{I}|$ bins. In Sect. 5.5.1, we describe this subroutine and show that it outputs at most $|\widetilde{I}|^{\nu}$ packings and runs in time $O(|\widetilde{I}|^{\nu+1})$, where $\nu := \gamma^{\gamma^{2/\varepsilon_1+1}}$ and $\gamma := 1 + 1/\varepsilon\varepsilon_1$.
3. **Fractionally pack items into compartments**:
   For each packing P of empty compartments output by `iterPackings`($\widetilde{I}$), fractionally pack $\widetilde{I}$ into P (if possible) using a linear program. Specifically, we compute an extreme point solution z^* to a constant-sized linear program FP($\widetilde{I}$, P). This gives us a fractional compartmental packing of $\widetilde{I}$. In Sect. 5.5.2, we define FP($\widetilde{I}$, P) and show that we can compute z^* in $O(m|\mathcal{T}|/\varepsilon_1) + O_\varepsilon(1)$ time, where m is the number of bins in P.
4. `greedyCPack`($\widetilde{I}$, P, z^*): **converts to non-fractional packing**:
   Discard a small set $D \subseteq \widetilde{I}$ of items and use z^* to compute a non-fractional packing of $\widetilde{I} - D$ into P. Formally, the subroutine `greedyCPack`($\widetilde{I}$, P, z^*) outputs a pair (Q, D), where Q is a packing of $\widetilde{I} - D$. We describe `greedyCPack` in Sect. 5.5.3.
5. Pack $I_{\text{med}} \cup D$ into bins using the NFDH algorithm.

See Fig. 9 for a visual overview of `skewedCPack`, and Algorithm 2 for a precise description of `skewedCPack`.

Definition 4 (Packing concatenation) Let P_1 be a packing of items I_1 into m_1 bins and P_2 be a packing of items I_2 into m_2 bins. Then define $P_1 + P_2$ as the packing of



(a) Guess the packing of empty compartments in each bin (Section 5.5.1).

(b) Fractionally pack wide and tall items into compartments. This partitions each compartment into *containers* (Section 5.5.2).

(c) Pack the items non-fractionally (Section 5.5.3).

Fig. 9 Major steps of `skewedCPack` after `rounding` I

items $I_1 \cup I_2$ into $m_1 + m_2$ bins, where the first m_1 bins pack I_1 according to P_1, and the next m_2 bins pack I_2 according to P_2.

Algorithm 2 `skewedCPack`$_\varepsilon(I)$: Packs a set I of δ-skewed rectangular items into bins without rotating the items. For a linear feasibility program X, `extPt`(X) returns an arbitrary extreme point solution to X if X is feasible, and returns `null` if X is infeasible.

1: $(\widetilde{I}, I_{\text{med}}) = \texttt{round}_\varepsilon(I)$.
2: Initialize Q_{best} to `null`.
3: **for** $P \in$ `iterPackings`$(\widetilde{I})$ **do** // `iterPackings` *is defined in Section 5.5.1.*
4: $z^* =$ `extPt`(FP$(\widetilde{I}, P)$). // FP *is defined in Section 5.5.2.*
5: **if** $z^* \neq$ `null` **then** // *if* $\widetilde{I}$ *can be packed into* P
6: $(Q, D) =$ `greedyCPack`$(\widetilde{I}, P, z^*)$.
7: // `greedyCPack` *is defined in Section 5.5.3.*
8: $Q_D = \text{NFDH}(D \cup I_{\text{med}})$.
9: **if** $Q + Q_D$ uses less bins than Q_{best} **then**
10: // $Q + Q_D$ *is defined in Definition 4.*
11: $Q_{\text{best}} = Q + Q_D$.
12: **end if**
13: **end if**
14: **end for**
15: **return** Q_{best}

In Sect. 5.5.3, we describe `greedyCPack` and prove the following result.

Lemma 25 *Let* $(Q, D) :=$ `greedyCPack`$(\widetilde{I}, P, z^*)$. *Let there be* m *bins in* P. *Then*

$$a(D) < 4\varepsilon_2 \left(\frac{13|\mathcal{T}|}{\varepsilon_1} m + \frac{|\mathcal{T}|(|\mathcal{T}|+1)}{2} + \frac{6|\mathcal{T}|}{\varepsilon\varepsilon_1} + \frac{2}{\varepsilon\varepsilon_1} \right),$$

and `greedyCPack`$(\widetilde{I}, P, z^*)$ *runs in time*

$$O\left(|\widetilde{I}|\log|\widetilde{I}| + \frac{|\mathcal{T}|}{\varepsilon_1^2}m + \frac{|\mathcal{T}|^2}{\varepsilon_1}\right).$$

Recall the function f from Sect. 5.1.1. Since $\varepsilon_2 := f(\varepsilon_1)$, we get

$$\varepsilon_2 = f(\varepsilon_1) = \frac{\varepsilon\varepsilon_1}{104(1+1/\varepsilon\varepsilon_1)^{2/\varepsilon_1-2}} \le \frac{\varepsilon\varepsilon_1}{104|\mathcal{T}|}. \tag{1}$$

The last inequality follows from the fact that $|\mathcal{T}| \le (1+1/\varepsilon\varepsilon_1)^{2/\varepsilon_1-2}$ (Lemma 21). Also, using $\varepsilon_1 \le \varepsilon \le 1/2$, we get $\delta \le \varepsilon_2 = f(\varepsilon_1) \le f(1/2) < 10^{-4}$.

Theorem 26 *If I is ε_2-skewed, the number of bins used by* `skewedCPack`$_\varepsilon(I)$ *is less than*

$$(1+14\varepsilon)\,\mathrm{opt}(I) + \frac{1}{20}\gamma^{2/\varepsilon_1-3} + 17,$$

where $\gamma := 1+1/\varepsilon\varepsilon_1$. The running time of `skewedCPack`$_\varepsilon(I)$ *is* $O_\varepsilon(|I|^{2+\gamma^{\gamma^{2/\varepsilon_1+1}}})$.

Proof In an optimal fractional compartmental bin packing of $\widetilde{I}$, let P^* be the corresponding packing of empty compartments into bins. Hence, P^* contains $m := \mathrm{fcopt}(\widetilde{I})$ bins. Since `iterPackings`$(\widetilde{I})$ iterates over all packings of compartments into bins, $P^* \in$ `iterPackings`$(\widetilde{I})$. Since wide and tall items in $\widetilde{I}$ can be packed into the compartments of P^*, we get that z^* is not `null`. By Lemma 5, the number of bins used by NFDH to pack $I_{\mathrm{med}} \cup D$ is less than $2a(I_{\mathrm{med}} \cup D)/(1-\delta) + 3 + 1/(1-\delta)$. Therefore, the number of bins used by `skewedCPack`(I) is less than

$$\begin{aligned}
&m + \frac{2a(I_{\mathrm{med}} \cup D)}{1-\delta} + 3 + \frac{1}{1-\delta} \\
&< \left(1 + \frac{104\varepsilon_2|\mathcal{T}|}{\varepsilon_1(1-\delta)}\right)m + \frac{2\varepsilon}{1-\delta}a(I) + \frac{8\varepsilon_2}{1-\delta}\left(\frac{|\mathcal{T}|(|\mathcal{T}|+1)}{2} + \frac{6|\mathcal{T}|+2}{\varepsilon\varepsilon_1}\right) + \frac{4}{1-\delta} \\
&\qquad \text{(by Lemma 25 and } a(I_{\mathrm{med}}) \le \varepsilon a(I)\text{)} \\
&\le \left(1 + \frac{\varepsilon}{1-\delta}\right)m + \frac{2\varepsilon}{1-\delta}a(I) + \frac{1}{13(1-\delta)}\left(\frac{\varepsilon\varepsilon_1|\mathcal{T}|}{2} + 58 + \frac{\varepsilon\varepsilon_1}{2} + \frac{2}{|\mathcal{T}|}\right). \\
&\qquad \text{(by Eq. (1))}
\end{aligned}$$

By Theorem 24 and Lemma 20, we get

$$m = \mathrm{fcopt}(\widetilde{I}) < (1+4\varepsilon)\,\mathrm{fopt}(\widetilde{I}) + 2 < (1+4\varepsilon)(1+\varepsilon)\,\mathrm{opt}(I) + 4 + 8\varepsilon.$$

