Approximating the Geometric Knapsack Problem in Near-Linear Time and Dynamically

Moritz Buchem
Technische Universität München, Germany

Paul Deuker
Technische Universität München, Germany

Andreas Wiese
Technische Universität München, Germany

Abstract

One important goal in algorithm design is determining the best running time for solving a problem (approximately). For some problems, we know the optimal running time, assuming certain conditional lower bounds. In this paper, we study the d-dimensional geometric knapsack problem in which we are far from this level of understanding. We are given a set of weighted d-dimensional geometric items like squares, rectangles, or hypercubes and a knapsack which is a square or a (hyper-)cube. Our goal is to select a subset of the given items that fit non-overlappingly inside the knapsack, maximizing the total profit of the packed items. We make a significant step towards determining the best running time for solving these problems approximately by presenting approximation algorithms whose running times are near-linear, i.e., $O(n \cdot \text{poly}(\log n))$, for any constant d and any parameter $\epsilon > 0$ (the exponent of $\log n$ depends on d and $1/\epsilon$).

In the case of (hyper)-cubes, we present a $(1+\epsilon)$-approximation algorithm. This improves drastically upon the currently best known algorithm which is a $(1+\epsilon)$-approximation algorithm with a running time of $n^{O_{\epsilon,d}(1)}$ where the exponent of n depends exponentially on $1/\epsilon$ and d. In particular, our algorithm is an efficient polynomial time approximation scheme (EPTAS). Moreover, we present a $(2+\epsilon)$-approximation algorithm for rectangles in the setting without rotations and a $(\frac{17}{9}+\epsilon) \approx 1.89$-approximation algorithm if we allow rotations by 90 degrees. The best known polynomial time algorithms for this setting have approximation ratios of $\frac{17}{9}+\epsilon$ and $1.5+\epsilon$, respectively, and running times in which the exponent of n depends exponentially on $1/\epsilon$. In addition, we give dynamic algorithms with polylogarithmic query and update times, having the same approximation guarantees as our other algorithms above.

Key to our results is a new family of structured packings which we call *easily guessable packings*. They are flexible enough to guarantee the existence of profitable solutions while providing enough structure so that we can compute these solutions very quickly.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

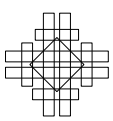
Keywords and phrases Geometric packing, approximation algorithms, dynamic algorithms

Digital Object Identifier 10.4230/LIPIcs.SoCG.2024.26

Related Version *Full Version*: https://arxiv.org/abs/2403.00536 [7]

1 Introduction

Knapsack is a fundamental problem in combinatorial optimization. We are given a knapsack with a specified capacity W and a set of n items, each of them characterized by its size s_i and its profit p_i. The goal is to compute a set of items that fits into the knapsack, maximizing the total profit. Knapsack is very well understood: there is an FPTAS for the problem with a running time of only $\tilde{O}(n + (1/\epsilon)^{2.2})$ [10] with an asymptotically almost matching conditional lower bound of $(n + 1/\epsilon)^{2-o(1)}$ [9, 25]. Even more, there are dynamic



algorithms for Knapsack that can maintain $(1+\epsilon)$-approximate solutions in polylogarithmic update time whenever an item is inserted to or removed from the input [11, 17]. It is an important goal in algorithm design to determine the best possible running time to solve (or approximate) a problem. Besides Knapsack, there are also many other problems for which we have (almost) matching upper and lower bounds, e.g., computing the Fréchet distance [4], Least Common Subsequence [1, 5], Negative Triangles [29], or Graph Diameter [28].

A natural generalization of Knapsack is the *d-dimensional geometric knapsack* problem in which the items are geometric objects like squares, rectangles, or hypercubes. Like in Knapsack, we want to select a subset of the given items, but now we also require that they are placed non-overlappingly inside the knapsack, which we assume to be a square or a (hyper-)cube. The problem is motivated by many practical applications like placing advertisements on a website, cutting pieces out of raw material like wood or metal, or loading cargo into a ship or a truck. Formally, we assume that we are given an integer N such that our knapsack is an axis-parallel square (if $d = 2$) or a (hyper-)cube (if $d \geq 3$) where each edge has length N. Also, we are given a set of items $\mathcal{I}$ where each item $i \in \mathcal{I}$ is a d-dimensional (hyper-)cube or a d-dimensional (hyper-)cuboid with axis-parallel edges and a given profit.