Therefore, the number of bins used by `skewedCPack`(I) is less than

$$\left((1+4\varepsilon)(1+\varepsilon)\left(1 + \frac{\varepsilon}{1-\delta}\right) + \frac{2\varepsilon}{1-\delta}\right)\mathrm{opt}(I)$$

$$
\begin{aligned}
&+ (4+8\varepsilon)\left(1+\frac{\varepsilon}{1-\delta}\right) + \frac{1}{13(1-\delta)}\left(\frac{\varepsilon\varepsilon_1|\mathcal{T}|}{2} + 58 + \frac{\varepsilon\varepsilon_1}{2} + \frac{2}{|\mathcal{T}|}\right)\\
&< \left(1+\left(\frac{27}{2}+\frac{13\delta}{2(1-\delta)}\right)\varepsilon\right)\mathrm{opt}(I) + \frac{\varepsilon\varepsilon_1|\mathcal{T}|}{26(1-\delta)} + \frac{1}{1-\delta}\left(16+\frac{53}{104}\right)\\
&\quad \text{(since } \varepsilon_1 \le \varepsilon \le 1/2 \text{ and } |\mathcal{T}| \ge 4 \text{ (by Lemma 21))}\\
&\le (1+14\varepsilon)\,\mathrm{opt}(I) + \frac{1}{20}\left(1+\frac{1}{\varepsilon\varepsilon_1}\right)^{2/\varepsilon_1-3} + 17.\\
&\quad \text{(since } \delta \le 1/35 \text{ and} |\mathcal{T}| \le (1+1/\varepsilon\varepsilon_1)^{2/\varepsilon_1-2} \text{ (by Lemma 21))}
\end{aligned}
$$

`iterPackings` outputs at most $|I|^{\nu}$ packings, where $\nu := \gamma^{\gamma^{2/\varepsilon_1+1}}$. For each packing P output by `iterPackings`, computing z^* takes $O(m|\mathcal{T}|/\varepsilon_1) \subseteq O_\varepsilon(|I|)$ time, where m is the number of bins in P, `greedyCPack` runs in $O(|I|\log|I| + |\mathcal{T}|m/\varepsilon_1^2 + |\mathcal{T}|^2/\varepsilon_1) \subseteq O_\varepsilon(|I|^2)$ time, and NFDH$(D \cup I_{\mathrm{med}})$ runs in $O(|I|\log|I|)$ time. Hence, the running time of $\texttt{skewedCPack}_\varepsilon(I)$ is $O_\varepsilon(|I|^{2+\nu})$. □

From Theorem 26 and Sect. 5.1, we get the following corollary.

Theorem 3 *Let $\varepsilon \in (0, 1/2]$. Let $f(x) := \varepsilon x/(104(1+1/(\varepsilon x))^{2/x-2})$. (Note that $f(x)$ increases with x, and $f(x) \le x\ \forall x \le 1/2$.) For any non-negative integer j, let $f^{(j)}(x)$ be x if $j = 0$ and $f(f^{(j-1)}(x))$ otherwise. Let $\eta := f^{\lceil 2/\varepsilon\rceil}(\varepsilon)$ and $\gamma := 1 + 1/(\varepsilon\eta)$.*

Let I be a set of η-skewed rectangular items. Then the number of bins used by $\texttt{skewedCPack}_\varepsilon(I)$ is less than

$$(1+14\varepsilon)\,\mathrm{opt}(I) + \frac{1}{20}\gamma^{2/\eta-3} + 17.$$

The running time of $\texttt{skewedCPack}_\varepsilon(I)$ is $O_\varepsilon(|I|^{2+\gamma^{\gamma^{2/\eta+1}}})$.

5.4 Proof of the Discretization Theorem

Theorem 22 (Discretization) *Given a fractional packing of items $\widetilde{J} \subseteq \widetilde{I}$ into a bin, we can remove tall and small items of total area less than ε and shift some of the remaining items to the left such that for every wide item i, we get $x_1(i), x_2(i) \in \mathcal{T}$.*

For wide items u and v in the bin, we say that $u \prec v$ iff the right edge of u is to the left of the left edge of v. Formally $u \prec v \iff x_2(u) \le x_1(v)$. We call u a *predecessor* of v. A sequence $[i_1, i_2, \ldots, i_k]$ such that $i_1 \prec i_2 \prec \ldots \prec i_k$ is called a *chain* ending at i_k. For a wide item i, define level(i) as the number of items in the longest chain ending at i. Formally, level$(i) := 1$ if i has no predecessors, and $\big(1 + \max_{j\prec i}\mathrm{level}(j)\big)$ otherwise. Let W_j be the items at level j, i.e., $W_j := \{i : \mathrm{level}(i) = j\}$. See Fig. 10 for an example. Note that the level of an item can be at most $1/\varepsilon_1 - 1$, since each wide item has width more than ε_1.

We now describe an algorithm for discretization. But first, we need to introduce two recursively-defined set families $(S_1, S_2, \ldots)$ and $(T_0, T_1, \ldots)$. Let $T_0 := \{0\}$ and $t_0 := 1$. For any $j > 0$, define

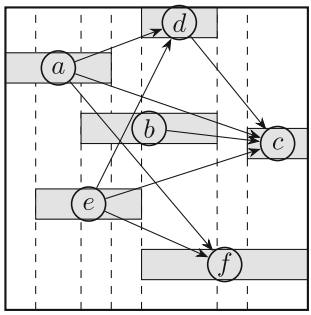


Fig. 10 Example illustrating the $\prec$ relationship between wide items in a bin. An edge is drawn from u to v iff $u \prec v$. Here $W_1 = \{a, e, b\}$, $W_2 = \{d, f\}$ and $W_3 = \{c\}$

1. $t_j := (1 + 1/\varepsilon\varepsilon_1)^{2j}$,
2. $\delta_j := \varepsilon\varepsilon_1/t_{j-1}$,
3. $S_j := T_{j-1} \cup \text{unif}(\delta_j)$,
4. $T_j := S_j \oplus R$.

Note that $\forall j > 0$, we have $T_{j-1} \subseteq S_j \subseteq T_j$ and $\delta_j^{-1} \in \mathbb{Z}$.

Lemma 27 *Let* $T'_0 := \{0\}$. *For all* $j \geq 1$, *let* $S'_j := \text{unif}(\delta_j) \oplus R^{(j-1)}$ *and* $T'_j := \text{unif}(\delta_j) \oplus R^{(j)}$. *Then* $S_j \subseteq S'_j$ *and* $T_j \subseteq T'_j$ *for all* j.

Proof by induction Base case: $T'_0 = T_0 = \{0\}$ and $S_1 = S'_1 = \text{unif}(\delta_1)$.

For all $j \geq 1$, by the induction hypothesis, $T_j := S_j \oplus R \subseteq S'_j \oplus R = T'_j$ and

$$\begin{aligned} S_{j+1} &= T_j \cup \text{unif}(\delta_{j+1}) \subseteq T'_j \cup \text{unif}(\delta_{j+1}) \\ &= (\text{unif}(\delta_j) \oplus R^{(j)}) \cup \text{unif}(\delta_{j+1}) \\ &\subseteq \text{unif}(\delta_{j+1}) \oplus R^{(j)} = S'_{j+1}. \qquad (\text{since unif}(\delta_j) \subseteq \text{unif}(\delta_{j+1})) \end{aligned}$$

□

Corollary 28 $T_{1/\varepsilon-1} \subseteq \mathcal{T}$.

Proof Recall the definition of μ and $\mathcal{T}$. Note that $\mu = \delta_{1/\varepsilon_1-1}$ and $\mathcal{T} = T'_{1/\varepsilon_1-1}$. □

Corollary 29 $|T_j| \leq |T'_j| \leq t_j$.

Proof Let $\gamma := 1 + 1/\varepsilon\varepsilon_1$. For $j = 0$, we get $|T_0| = |T'_0| = 1 = t_0$. For $j \geq 1$,

$$|T'_j| \leq \frac{1}{\delta_j} + \gamma^j = \frac{\gamma^{2j-2}}{\varepsilon\varepsilon_1} + \gamma^j \leq \frac{\gamma^{2j-1}}{\varepsilon\varepsilon_1} + \gamma^{2j-1} = \gamma^{2j} = t_j.$$

□

Our discretization algorithm proceeds in $1/\varepsilon_1 - 1$ stages, where in the jth stage, we apply two transformations to the items in the bin, called *strip-removal* and *compaction*.

Strip-removal: For each $x \in T_{j-1}$, consider a strip of width δ_j and height 1 in the bin whose left edge has coordinate x. Discard the slices of tall and small items inside the strips.

Compaction: Move all tall and small items as much towards the left as possible (imagine a gravitational force acting leftwards on the tall and small items) while keeping the wide items fixed. Then move each wide item $i \in W_j$ leftwards till $x_1(i) \in S_j$.

Observe that the algorithm maintains the following invariant: *after k stages, for each $j \in [k]$, each item $i \in W_j$ has $x_1(i) \in S_j$ (and hence $x_2(i) \in T_j$)*. This ensures that after the algorithm ends, $x_1(i), x_2(i) \in \mathcal{T}$. All that remains to prove is that the total area of items discarded during strip-removal is at most ε and that compaction is always possible.

Lemma 30 *Items discarded by strip-removal (across all stages) have total area less than ε.*

Proof In the jth stage, we create $|T_{j-1}|$ strips, and each strip has total area at most δ_j. Therefore, the area discarded in the jth stage is at most $|T_{j-1}|\delta_j \leq t_{j-1}\delta_j = \varepsilon\varepsilon_1$. Since there can be at most $1/\varepsilon_1 - 1$ stages, we discard a total area of less than ε across all stages. □

Lemma 31 *Compaction always succeeds, i.e., in the jth stage, while moving item $i \in W_j$ leftwards, no other item will block its movement.*

Proof Let $i \in W_j$. Let z be the x-coordinate of the left edge of the strip immediately to the left of item i, i.e., $z := \max(\{x \in T_{j-1} : x \leq x_1(i)\})$. For any wide item i', we have $x_2(i') \leq x_1(i) \iff i' \prec i \iff \text{level}(i') \leq j-1$. By our invariant, we get $\text{level}(i') \leq j-1 \implies x_2(i') \in T_{j-1} \implies x_2(i') \leq z$. Therefore, for every wide item i', $x_2(i') \notin (z, x_1(i)]$.

In the jth strip-removal, we cleared the strip $[z, z+\delta_j] \times [0, 1]$. If $x_1(i) \in [z, z+\delta_j]$, then i can freely move to z, and $z \in T_{j-1} \subseteq S_j$. Since no wide item has its right edge in $(z, x_1(i)]$, if $x_1(i) > z + \delta_j$, all the tall and small items whose left edge lies in $[z + \delta_j, x_1(i)]$ will move leftwards by at least δ_j during compaction. Hence, there would be an empty space of width at least δ_j to the left of item i (see Fig. 11). Therefore, we can move i leftwards to make $x_1(i)$ a multiple of δ_j, and then $x_1(i)$ would belong to S_j. □

Hence, compaction always succeeds and we get $x_1(i), x_2(i) \in \mathcal{T}$ for each wide item i. This completes the proof of the Discretization Theorem (Theorem 22).

5.5 Packing Algorithm Details

5.5.1 `iterPackings`

We describe a subroutine, called `iterPackings`$(\widetilde{I})$, that outputs all packings of empty compartments into at least $\lceil a(\widetilde{I}) \rceil$ bins and at most $|\widetilde{I}|$ bins. A packing of empty compartments in a bin is called a *configuration*. We first enumerate all configurations and then output multisets of configurations of cardinality ranging from $\lceil a(\widetilde{I}) \rceil$ to $|\widetilde{I}|$.

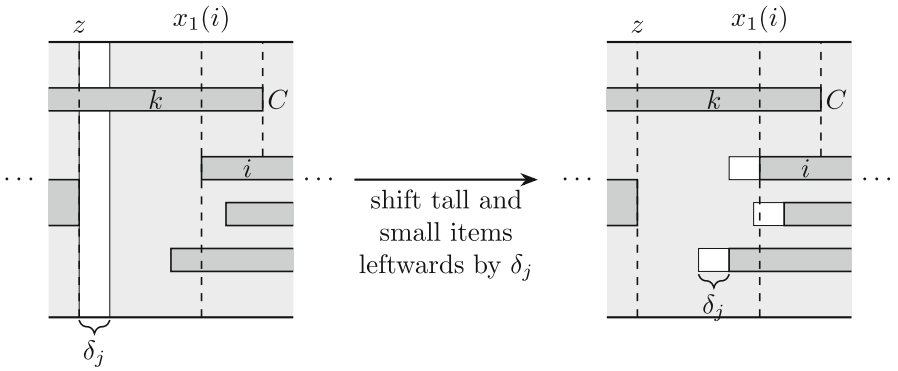


Fig. 11 This figure shows a region in the bin in the vicinity of item $i \in W_j$. It illustrates how shifting tall and small items during compaction in the jth stage creates a free space of width δ_j to the left of some wide items, including i. Wide items are shaded dark and the lightly shaded region potentially contains tall and small items. Note that some tall and small items in the region C may be unable to shift left because item k is blocking them. All other tall and small items in this figure to the right of z can shift left by δ_j

There can be at most $n_W := 3(1/\varepsilon_1 - 1)|\mathcal{T}| + 1$ wide compartments in a bin. For each wide compartment C, $x_1(C) \in \mathcal{T}$, $x_2(C) \in \mathcal{T} - \{0\} \cup \{1\}$, $y_1(C) \in \text{unif}(\varepsilon_{\text{cont}})$, and $y_2(C) \in \text{unif}(\varepsilon_{\text{cont}}) - \{0\} \cup \{1\}$. (Recall the definitions of x_1, x_2, y_1, y_2 from Sect. 5.2.2, unif from Sect. 5.2.1, $\varepsilon_{\text{cont}} := \varepsilon\varepsilon_1/6|\mathcal{T}|$ by Definition 3 from Sect. 5.2.3.)

Hence, for $n_C := (|\mathcal{T}|^2/\varepsilon_{\text{cont}}^2)^{n_W}$, there are at most n_C different configurations and we can enumerate them in $O(n_C)$ time. Furthermore,

$$n_C := \left(\frac{|\mathcal{T}|^2}{\varepsilon_{\text{cont}}^2}\right)^{n_W} \leq \left(\frac{6|\mathcal{T}|^2}{\varepsilon\varepsilon_1}\right)^{6|\mathcal{T}|/\varepsilon_1} \leq \left(1 + \frac{1}{\varepsilon\varepsilon_1}\right)^{\left(1+\frac{1}{\varepsilon\varepsilon_1}\right)^{2/\varepsilon_1+1}}.$$

Since each configuration can have at most $|\widetilde{I}|$ bins, the number of combinations of configurations is at most $(|\widetilde{I}| + 1)^{n_C}$. Therefore, we can output all possible bin packings of empty compartments in $O(|\widetilde{I}|^{n_C+1})$ time by iterating over all elements of $\{0, 1, \ldots, |\widetilde{I}|\}^{n_C}$. This completes the description of `iterPackings`.

5.5.2 Fractionally Packing Items Into Compartments

For each bin packing P of empty compartments, we try to fractionally pack the items $\widetilde{I}$ into the bins. To do this, we create a feasibility linear program, called $\text{FP}(\widetilde{I}, P)$, that is feasible iff wide and tall items in $\widetilde{I}$ can be fractionally packed into the compartments in P. If $\text{FP}(\widetilde{I}, P)$ is feasible, then small items can also be fractionally packed since P contains at least $a(\widetilde{I})$ bins. Recall that small items can be sliced both vertically and horizontally, and they can be packed both inside and outside compartments.

Let $w'_1, w'_2, \ldots, w'_p$ be the distinct widths of wide compartments in P. Let U_j be the set of wide compartments in P having width w'_j. Let $h(U_j)$ be the sum of heights of the compartments in U_j. By Definition 3, we know that $p \leq |\mathcal{T}|(|\mathcal{T}|+1)/2 \leq |\mathcal{T}|^2$. Let $w_1, w_2, \ldots, w_r$ be the distinct widths of items in $\widetilde{W}$ (recall that $\widetilde{W}$ is the set of

wide items in $\widetilde{I}$). Let $\widetilde{W}_j$ be the items in $\widetilde{W}$ having width w_j. Let $h(\widetilde{W}_j)$ be the sum of heights of all items in $\widetilde{W}_j$. By Claim 42 in Appendix D, we get $r \leq 1/\varepsilon\varepsilon_1$.

Let $C := [C_0, C_1, \ldots, C_r]$ be a vector, where $C_0 \in [p]$ and $C_j \in \mathbb{Z}_{\geq 0}$ for $j \in [r]$. C is called a *wide configuration* iff $w(C) := \sum_{j=1}^{r} C_j w_j \leq w'_{C_0}$. Intuitively, a wide configuration C represents a set of wide items that can be placed side-by-side into a compartment of width w'_{C_0}. Let $\mathcal{C}$ be the set of all wide configurations. Then $|\mathcal{C}| \leq p/\varepsilon_1^r$, which is a constant. Let $\mathcal{C}_j := \{C \in \mathcal{C} : C_0 = j\}$.

To fractionally pack $\widetilde{W}$ into wide compartments, we must determine the height of each configuration. Let $x \in \mathbb{R}_{\geq 0}^{|\mathcal{C}|}$ be a vector where x_C denotes the height of configuration C. Then $\widetilde{W}$ can be packed into wide compartments according to x iff x is a feasible solution to the following feasibility linear program, named $\mathrm{FP}_W(\widetilde{W}, P)$:

$$\sum_{C \in \mathcal{C}} C_j x_C \geq h(\widetilde{W}_j) \,\forall j \in [r] \qquad (\widetilde{W}_j \text{ should be covered})$$

$$\sum_{C \in \mathcal{C} \text{ and } C_0 = j} x_C \leq h(U_j) \forall j \in [p] \qquad (\mathcal{C}_j \text{ should fit in } U_j)$$

$$x_C \geq 0 \forall C \in \mathcal{C}$$

Let x^* be an extreme point solution to $\mathrm{FP}_W(\widetilde{W}, P)$ (if $\mathrm{FP}_W(\widetilde{W}, P)$ is feasible). By Rank Lemma,[3] at most $p + r$ entries of x^* are non-zero. Since the number of variables and constraints is constant, x^* can be computed in constant time.