Unfortunately, our understanding of the *d-dimensional geometric knapsack* problem falls short in comparison to our understanding of Knapsack. There is a polynomial time $(1+\epsilon)$-approximation algorithm for each $\epsilon > 0$ if all input items are (hyper-)cubes, due to Jansen, Khan, Lira, and Sreenivas [22]. In the running time of this algorithm, the exponent of n depends exponentially on d and $1/\epsilon$. However, there is no (conditional) lower bound known that justifies this. From all we know, it might still be possible to obtain a better running time of the form $f(\epsilon, d)n^{O(1)}$, for which the exponent of n depends neither on ϵ nor on d, but it is only a small constant like 2 or even 1. Note that there are problems for which we know conditional running time lower bounds that rule this out, e.g., lower bounds of $\Omega(n^2)$ or $\Omega(n^3)$, based on assumptions like the (Strong) Exponential Time Hypothesis or the 3-SUM conjecture (see, e.g., [1, 2, 6, 4, 5] and references therein). For example, for the Graph Diameter problem there is a lower bound of $\Omega(m^2)$, with m being the number of edges of the given graph, for computing a better approximation ratio than 3/2 [28]. However, no such lower bounds are known for d-dimensional geometric knapsack.

For the special case of squares, i.e., $d = 2$, there is a $(1+\epsilon)$-approximation known with a running time of the form $f(\epsilon)n^{O(1)}$ due to Heydrich and Wiese [18]. However, even in this result the running time is much slower than linear time since the algorithm uses an initial guessing step with $\Omega(n)$ options and for each of these option solves several linear programs of size $\Omega(n)$ each. Furthermore, there is no dynamic algorithm known for d-dimensional geometric knapsack, not even for the special case of squares (i.e., if $d = 2$).

If we allow more general shapes than squares, cubes, and hypercubes, we understand the problem even less. For two-dimensional axis-parallel rectangles, the best known polynomial time approximation algorithm is due to Gálvez, Grandoni, Ingala, Heydrich, Khan, and Wiese [12], having an approximation ratio of $1.89 + \epsilon$. If it is allowed to rotate rectangles by 90 degrees, then a $(1.5+\epsilon)$-approximation algorithm is known [12]. The problem is not known to be APX-hard, so it may even admit a PTAS. Also in the mentioned results, the exponent of n in the running time depends exponentially on $1/\epsilon$. However, we do not know any lower bound of the needed running time to solve the problem. Thus, it might well be possible that we can achieve these or similar results in a running time that is much faster, e.g., $O(n \cdot \text{poly}(\log n))$. Also, it is open whether a dynamic algorithm exists for the problem.

1.1 Our contribution

Our first result is a $(1+\epsilon)$-approximation algorithm for the d-dimensional geometric knapsack problem for squares, cubes, and hypercubes with a running time that is near-linear, i.e., $O(n \cdot \text{poly}(\log n))$ for any constant d and $\epsilon > 0$, where the exponent of $\log n$ depends (exponentially) on d as well as $1/\epsilon$. In particular, this drastically improves the exponent of n in the running time in [22] from a value that is exponential in the dimension d and $1/\epsilon$ to only 1, which is in particular completely independent of d and $1/\epsilon$. This even implies that our algorithm is an *efficient* polynomial time approximation scheme (EPTAS)[1]. Thus, up to polylogarithmic factors, we obtain the fastest possible running time of a PTAS for any fixed d and ϵ. Note that, for constant d, this yields a distinction to problems for which there are lower bounds of $\Omega(n^2)$ or $\Omega(n^3)$ based on (S)ETH or other hypotheses, e.g., [1, 2, 6, 4, 5, 28].

For the case of rectangles, we present a $(2+\epsilon)$-approximation algorithm with a running time of $O(n \cdot \text{poly}(\log n))$ for any constant $\epsilon > 0$. If it is allowed to rotate the rectangles by 90 degrees, we obtain an approximation ratio of $\frac{17}{9} + \epsilon$ with the same running time bound. Thus, our algorithms are much faster than the best known polynomial time algorithms for the problem [12]; in their running times, the exponent of n depends exponentially on $1/\epsilon$ while our exponent is only 1. Although we need much less running time, our approximation factors are not much higher than their ratios of $1.89 + \epsilon$ and $1.5 + \epsilon$, respectively.