Let $\widetilde{H}$ be the set of tall items in $\widetilde{I}$. Items in $\widetilde{H}$ have at most $1/\varepsilon\varepsilon_1$ distinct heights. Let there be q distinct heights of tall compartments in P. By Definition 3, we know that $q \leq 1/\varepsilon_{\mathrm{cont}} = 6|\mathcal{T}|/\varepsilon\varepsilon_1$. We can similarly define *tall configurations* and we can similarly define a feasibility linear program for tall items, named $\mathrm{FP}_H(\widetilde{H}, P)$. $\widetilde{H}$ can be packed into tall compartments in P iff $\mathrm{FP}_H(\widetilde{H}, P)$ is feasible. Let y^* be an extreme point solution to $\mathrm{FP}_H(\widetilde{H}, P)$. Then y^* can be computed in constant time and y^* has at most $q + 1/\varepsilon\varepsilon_1$ positive entries.

Therefore, $\widetilde{I}$ can be packed into P iff the feasibility linear program $\mathrm{FP}(\widetilde{I}, P) := \mathrm{FP}_W(\widetilde{W}, P) \times \mathrm{FP}_H(\widetilde{H}, P)$ is feasible. We can show that $z^* := (x^*, y^*)$ is an extreme point of $\mathrm{FP}(\widetilde{I}, P)$ iff x^* is an extreme point of $\mathrm{FP}_W(\widetilde{W}, P)$ and y^* is an extreme point of $\mathrm{FP}_H(\widetilde{H}, P)$.

Let there be m bins in P. Then in $O(mn_W)$ time, we can find w'_j and $h(U_j)$ for all $j \in [p]$. In $O(p/\varepsilon_1^r)$ time, we can compute $\mathcal{C}$. $\mathrm{FP}_W(\widetilde{W}, P)$ has $p + r$ constraints and $|\mathcal{C}|$ variables. Hence, we can find an extreme point solution to it using the simplex algorithm [44], and the running time would be a constant (though it will be a very large constant that is a function of ε), provided an appropriate pivoting rule is used [45]. Similarly, it takes $O(mn_H)$ time to scan P for information relevant to $\mathrm{FP}_H(\widetilde{H}, P)$, and we can use the simplex algorithm to solve it in constant time. Hence, given $\widetilde{I}$ and P, we can obtain an extreme-point solution to $\mathrm{FP}(\widetilde{I}, P)$ in $O(m|\mathcal{T}|/\varepsilon_1) + O_\varepsilon(1)$ time.

(Although there are polynomial-time algorithms for solving linear programs, they are not strongly polynomial-time. Since we want to get an extreme point solution

[3] Rank Lemma: the number of non-zero variables in an extreme-point solution to a linear program is at most the number of non-trivial constraints [42, Lemma 2.1.4]..

and we do not want to worry about the bit-length of the input, we use the simplex algorithm).

The solution (x^*, y^*) shows us how to split each compartment into *shelves*, where each shelf corresponds to a configuration C. A shelf in a wide compartment having configuration C can be split into C_j *containers* of width w_j and one container of width $w'_{C_0} - w(C)$. Shelves in tall compartments can be split into containers analogously. Let there be m bins in P. After splitting the configurations across compartments, we get at most $m_{\text{shelves}} := p + q + 2/\varepsilon\varepsilon_1 + m(n_W + n_H)$ shelves.

5.5.3 `greedyCPack`

Let there be m bins in a packing P of empty compartments into bins. Suppose it is possible to pack $\widetilde{I}$ into P. Let (x^*, y^*) be an extreme-point solution to $\text{FP}(\widetilde{I}, P)$. This gives us a fractional compartmental packing of $\widetilde{I}$ into m bins. We now show how to convert this to a non-fractional compartmental packing by removing some items of small total area. Formally, we give an algorithm called `greedyCPack`$(\widetilde{I}, P, (x^*, y^*))$. It returns a pair (Q, D), where Q is a (non-fractional) compartmental bin packing of items $\widetilde{I} - D$, where the compartments in the bins are as per P. D is called the set of discarded items, and we prove that $a(D)$ is small.

For a configuration C in a wide compartment, there is a container of width $w'_{C_0} - w(C)$ available for packing small items. Similarly, there are containers inside tall compartments into which we can pack small items. Hence, there are m_{shelves} containers available inside compartments for packing small items.

Lemma 32 (Similar to Lemma 3.2 in [4]) *Let there be a set I of rectangles packed inside a bin. Then there is a polynomial-time algorithm that can decompose the empty space in the bin into at most $3|I| + 1$ rectangles by making horizontal cuts only.*

Proof Extend the top and bottom edge of each rectangle leftwards and rightwards till they hit another rectangle or an edge of the bin. This decomposes the empty region into rectangles R. See Fig. 12.

For each rectangle $i \in I$, the top edge of i is the bottom edge of a rectangle in R, the bottom edge of i is the bottom edge of two rectangles in R. Apart from possibly the rectangle in R whose bottom edge is at the bottom of the bin, the bottom edge of every rectangle in R is either the bottom or top edge of a rectangle in I. Therefore, $|R| \leq 3|I| + 1$. □

By Lemma 32, we can partition the space outside compartments into at most $m(3(n_W + n_H) + 1)$ containers. Therefore, the total number of containers available for packing small items is at most $m_{\text{small}} := m_{\text{shelves}} + m(3(n_W + n_H) + 1)$. These containers reserved for packing small items are called *small containers*.

Greedily assign small items to small containers, i.e., keep assigning small items to a container till the area of items assigned to it is at least the area of the container, and then resume from the next container. Each small item will get assigned to some container. For each small container C, pack the largest possible prefix of the assigned items using the Next-Fit Decreasing Height (NFDH) algorithm. By Lemma 4, the area of unpacked

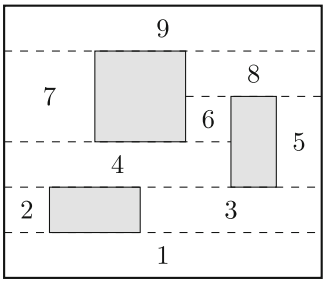


Fig. 12 Using horizontal cuts to partition the empty space around the 3 items into 9 rectangular regions

items would be less than $\varepsilon_2 + \delta + \varepsilon_2\delta$. Summing over all containers, we get that the total area of unpacked small items is less than $(\varepsilon_2 + \delta + \varepsilon_2\delta)m_{\text{small}} \leq 3\varepsilon_2 m_{\text{small}}$.

For each j, greedily assign wide items from $\widetilde{W}_j$ to containers of width w_j, i.e., keep assigning items till the height of items exceeds the height of the container. Each wide item will get assigned to some container. Then discard the last item from each container. For each shelf in a wide compartment having configuration C, the total area of items we discard is at most $\delta w(C)$. Similarly, we can discard tall items of area at most $\delta h(C)$ from each shelf in a tall compartment having configuration C. Hence, across all configurations, we discard wide and tall items of area at most $\delta m_{\text{shelves}}$.

Lemma 25 *Let* $(Q, D) := \texttt{greedyCPack}(\widetilde{I}, P, z^*)$. *Let there be* m *bins in* P. *Then*

$$a(D) < 4\varepsilon_2 \left(\frac{13|\mathcal{T}|}{\varepsilon_1} m + \frac{|\mathcal{T}|(|\mathcal{T}|+1)}{2} + \frac{6|\mathcal{T}|}{\varepsilon\varepsilon_1} + \frac{2}{\varepsilon\varepsilon_1} \right),$$

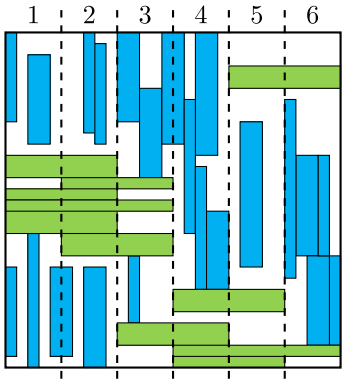
and $\texttt{greedyCPack}(\widetilde{I}, P, z^*)$ *runs in time*

$$O\left(|\widetilde{I}| \log |\widetilde{I}| + \frac{|\mathcal{T}|}{\varepsilon_1^2} m + \frac{|\mathcal{T}|^2}{\varepsilon_1} \right).$$

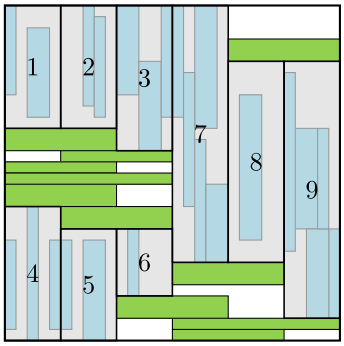
Proof Recall that $n_H := (1/\varepsilon_1 - 1)|\mathcal{T}|$, $n_W := 3n_H + 1$, $p \leq |\mathcal{T}|(|\mathcal{T}|+1)/2$, $q \leq 1/\varepsilon_{\text{cont}} = 6|\mathcal{T}|/\varepsilon\varepsilon_1$, and $m_{\text{shelves}} := p + q + 2/\varepsilon\varepsilon_1 + m(n_W + n_H))$. Hence,

$$\begin{aligned} a(D) &< 3\varepsilon_2 m_{\text{small}} + \delta m_{\text{shelves}} \\ &\leq \varepsilon_2(4m_{\text{shelves}} + m(9(n_W + n_H) + 3)) \\ &\leq \varepsilon_2(4(p + q + 2/\varepsilon\varepsilon_1) + m(13(n_W + n_H) + 3)) \\ &\leq 4\varepsilon_2 \left(\frac{13|\mathcal{T}|}{\varepsilon_1} m + \frac{|\mathcal{T}|(|\mathcal{T}|+1)}{2} + \frac{6|\mathcal{T}|}{\varepsilon\varepsilon_1} + \frac{2}{\varepsilon\varepsilon_1} \right). \end{aligned}$$

To implement `greedyCPack`, we first sort the small items by height. This takes $O(|\widetilde{I}| \log |\widetilde{I}|)$ time. Then the time to (non-fractionally) pack items into containers is $O(m_{\text{cont}} + |\widetilde{I}|)$, where m_{cont} is the total number of containers. Hence, the running time



(a) A packing of items in a bin. Wide items are green and tall items are blue. Draw vertical lines at x-coordinates from $\mathcal{T} - \{0\}$. They divide the bin into columns. In this figure, we have 6 columns.



(b) Wide items divide each column into *cells*. Each cell containing a tall item is called a *tall cell*. There are 9 tall cells in this figure, which are shaded gray.

Fig. 13 Creating tall cells in a bin

of `greedyCPack` is $O(|\widetilde{I}| \log |\widetilde{I}| + m_{\text{cont}})$. Furthermore,

$$m_{\text{cont}} \leq m_{\text{shelves}}/\varepsilon_1 + m_{\text{small}} \in O\left(\frac{|\mathcal{T}|}{\varepsilon_1^2} m + \frac{|\mathcal{T}|^2}{\varepsilon_1}\right).$$

□

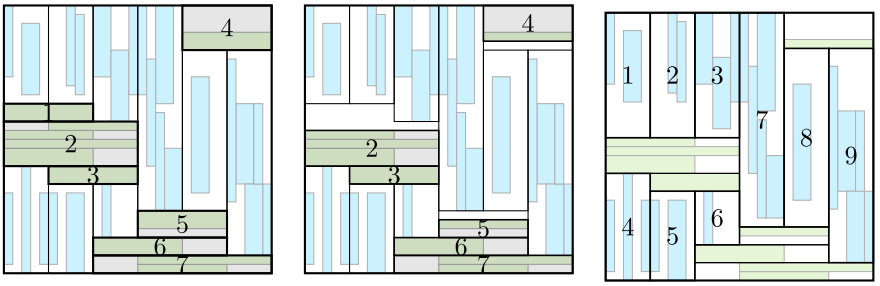
5.6 Compartmentalizing a Discretized Packing

Recall Lemma 32, proved in Sect. 5.5.3. Also see Fig. 12 in Sect. 5.5.3 for an example application.

Lemma 32 (Similar to Lemma 3.2 in [4]) *Let there be a set I of rectangles packed inside a bin. Then there is a polynomial-time algorithm that can decompose the empty space in the bin into at most $3|I| + 1$ rectangles by making horizontal cuts only.*

Lemma 23 *If $x_1(i), x_2(i) \in \mathcal{T}$ for each wide item i in a bin, then by removing wide and small items of area less than ε, we can get a fractional compartmental packing of the remaining items into a bin.*

Proof Draw vertical lines in the bin at the x-coordinates in $\mathcal{T} - \{0\}$. This splits the bin into $|\mathcal{T}|$ columns (see Fig. 13a). Each column has 0 or more wide items crossing it. These wide items divide the column into cells. A cell is called tall iff it contains a tall item (see Fig. 13b). There can be at most $1/\varepsilon_1 - 1$ tall cells in a column, so there can be at most $(1/\varepsilon_1 - 1)|\mathcal{T}|$ tall cells in the bin.



(a) Partition the space outside tall cells into rectangular regions by extending the horizontal edges of tall cells (see Lemma 32). Each rectangular region containing a wide item is called a *box*. There are 7 boxes in this figure, which are shaded gray.

(b) For each box, discard some items and shift horizontal edges to make their y-coordinates multiples of $\varepsilon_{\text{cont}}$. Boxes that continue to contain a wide item are now wide compartments.

(c) Wide compartments divide each column into rectangular regions. Each such region containing a tall item is a tall compartment. There are 9 tall compartments in this figure.

Fig. 14 Obtaining compartments

By Lemma 32, we can use horizontal cuts to partition the space outside tall cells into at most $3(1/\varepsilon_1 - 1)|\mathcal{T}| + 1$ rectangular regions (this may slice some wide items). See Fig. 14a. If a region contains a wide item, call it a box.

For each box i, slice and discard some items from the bottom of the box and increase $y_1(i)$ so that it becomes a multiple of $\varepsilon_{\text{cont}}$. Then slice and discard some items from the top of the box and reduce $y_2(i)$ so that it becomes a multiple of $\varepsilon_{\text{cont}}$. The total area of items discarded is less than $2\varepsilon_{\text{cont}}$. If i continues to contain a wide item, it becomes a wide compartment. Now all wide items belong to some wide compartment (see Fig. 14b).

Each column has 0 or more wide compartments crossing it. These wide compartments divide the column into rectangular regions. Each region that contains a tall item is a tall compartment (see Fig. 14c).

Therefore, by removing wide and small items of area less than $6|\mathcal{T}|\varepsilon_{\text{cont}}/\varepsilon_1 \leq \varepsilon$, we get a compartmental packing of items where there are at most $(1/\varepsilon_1 - 1)|\mathcal{T}|$ tall compartments and at most $3(1/\varepsilon_1 - 1)|\mathcal{T}| + 1$ wide compartments. □

Funding Arindam Khan is supported by the Pratiksha Trust Young Investigator Award and Google ExploreCSR grant.

Declarations

Conflict of interest Eklavya Sharma is a student at the University of Illinois, Urbana-Champaign. The following members of Algorithmica's editorial board are affiliated with the same university: Timothy M. Chan. Arindam Khan was a postdoctoral researcher at Technical University, Munich, Germany from November 2017 to December 2018. The following members of Algorithmica's editorial board are affiliated with the same university: Susanne Albers, Harald Raecke.

Appendix A APoG for the Rotational Case

Theorem 33 *Let* APoG^{nr} *and* APoG^{r} *be the APoG for the non-rotational and rotational versions, respectively, restricted to the* δ*-skewed case. Then* $\text{APoG}^{\text{r}} \leq \text{APoG}^{\text{nr}}$.

Proof For a set I of δ-skewed rectangular items, let $\text{opt}^{\text{nr}}(I)$ and $\text{opt}^{\text{r}}(I)$ be the minimum number of bins needed to pack I in the non-rotational and rotational versions, respectively. Let $\text{opt}_g^{\text{nr}}(I)$ and $\text{opt}_g^{\text{r}}(I)$ be the minimum number of guillotinable bins needed to pack I in the non-rotational and rotational versions, respectively. Assume w.l.o.g. that the bin has width and height at least 1.

Let I be any set of δ-skewed items. Let K be the corresponding rotated items in the optimal rotational packing of I, i.e., $\text{opt}^{\text{r}}(I) = \text{opt}^{\text{nr}}(K)$. Then

$$\begin{aligned} \text{opt}_g^{\text{r}}(I) &\leq \text{opt}_g^{\text{nr}}(K) \\ &\leq \text{APoG}^{\text{nr}}\, \text{opt}^{\text{nr}}(K) + c \qquad (c \text{ is a constant}) \\ &= \text{APoG}^{\text{nr}}\, \text{opt}^{\text{r}}(I) + c. \end{aligned}$$

Hence, we get $\text{APoG}^{\text{r}} \leq \text{APoG}^{\text{nr}}$. □

Appendix B `skewedCPack` with Item Rotations

In this section, we briefly explain changes to `skewedCPack` and its analysis so that they work for the case where items can be rotated by 90° and the ratio of the bin's width and height is a constant.

For the rotational version of the problem, we assume w.l.o.g. that the width of the bin is 1 and the height of the bin is at least 1, because we can rotate the bin and scale its width and height equally. We also assume that the height of the bin is a constant. Since the input is δ-skewed, we can assume w.l.o.g. that every item in the input has width at most δ, since otherwise we can rotate the item by 90°.

To handle the rotational case, we do not require any change in Sect. 5.1.

The structural theorem in Sect. 5.2 doesn't require any conceptual modifications. The only change is that the number of tall compartments in a tall cell can be more than $1/\varepsilon_1 - 1$. Specifically, if the height of the bin is H, the number of tall compartments in a tall cell is now upper-bounded by H/ε_1. (Hence, for the number of compartments to be a constant, we require the bin's height to be a constant). This will increase the running time of `iterPackings`, but there will be no change to Theorem 24.