Moreover, we present the first dynamic algorithms for d-dimensional geometric knapsack with hypercubes and rectangles with and without rotations by 90 degrees. These algorithms maintain solutions with the same approximation guarantees as stated above, with polylogarithmic worst-case query and update times. In comparison, note that there are problems for which there are polynomial conditional lower bounds for the update and query time for dynamic algorithms [16]. We remark that our algorithms maintain *implicit* solutions in the sense that after each update operation, the answers to all query operations refer to the same fixed approximate solution. Note that after adding or removing a single item (e.g., a very large but very profitable item), it can be necessary to change $\Omega(n)$ items in the current solution in order to maintain a bounded approximation guarantee. Therefore, it is impossible to maintain explicit solutions with our update and query times.

1.2 Techniques

The known algorithms for the d-dimensional geometric knapsack problem for squares, cubes, hypercubes, and rectangles are based on the existence of structured packings into $O_{\epsilon,d}(1)$ boxes (i.e., a constant number of boxes for each fixed ϵ and d). In these algorithms, one first guesses these boxes (i.e., enumerates all possibilities) which already yields a running time bound of $n^{O_{\epsilon,d}(1)}$. It is not clear how to make use of these structured packings without guessing the boxes first. Instead, we use a different type of structured packings which we call *easily guessable packings*. They are

- **(i)** flexible enough so that they allow for very profitable solutions, and
- **(ii)** structured enough so that we can compute these solutions very fast.

In these packings, each box is specified by some parameters (e.g. height and width) and we can guess *all but at most one* parameter for each box in time $O(\text{poly}(\log n))$. In contrast, in the previous results, $n^{\Omega(1)}$ time is needed already for one single parameter. However, for each box, there may still be one parameter whose value we have not guessed yet. To determine them, we use an important property of our easily guessable packings. There is a partition of the input items and a partition of the boxes for which we have not yet guessed all parameters.

[1] Note that there exists a function f such that for any n and k we have that $(\log n)^k \leq f(k) \cdot n^{O(1)}$.

For each resulting subset of items, all items of this set can be assigned only to boxes in one specific set of boxes in the partition. Moreover, all boxes in the latter set have a certain (identical) value for the remaining (not yet guessed) parameter. This allows us to adapt the indirect guessing framework from [18] to guess the remaining parameter approximately for each box step by step, while losing only a factor of $1+\epsilon$ in our profit, compared to guessing it exactly in time $n^{\Omega(1)}$.

We refer to the full version of this paper [7] for missing proofs and details.

1.3 Other related work

Prior to the results for two-dimensional geometric knapsack for rectangles mentioned above, a polynomial time $(2+\epsilon)$-approximation algorithm was presented by Jansen and Zhang [19] in which the exponent of n in the running time is exponential in $1/\epsilon$. For the special case of unweighted rectangles, the same authors gave a faster $(2+\epsilon)$-approximation algorithm with a running time of $O(n^{1/\epsilon+1})$ [23]. If one allows pseudo-polynomial running time instead of polynomial running time, there is also a $(4/3+\epsilon)$-approximation algorithm in the setting of weighted rectangles known due to Gálvez, Grandoni, Khan, Ramírez-Romero, and Wiese [13], having a running time of $(nN)^{O_\epsilon(1)}$. Also here, the exponent of nN is exponential in $1/\epsilon$. In addition, there is a $(1+\epsilon)$-approximation algorithm by Adamaszek and Wiese [3] with quasi-polynomial running time for any constant $\epsilon > 0$, assuming quasi-polynomially bounded input data. If the input objects are triangles that can be freely rotated, there is a polynomial time $O(1)$-approximation algorithm due to Merino and Wiese [27].

Another way to generalize Knapsack is to allow several knapsacks into which the items can be packed, possibly with different capacities. This generalization still admits $(1+\epsilon)$-approximation algorithms with a running time of $n^{O_\epsilon(1)}$ [8, 24], and even with a running time of the form $f(\epsilon)n^{O(1)}$ for some function f [20, 21]. Furthermore, there is a dynamic algorithm known for the problem with polylogarithmic update time which also achieves an approximation ratio of $1+\epsilon$ [11].