The feasibility linear program of Sect. 5.5.2 will have to change to take item rotations into account. Instead of using two programs— FP_W and FP_H—which fractionally pack wide and tall items separately, we use just one program which will also decide which items to rotate. To do this, we allow an item to belong to both wide and tall configurations. The number of constraints in the feasibility linear program will be a constant that depends on ε and ε_1.

The `greedyCPack` algorithm of Sect. 5.5.3 will remain the same, but the total area of discarded items will be slightly different because the number of compartments

in a bin can now be larger. Finally, the AAR of `skewedCPack` for the rotational version will be $1 + \Theta(1)\varepsilon$ by the same kind of analysis as in Sect. 5.3.

Appendix C Next-Fit Decreasing Height (NFDH)

In this section, we give proofs of the results in Sect. 2 related to NFDH algorithm [15].

C.1 Next-Fit

Our proofs rely on the Next-Fit algorithm for classical bin packing.

In the classical bin packing problem, we are given a set I of items, and an infinite supply of bins of capacity C, where each item i has a size $s_i \in (0, C]$ associated with it. We need to pack the items into the minimum number of bins such that in each bin, items have total size at most C. Formally, for a set J of items, define $\text{size}(J) := \sum_{i \in J} s_i$. Then our goal is to partition the items I into sets $B_1, B_2, \ldots, B_m$ such that m is minimized and $\text{size}(B_j) \leq C$ for all $j \in [m]$.

In the Next-Fit algorithm, we repeatedly pack items into a bin till it is possible to do so. Then we close that bin, open a new bin, and resume. See Algorithm 3 for a more precise description.

Algorithm 3 `NextFit`(I, C): Takes a classical bin packing instance (I, C) as input (I is a set of items, C is the bin capacity) and packs them into bins, where bins are numbered from 1 onwards.

```
Set currBin = 1 and totSize = 0.
for i ∈ I do
    if totSize + s_i ≤ C then
        totSize += s_i
    else
        totSize = 0
        currBin += 1
    end if
    Pack item i in bin number currBin.
end for
```

Lemma 34 [46] *Let I be a classical bin packing instance where bin capacity is C. Then the number of bins used by the Next-Fit algorithm* [46] *to pack I is less than* $2\,\text{size}(I)/C + 1$.

Lemma 35 [46] *Let I be a classical bin packing instance, where each item has size at most ε and bin capacity is C. Then the number of bins used by the Next-Fit algorithm* [46] *to pack I is less than* $\text{size}(I)/(C - \varepsilon) + 1$.

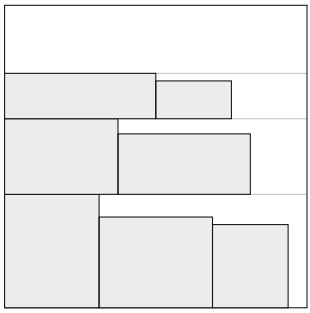


Fig. 15 7 items packed into 3 shelves by NFDH

C.2 Description of NFDH

Suppose we have rectangular items I that we want to pack into bins of width W and height H. The Next-Fit Decreasing Height (NFDH) algorithm [15] packs items into rectangular containers of width W called *shelves*, and packs the shelves into bins. For each shelf, the bottom edge of each item in it touches the bottom edge of the shelf. See Fig. 15 for an example output of NFDH.

The NFDH algorithm works roughly as follows:

1. Sort the items in non-increasing order of height.
2. Create a shelf whose height equals the height of the first item. Pack the largest prefix of I in the shelf from left-to-right. Then recursively pack the remaining items into more shelves.
3. Treat the shelves as a classical bin packing instance where the size of a shelf equals its height, and bin capacity is H. Pack the shelves into bins using `NextFit`.

See Algorithm 4 for an equivalent but more precise description of NFDH.

C.3 Results on NFDH

Lemma 36 [15] *Let I be a set of rectangles of width at most 1. Then I can be packed (without rotation) into a rectangular bin of width 1 and height less than $2a(I) + max_{i\in I} h(i)$ using NFDH.*

Lemma 37 *Let I be a set of rectangles of width at most ε. Then I can be packed (without rotation) into a rectangular bin of width W and height less than $a(I)/(W - \varepsilon) + max_{i\in I} h(i)$ using NFDH.*

Proof Let there be p shelves output by NFDH. Let S_j be the items in the jth shelf. Let h_j be the height of the jth shelf. Let H be the sum of heights of all the shelves.

For $j \in [p-1]$, in the jth shelf, the total width of items is more than $(W - \varepsilon)$ and each item has height more than h_{j+1}. Therefore, $a(S_j) > h_{j+1}(W - \varepsilon)$.

Let H be the sum of heights of all the shelves. Then $a(I) > \sum_{i=1}^{p-1} a(S_j) \geq \sum_{i=1}^{p-1} h_{j+1}(W - \varepsilon) \geq (W - \varepsilon)(H - h_1)$. This implies $H < a(I)/(W - \varepsilon) + h_1$. □

Lemma 4 *Let I be a set of items where each item i has $w(i) \leq \delta_W$ and $h(i) \leq \delta_H$. NFDH can pack I into a bin of width W and height H if $a(I) \leq (W - \delta_W)(H - \delta_H)$.*

Algorithm 4 NFDH(I, W, H): Takes a set of rectangular items I as input. Packs the items into shelves and packs the shelves into bins. At any point in time, at most one bin is open and at most one shelf is open, where *open* means 'available to be used for packing'. The remaining bins and shelves are closed, i.e., nothing can be added or removed from them.

```
Set shelfH = 0, x = 0, y = 0.
Sort I in decreasing order of height.
Let shelfH be the height of the first item in I.
Open a bin. Open a shelf of height shelfH and place it at the bottom of the open bin.
for i ∈ I do
    if x + w(i) > W then
        x = 0
        y += shelfH
        shelfH = h(i)
        Close the current shelf and open a new shelf of height shelfH.
        if y + h(i) > H then
            y = 0
            Close the current bin and open a new bin.
        end if
        Place the open shelf at position y in the open bin.
    end if
    x += w(i).
    Pack item i in the open shelf at position x.
    // Open shelf's bottom edge is at distance y from open bin's bottom edge.
end for
```

Proof (Also proved as Lemma 2(iii) in [47].)

NFDH packs the items into shelves of width W. Let $\widetilde{H}$ be the total height of the shelves. By Lemma 37, we get $\widetilde{H} < a(I)/(W - \delta_W) + \delta_H \leq H$. Therefore, NFDH can fit I into the bin. □

Lemma 38 *Let I be a set of rectangular items where each item has height at most δ. Then the number of bins required by NFDH to pack I is less than $(2a(I) + 1)/(1 - \delta)$.*

Proof The bin packing version of NFDH first packs I into shelves and then packs the shelves into bins using Next-Fit. Let H be the sum of heights of all the shelves. By Lemma 36, $H < 2a(I) + \delta$. By Lemma 35, the number of bins is less than $1 + H/(1 - \delta) < (2a(I) + 1)/(1 - \delta)$. □

Lemma 39 *Let I be a set of rectangular items where each item has width at most δ. Then the number of bins required by NFDH to pack I is less than $2a(I)/(1 - \delta) + 3$.*

Proof The bin packing version of NFDH first packs I into shelves and then packs the shelves into bins using Next-Fit. Let H be the sum of heights of all the shelves. By Lemma 37, $H < a(I)/(1 - \delta) + 1$. By Lemma 34, the number of bins is less than $2H + 1 < 2a(I)/(1 - \delta) + 3$. □

Lemma 40 *Let I be a set of rectangular items where each item has width at most δ_W and height at most δ_H. Then the number of bins required by NFDH to pack I is at most $a(I)/(1 - \delta_W)(1 - \delta_H) + 1/(1 - \delta_H)$.*

Proof The bin packing version of NFDH first packs I into shelves and then packs the shelves into bins using Next-Fit. Let H be the sum of heights of all the shelves. By Lemma 37, $H < a(I)/(1-\delta_W) + \delta_H$. By Lemma 35, the number of bins is less than $1 + H/(1-\delta_H) < a(I)/(1-\delta_W)(1-\delta_H) + 1/(1-\delta_H)$. □

Appendix D Linear Grouping

In this section, we describe the *linear grouping* technique [3, 24] for wide and tall items.

Let ε and ε_1 be constants in $(0, 1)$ such that $(\varepsilon\varepsilon_1)^{-1} \in \mathbb{Z}$. Let W be a set of items where each item has width more than ε_1. We describe an algorithm called `lingroupWide` that takes W, ε and ε_1 as input and returns the set $\widehat{W}$ as output, where $\widehat{W}$ is obtained by increasing the width of each item in W.