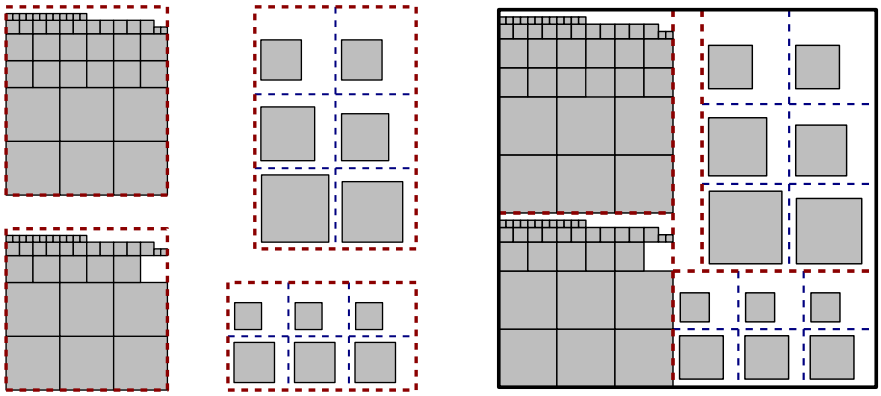
2 Algorithms for d-dimensional hypercubes

In this section we present our PTAS and dynamic algorithm for the geometric knapsack problem with d-dimensional hypercubes. Let $\epsilon > 0$ and assume w.l.o.g. that $1/\epsilon \in \mathbb{N}$ and $\epsilon < 1/2^{d+2}$; we assume that $d \in \mathbb{N}$ is a constant. We are given a set of n items $\mathcal{I}$ where each item $i \in \mathcal{I}$ is a hypercube with side length $s_i \in \mathbb{N}$ in each dimension and profit $p_i \in \mathbb{N}$, and an integer N such that the knapsack has side length N (in each dimension). In the following, we present a simplified version of our algorithm with a running time of $O(n \cdot \mathrm{poly}(\log N))$, which we can improve to $O(n \cdot \mathrm{poly}(\log n))$ using more involved technical ideas, see [7].

In our algorithms, we use a special data structure to store our items $\mathcal{I}$. This data structure will allow us to quickly update the input and obtain important information about the items in $\mathcal{I}$. It is defined in the following lemma, which can be proven using standard data structures for range counting/reporting for points in two dimensions (corresponding to side length and profit of each item) [26].

▶ **Lemma 1.** *There is a data structure for the items $\mathcal{I}$ that allows the following operations:*

- *Insertion and deletion of an item in time $O(\log^2 n)$.*
- *Given four values $a, b, c, d \in \mathbb{N}$, return the cardinality of the set $\mathcal{I}(a,b,c,d) := \{i \in I : a \le s_i \le b, c \le p_i \le d\}$ in time $O(\log n)$.*
- *Given four values $a, b, c, d \in \mathbb{N}$, return the set $\mathcal{I}(a,b,c,d) := \{i \in I : a \le s_i \le b, c \le p_i \le d\}$ time $O(\log n + |\mathcal{I}(a,b,c,d)|)$.*



(a) Packing of items into single boxes. **(b)** Packing of boxes into knapsack.

Figure 1 Visualization of packing using boxes.

In the following, we will refer to this data structure as our *item data structure*. Observe that we can insert all given items $\mathcal{I}$ into it in time $O(n \log^2 n)$. Additionally, we use balanced binary search trees (see e.g. [14]) to store the set of item side lengths and profits; an item can be inserted, deleted, and queried in time $O(\log n)$ in each of these search trees.

2.1 Easily guessable packing of hypercubes

Our algorithm is based on a structured packing into $O_{\epsilon,d}(1)$ boxes, i.e., d-dimensional hypercuboids, such that the profit of the packed items is at least $(1 - O(\epsilon))\mathrm{OPT}$, where OPT denotes the optimal solution of the given instance. These boxes are packed non-overlappingly inside the knapsack and each item is packed into one of these boxes (see Figure 1). Intuitively, in our algorithm we will guess the sizes of these boxes and compute an assignment of items into the boxes. Our boxes are designed such that we can make these guesses quickly and such that it is easy to place all items that were assigned to each box.