`lingroupWide`$(W, \varepsilon, \varepsilon_1)$ first arranges the items W in non-increasing order of width and stacks them one-over-the-other (i.e., the widest item in W is at the bottom). Let h_L be the height of the stack. Let $y(i)$ be the y-coordinate of the bottom edge of item i. Split the stack into sections of height $\varepsilon\varepsilon_1 h_L$ each. For $j \in [1/\varepsilon\varepsilon_1]$, let w_j be the width of the widest item intersecting the jth section, i.e.,

$$w_j := \max(\{w(i) : i \in W \text{ and } (y(i), y(i)+h(i)) \cap ((j-1)\varepsilon\varepsilon_1 h_L, j\varepsilon\varepsilon_1 h_L) \neq \emptyset\}).$$

Round up the width of each item i to the smallest w_j that is at least $w(i)$ (see Fig. 16). Let W_j be the items whose width got rounded to w_j and let $\widehat{W}_j$ be the resulting rounded items. (There may be ties, i.e., there may exist $j_1 < j_2$ such that $w_{j_1} = w_{j_2}$. In that case, define $W_{j_2} := \widehat{W}_{j_2} := \emptyset$. This ensures that all W_j are disjoint.) Finally, define $\widehat{W} := \bigcup_j \widehat{W}_j$. Then $\widehat{W}_j$ is called the jth linear group of $\widehat{W}$.

We can similarly define the algorithm `lingroupTall`. Let H be a set of items where each item has height more than ε_1. `lingroupTall` takes H, ε and ε_1 as input and returns $\widehat{H}$, where $\widehat{H}$ is obtained by increasing the height of each item in H.

Lemma 41 `lingroupWide`$(W, \varepsilon, \varepsilon_1)$ *runs in* $O(|W| \log |W| + 1/\varepsilon\varepsilon_1)$ *time.* `lingroupTall`$(H, \varepsilon, \varepsilon_1)$ *runs in* $O(|H| \log |H| + 1/\varepsilon\varepsilon_1)$ *time.*

Proof Let $n := |W|$. Arranging items in W in non-increasing order of width takes $O(n \log n)$ time. Computing $w_1, w_2, \ldots$ takes $O(n + 1/\varepsilon\varepsilon_1)$ time. Rounding up the width of all items takes $O(n + 1/\varepsilon\varepsilon_1)$ time (by iterating over sorted lists W and $[w_1, w_2, \ldots]$). Hence, the total running time of `lingroupWide`$(W, \varepsilon, \varepsilon_1)$ is $O(n \log n + 1/\varepsilon\varepsilon_1)$. The running time of `lingroupTall` can be derived analogously. □

Claim 42 *Items in* `lingroupWide`$(W, \varepsilon, \varepsilon_1)$ *have at most* $1/(\varepsilon\varepsilon_1)$ *distinct widths. Items in* `lingroupTall`$(H, \varepsilon, \varepsilon_1)$ *have at most* $1/(\varepsilon\varepsilon_1)$ *distinct heights.*

Lemma 43 *Let W, H and S be sets of items, where items in W have width more than ε_1 and items in H have height more than ε_1. Let* $\widehat{W} :=$ `lingroupWide`$(W, \varepsilon, \varepsilon_1)$ *and* $\widehat{H} :=$ `lingroupTall`$(H, \varepsilon, \varepsilon_1)$. *If we allow slicing items in $\widehat{W}$ and $\widehat{H}$ using*

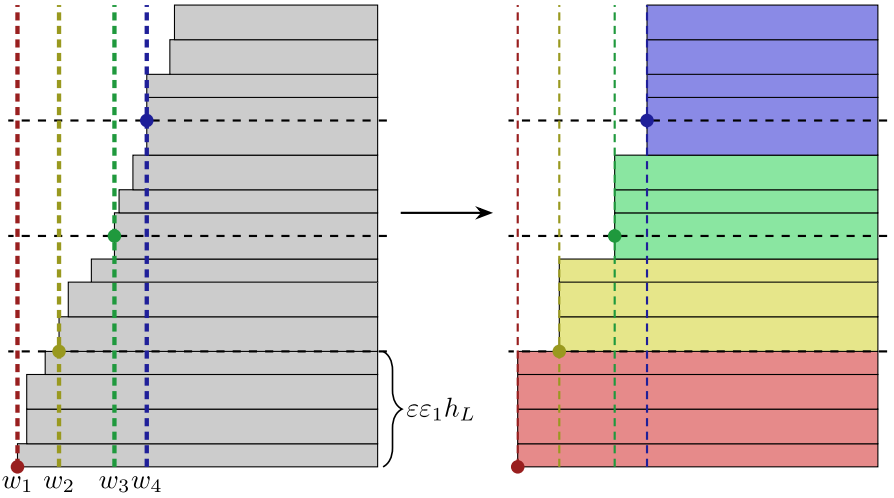


Fig. 16 Example invocation of `lingroupWide` for $\varepsilon = \varepsilon_1 = 1/2$. Each linear group of $\widehat{W}$ is given a unique color

horizontal and vertical cuts, respectively, then we can pack $\widehat{W} \cup \widehat{H} \cup S$ *into less than* $(1+\varepsilon)\,\mathrm{opt}(W \cup H \cup S) + 2$ *bins.*

Proof `lingroupWide`$(W, \varepsilon, \varepsilon_1)$ arranges the items W in non-increasing order of width and stacks them one-over-the-other. Let h_L be the height of the stack. Split the stack into sections of height $\varepsilon\varepsilon_1 h_L$ each. For $j \in [1/\varepsilon\varepsilon_1]$, let W'_j be the slices of items that lie between heights $(j-1)\varepsilon\varepsilon_1 h_L$ and $j\varepsilon\varepsilon_1 h_L$. Let $\widehat{W}'_j$ be the corresponding rounded items from W'_j. Similarly define H'_j and $\widehat{H}'_j$.

Consider the optimal packing of $W \cup H \cup S$. To convert this to a packing of $\widehat{W} \cup \widehat{H} \cup S - (\widehat{W}'_1 \cup \widehat{H}'_1)$, unpack W'_1 and H'_1, and for each $j \in [1/\varepsilon\varepsilon_1 - 1]$, pack $\widehat{W}'_{j+1}$ in the place of W'_j and pack $\widehat{H}'_{j+1}$ in the place of H'_j, possibly after slicing the items. This gives us a packing of $\widehat{W} \cup \widehat{H} \cup S - (\widehat{W}'_1 \cup \widehat{H}'_1)$ into m_1 bins, where $m_1 \le \mathrm{opt}(W \cup H \cup S)$.

We can pack $\widehat{H}'_1$ by stacking the items side-by-side on the base of bins. We can pack $\widehat{W}'_1$ into bins by stacking the items one-over-the-other. Recall that h_L is the total height of items in $\widehat{W}$. Let w_L be the total width of items in $\widehat{H}$. This gives us a packing of $\widehat{W}'_1 \cup \widehat{H}'_1$ into $m_2 := \lceil \varepsilon\varepsilon_1 h_L \rceil + \lceil \varepsilon\varepsilon_1 w_L \rceil$ bins. Since $\mathrm{opt}(W \cup H \cup S) \ge a(W) + a(H) \ge \varepsilon_1(h_L + w_L)$, we get $m_2 = \lceil \varepsilon\varepsilon_1 h_L \rceil + \lceil \varepsilon\varepsilon_1 w_L \rceil < \varepsilon\,\mathrm{opt}(W \cup H \cup S) + 2$. Therefore, we get a packing of $\widehat{W} \cup \widehat{H} \cup S$ into $m_1 + m_2$ bins and $m_1 + m_2 < (1+\varepsilon)\,\mathrm{opt}(W \cup H \cup S) + 2$. □

References

1. Khan, A., Sharma, E.: Tight approximation algorithms For Geometric Bin Packing with skewed items. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 207, pp.