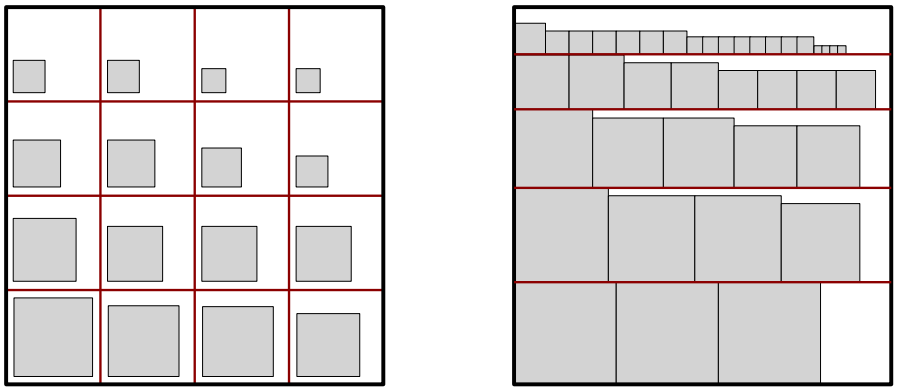
In [22], a structured packing was presented that lead to a PTAS with a running time of $n^{O_{\epsilon,d}(1)}$ for our problem. They use two specific types of boxes. In the formal definition of those, we need the volume and surface of a d-dimensional box B which we define as $\mathrm{VOL}_d(B) := \prod_{d'=1}^{d} \ell_{d'}(B)$ and $\mathrm{SURF}_d(B) := 2\sum_{d'=1}^{d} \mathrm{VOL}_d(B)/\ell_{d'}(B)$, respectively.

▶ **Definition 2** ($\mathcal{V}$-boxes and $\mathcal{N}$-boxes [22]). *Let B be a d-dimensional hypercuboid, $\mathcal{I}$ be a set of items packed in it, and let $\hat{s}$ be an upper bound on the side lengths of the items in $\mathcal{I}$. We say that B is*

- *a $\mathcal{V}$-box if its side lengths can be written as $n_1\hat{s}, n_2\hat{s}, \ldots, n_d\hat{s}$ where $n_1, n_2, \ldots, n_d \in \mathbb{N}_+$ and the volume of the packed items is at most $\mathrm{VOL}_d(B) - \hat{s}\frac{\mathrm{SURF}_d(B)}{2}$,*
- *an $\mathcal{N}$-box if its side lengths can be written as $n_1\hat{s}, n_2\hat{s}, \ldots, n_d\hat{s}$ where $n_1, n_2, \ldots, n_d \in \mathbb{N}_+$ and the number of packed items is at most $\prod_{i=1}^{d} n_i$.*

These boxes and the structured packing using them, however, is not enough for our purposes since each parameter must be guessed. It is unclear how to do this faster than in time $n^{\Omega(1)}$, already for one single parameter. Therefore, we refine the packing presented in [22] by adapting the type of boxes to include additional technical properties.

Our first type of boxes are $\mathcal{N}^*$-boxes which are a refinement of the aforementioned $\mathcal{N}$-boxes. For each $\mathcal{N}^*$-box B we specify two additional parameters $s_{\min}(B)$ and $s_{\max}(B)$ and require for each item $\in \mathcal{I}$ packed in B that $s_{\min}(B) \leq s_i \leq s_{\max}(B)$. These values $s_{\min}(B)$



(a) Sorted packing of an $\mathcal{N}^*$-box: each item is packed into a single grid cell.

(b) NFDH packing of an $\mathcal{S}$-box: items are packed greedily into shelves.

Figure 2 Visualization of an $\mathcal{N}^*$- and an $\mathcal{S}$-box for $d = 2$.

and $s_{\max}(B)$ will help us in our computation later. Intuitively, the box B is partitioned into a d-dimensional grid with spacing $s_{\max}$, such that each item in B is placed inside one of these grid cells (similar as the $\mathcal{N}^*$-boxes), see Figure 2a.

▶ **Definition 3** ($\mathcal{N}^*$-box). *Let B be a d-dimensional hypercuboid with given values $s_{\min}(B) \geq 0$ and $s_{\max}(B) \geq 0$ and suppose we are given a packing of items $\mathcal{I}' \subseteq \mathcal{I}$ inside B. We say that B is an $\mathcal{N}^*$-box if $s_{\min}(B) \leq \min_{i\in\mathcal{I}'} s_i$ and $s_{\max}(B) \geq \max_{i\in\mathcal{I}'} s_i$ and if its side lengths can be written as $n_1(B)s_{\max}(B), \ldots, n_d(B)s_{\max}(B)$ with $n_1(B), \ldots, n_d(B) \in \mathbb{N}$ such that $|\mathcal{I}'| \leq \prod_{i=1}^{d} n_i(B)$.*

Our second type of boxes are $\mathcal{S}$-boxes. Intuitively, an $\mathcal{S}$-box B contains only items that are very small in each dimension compared to B. For technical reasons, we require that the packed items do not use the full volume of B, even if we increase each item size to the next larger power of $1 + \epsilon$. This will allow us to compute our solution efficiently since we may round up item sizes and profits to powers of $1 + \epsilon$.

Throughout this paper, for any $x > 0$ we define $\lceil x \rceil_{1+\epsilon}$ to be the smallest power of $1 + \epsilon$ that is larger than x rounded down. Similarly, we define $\lfloor x \rfloor_{1+\epsilon}$ to be the largest power of $1 + \epsilon$ that is smaller than x rounded up.

▶ **Definition 4** ($\mathcal{S}$-boxes). *Let B be a d-dimensional hypercuboid with side length $\ell_{d'}(B) \in \mathbb{N}_0$ for each $d' \in [d]$ and suppose we are given a packing of items $\mathcal{I}' \subseteq \mathcal{I}$ inside B. We say that B is a $\mathcal{S}$-box if for each item $i \in \mathcal{I}'$ we have $s_i \leq \epsilon \min_{d'\in[d]} \ell_{d'}(B)$ and additionally $\sum_{i\in\mathcal{I}'} \lceil s_i \rceil_{1+\epsilon}^{d} \leq (1 - 2d \cdot \epsilon)\mathrm{VOL}_d(B)$.*

For an $\mathcal{S}$-box B, we can show that all items in $\mathcal{I}'$ can be packed inside B using the Next-Fit-Decreasing-Height (NFDH) algorithm. NFDH is a greedy algorithm that orders the items non-increasingly by their sizes and then packs them greedily in this order into shelves within the box B (see Figure 2b).

▶ **Lemma 5** (implied by Lemma 4 in [15]). *Let B be a box and let $\mathcal{I}'$ be a set of items such that $s_i \leq 2\epsilon\ell_{\min}(B)$ for each $i \in \mathcal{I}'$ and $\sum_{i\in\mathcal{I}'} \lceil s_i \rceil_{1+\epsilon}^{d} \leq \mathrm{VOL}_d(B) - 2(d-1) \cdot \epsilon\mathrm{VOL}_d(B)$. Then, NFDH finds a packing of $\mathcal{I}'$ into B.*

We now prove that there is always a $(1 + O(\epsilon))$-approximate *easily guessable packing* using $O_{\epsilon,d}(1)$ boxes, each of them being an $\mathcal{N}^*$- or an $\mathcal{S}$-box. In order to do so we refine the structured packing in [22] which uses $\mathcal{N}$- and $\mathcal{V}$-boxes. The key additional property is that

there are $O_\epsilon(1)$ values $k_1, k_2, \ldots, k_r$ such that each $\mathcal{N}^*$-box B contains only items whose respective sizes are within the interval $(k_{j-1}, k_j]$ for some j. In particular, we have that $s_{\min}(B) = k_{j-1}$ and $s_{\max}(B) = k_j$ and the values $k_1, k_2, \ldots, k_r$ yield a partition of the items in $\mathcal{I}$ of the form $(k_{j-1}, k_j]$. This will allows us to apply an indirect guessing framework later to guess the values $k_1, k_2, \ldots, k_r$ quickly. All remaining parameters of the boxes can be easily guessed in time $O(\log N)$ each (and there are only $O_{\epsilon,d}(1)$ parameters in total).