22–12223. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2021.22
2. Hoberg, R., Rothvoss, T.: A logarithmic additive integrality gap for bin packing. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 2616–2625 (2017). https://doi.org/10.1137/1.9781611974782.172
3. De La Vega, W.F., Lueker, G.S.: Bin packing can be solved within $1+\varepsilon$ in linear time. Combinatorica **1**(4), 349–355 (1981). https://doi.org/10.1007/BF02579456
4. Bansal, N., Correa, J.R., Kenyon, C., Sviridenko, M.: Bin packing in multiple dimensions: inapproximability results and approximation schemes. Math. Oper. Res. **31**(1), 31–49 (2006). https://doi.org/10.1287/moor.1050.0168
5. Chung, F.R., Garey, M.R., Johnson, D.S.: On packing two-dimensional bins. SIAM J. Algebr. Discrete Methods **3**(1), 66–76 (1982). https://doi.org/10.1137/0603007
6. Caprara, A.: Packing d-dimensional bins in d stages. Math. Oper. Res. **33**, 203–215 (2008). https://doi.org/10.1287/moor.1070.0289
7. Bansal, N., Caprara, A., Sviridenko, M.: A new approximation method for set covering problems, with applications to multidimensional bin packing. SIAM J. Comput. **39**(4), 1256–1278 (2010). https://doi.org/10.1137/080736831
8. Jansen, K., Prädel, L.: New approximability results for two-dimensional bin packing. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 919–936 (2013). https://doi.org/10.1007/s00453-014-9943-z
9. Bansal, N., Khan, A.: Improved approximation algorithm for two-dimensional bin packing. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 13–25 (2014). https://doi.org/10.1137/1.9781611973402.2
10. Chlebík, M., Chlebíková, J.: Hardness of approximation for orthogonal rectangle packing and covering problems. J. Discrete Algorithms **7**(3), 291–305 (2009). https://doi.org/10.1016/j.jda.2009.02.002
11. Sweeney, P.E., Paternoster, E.R.: Cutting and packing problems: a categorized, application-orientated research bibliography. J. Oper. Res. Soc. **43**(7), 691–706 (1992). https://doi.org/10.1057/jors.1992.101
12. Gilmore, P.C., Gomory, R.E.: Multistage cutting stock problems of two and more dimensions. Oper. Res. **13**(1), 94–120 (1965). https://doi.org/10.1287/opre.13.1.94
13. Caprara, A., Lodi, A., Monaci, M.: Fast approximation schemes for two-stage, two-dimensional bin packing. Math. Oper. Res. **30**(1), 150–172 (2005). https://doi.org/10.1287/moor.1040.0112
14. Bansal, N., Lodi, A., Sviridenko, M.: A tale of two dimensional bin packing. In: Symposium on Foundations of Computer Science (FOCS), pp. 657–666. IEEE (2005). https://doi.org/10.1109/SFCS.2005.10
15. Coffman, E.G., Garey, M.R., Johnson, D.S., Tarjan, R.E.: Performance bounds for level-oriented two-dimensional packing algorithms. SIAM J. Comput. **9**, 808–826 (1980). https://doi.org/10.1137/0209062
16. Gálvez, W., Grandoni, F., Ameli, A.J., Jansen, K., Khan, A., Rau, M.: A tight $(3/2+\varepsilon)$ approximation for skewed strip packing. In: International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) (2020). https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2020.44
17. Christensen, H.I., Khan, A., Pokutta, S., Tetali, P.: Approximation and online algorithms for multidimensional bin packing: a survey. Comput. Sci. Rev. **24**, 63–79 (2017). https://doi.org/10.1016/j.cosrev.2016.12.001
18. Bansal, N., Eliáš, M., Khan, A.: Improved approximation for vector bin packing. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1561–1579. SIAM (2016). https://doi.org/10.1137/1.9781611974331.ch106
19. Sandeep, S.: Almost optimal inapproximability of multidimensional packing problems. In: Symposium on Foundations of Computer Science (FOCS), pp. 245–256 (2022). https://doi.org/10.1109/FOCS52979.2021.00033
20. Khan, A., Sharma, E., Sreenivas, K.V.N.: Geometry meets vectors: approximation algorithms for multidimensional packing. In: Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022), vol. 250, pp. 23:1–23:22 (2022). https://doi.org/10.4230/LIPIcs.FSTTCS.2022.23
21. Khan, A., Sharma, E., Sreenivas, K.V.N.: Approximation algorithms for generalized multidimensional knapsack (2021) arXiv:2102.05854 [cs.DS]

22. Steinberg, A.: A strip-packing algorithm with absolute performance bound 2. SIAM J. Comput. **26**(2), 401–409 (1997). https://doi.org/10.1137/S0097539793255801
23. Harren, R., Jansen, K., Prädel, L., Van Stee, R.: A $(5/3 + \varepsilon)$-approximation for strip packing. In: Workshop on Algorithms and Data Structures (WADS), pp. 475–487. Springer (2011). https://doi.org/10.1007/978-3-642-22300-6_40
24. Kenyon, C., Rémila, E.: Approximate strip packing. In: Symposium on Foundations of Computer Science (FOCS), pp. 31–36 (1996). https://doi.org/10.1109/SFCS.1996.548461
25. Jansen, K., van Stee, R.: On strip packing with rotations. In: ACM Symposium on Theory of Computing (STOC), pp. 755–761 (2005). https://doi.org/10.1145/1060590.1060702
26. Jansen, K., Zhang, G.: On rectangle packing: maximizing benefits. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), vol. 4, pp. 204–213 (2004)
27. Gálvez, W., Grandoni, F., Heydrich, S., Ingala, S., Khan, A., Wiese, A.: Approximating geometric knapsack via L-packings. In: Symposium on Foundations of Computer Science (FOCS), pp. 260–271. IEEE (2017). https://doi.org/10.1109/FOCS.2017.32
28. Gálvez, W., Grandoni, F., Khan, A., Ramirez-Romero, D., Wiese, A.: Improved approximation algorithms for 2-dimensional knapsack: packing into multiple L-shapes, spirals and more. In: International Symposium on Computational Geometry (SoCG), vol. 189, pp. 39:1–39:17 (2021). https://doi.org/10.4230/LIPIcs.SoCG.2021.39
29. Sharma, E.: Harmonic algorithms for packing d-dimensional cuboids into bins. In: Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021), vol. 213, pp. 32:1–32:22 (2021). https://doi.org/10.4230/LIPIcs.FSTTCS.2021.32
30. Seiden, S.S., Woeginger, G.J.: The two-dimensional cutting stock problem revisited. Math. Program. **102**(3), 519–530 (2005). https://doi.org/10.1007/s10107-004-0548-1
31. Khan, A., Maiti, A., Sharma, A., Wiese, A.: On guillotine separable packings for the two-dimensional geometric knapsack problem. In: International Symposium on Computational Geometry (SoCG), vol. 189, pp. 48:1–48:17 (2021). https://doi.org/10.4230/LIPIcs.SoCG.2021.48
32. Pach, J., Tardos, G.: Cutting glass. In: International Symposium on Computational Geometry (SoCG), pp. 360–369 (2000). https://doi.org/10.1145/336154.336223
33. Adamaszek, A., Har-Peled, S., Wiese, A.: Approximation schemes for independent set and sparse subsets of polygons. J. ACM **66**(4), 29:1–29:40 (2019). https://doi.org/10.1145/3326122
34. Khan, A., Pittu, M.R.: On guillotine separability of squares and rectangles. In: International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) (2020). https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2020.47
35. Abed, F., Chalermsook, P., Correa, J., Karrenbauer, A., Pérez-Lantero, P., Soto, J.A., Wiese, A.: On guillotine cutting sequences. In: International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), pp. 1–19 (2015). https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2015.1
36. Mitchell, J.S.B.: Approximating Maximum Independent Set for Rectangles in the Plane. In: Symposium on Foundations of Computer Science (FOCS), pp. 339–350 (2022). https://doi.org/10.1109/FOCS52979.2021.00042
37. Gálvez, W., Khan, A., Mari, M., Mömke, T., Pittu, M.R., Wiese, A.: A 3-approximation algorithm for maximum independent set of rectangles. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 894–905 (2022). https://doi.org/10.1137/1.9781611977073.38. SIAM
38. Fishkin, A.V., Gerber, O., Jansen, K.: On efficient weighted rectangle packing with large resources. In: International Symposium on Algorithms and Computation (ISAAC), pp. 1039–1050. Springer (2005). https://doi.org/10.1007/11602613_103
39. Alamdari, S., Biedl, T., Chan, T.M., Grant, E., Jampani, K.R., Keshav, S., Lubiw, A., Pathak, V.: Smart-grid electricity allocation via strip packing with slicing. In: Workshop on Algorithms and Data Structures (WADS), pp. 25–36. Springer (2013). https://doi.org/10.1007/978-3-642-40104-6_3
40. Deppert, M.A., Jansen, K., Khan, A., Rau, M., Tutas, M.: Peak Demand Minimization via Sliced Strip Packing. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021), vol. 207, pp. 21:1–21:24 (2021). https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2021.21
41. Gálvez, W., Grandoni, F., Ameli, A.J., Khodamoradi, K.: Approximation Algorithms for Demand Strip Packing. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021), vol. 207, pp. 20:1–20:24 (2021). https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2021.20

42. Lau, L.C., Ravi, R., Singh, M.: Iterative Methods in Combinatorial Optimization, vol. 46. Cambridge University Press, Cambridge (2011)
43. Prädel, L.D.: Approximation algorithms for geometric packing problems. PhD thesis, Kiel University (2012). https://macau.uni-kiel.de/servlets/MCRFileNodeServlet/dissertation_derivate_00004634/dissertation-praedel.pdf?AC=N
44. Vanderbei, R.J.: Linear Programming: Foundations and Extensions. International Series in Operations Research & Management Science. Springer, Berlin (2013). https://doi.org/10.1007/978-1-4614-7630-6
45. Bland, R.G.: New finite pivoting rules for the simplex method. Math. Oper. Res. **2**(2), 103–107 (1977). https://doi.org/10.1287/moor.2.2.103
46. Johnson, D.S.: Near-optimal bin packing algorithms. PhD thesis, Massachusetts Institute of Technology, USA (1973)
47. Bansal, N., Caprara, A., Jansen, K., Prädel, L., Sviridenko, M.: A structural lemma in 2-dimensional packing, and its implications on approximability. In: International Symposium on Algorithms and Computation (ISAAC), pp. 77–86. Springer (2009). https://doi.org/10.1007/978-3-642-10631-6_10