▶ **Lemma 6** (Near-optimal packing of hypercubes). *For any instance $\mathcal{I}$ of the d-dimensional hypercube knapsack problem and any $\epsilon < 1/2^{d+2}$, there exists a packing with the following properties:*

- **i)** *It consists of $\mathcal{N}^*$- and $\mathcal{S}$-boxes whose total number is bounded by a value $C_{\text{boxes}}(\epsilon, d)$ depending only on ϵ and d.*
- **ii)** *There exist values $k_1, k_2, \ldots, k_r \in \mathbb{Z}_{\geq 0}$ with $r \in O_{\epsilon,d}(1)$ such that if B is an $\mathcal{N}^*$-box there exists a value $j_B \in \{1, 2, \ldots, r\}$ such that $s_{\max}(B) = k_{j_B}$ and $s_{\min}(B) = k_{j_B - 1}$ with $k_0 := 0$.*
- **iii)** *For each $\mathcal{S}$-box B and each $d' \in [d]$ we have that $\ell_{d'}(B) = \lfloor x \rfloor_{1+\epsilon}$ for some $x \in [N]$.*
- **iv)** *For each $\mathcal{N}^*$-box B and each $d' \in [d]$ we have that $n_{d'}(B) = \lfloor n' \rfloor_{1+\epsilon}$ for some $n' \in [n]$ if $n_{d'}(B) > 1/\epsilon$.*
- **v)** *The total profit of the packing is at least $(1 - 2^{O(d)}\epsilon)\text{OPT}$.*

Proof sketch. We start with the structured packing due to Jansen, Khan, Lira and Sreenivas [22] which consists of $O_{\epsilon,d}(1)$ many $\mathcal{N}$- and $\mathcal{V}$-boxes and items with a total profit of at least $(1 - 2^{d+2}\epsilon)\text{OPT}$. We modify this packing as follows. First, we consider the $\mathcal{V}$-boxes and split each of them into at most $O_{\epsilon,d}(1)$ many $\mathcal{N}^*$- and $\mathcal{S}$-boxes. For each resulting $\mathcal{N}^*$-box B, we define $s_{\max}(B)$ to be the maximum size of an item packed into B. This yields a packing satisfying property i). In order to satisfy property ii), we first introduce a value k_j for each distinct value $s_{\max}(B)$ of the $\mathcal{N}^*$-boxes constructed so far. We order the resulting values $k_1, ..., k_r$ increasingly. Then, we split the $\mathcal{N}^*$-boxes into smaller boxes such that for each resulting box B, the range of item sizes of B is contained in $(k_{j-1}, k_j]$ for some j; hence, we define $s_{\min}(B) := k_{j-1}$. We ensure that the number of boxes increases by at most a factor of $O_{\epsilon,d}(1)$ in this step. Next, we modify the packing to satisfy properties iii) and iv). For each $\mathcal{N}^*$-box B and each dimension $d' \in [d]$, we round $n_{d'}(B)$ down to $\lfloor n_{d'}(B) \rfloor_{1+\epsilon}$. This decreases the maximum number of items we can pack into B by at most a factor of $(1+\epsilon)^d = 1 + O(\epsilon)$; hence, also our profit reduces by at most this factor. Finally, for each $\mathcal{S}$-box B we round down each side length $\ell_{d'}(B)$ to $\lfloor \ell_{d'}(B) \rfloor_{1+\epsilon}$. Since all items in B are small compared to each side length of B, our profit reduces by at most a factor of $1 + \epsilon$. ◀

2.2 Computing a packing

In this section we discuss how to compute our packing. Intuitively, we try to guess the packing due to Lemma 6. We assume that all input items $\mathcal{I}$ are stored in our item data structure (see to Lemma 1) and we discard all items with profit less than $\epsilon \frac{p_{\max}}{n}$ where $p_{\max} := \max_{i \in \mathcal{I}} p_i$. The total profit of the discarded items is at most $n \cdot \epsilon \frac{p_{\max}}{n} \leq \epsilon\text{OPT}$.

Let $\mathcal{B}$ denote the set of boxes due to Lemma 6. In our algorithm, we first guess all parameters of the boxes in $\mathcal{B}$ apart from the values $k_1, \ldots, k_r$. We then modify the indirect guessing framework introduced by Heydrich and Wiese [18] to approximately compute the values $k_1, \ldots, k_r$. Our computed values might be imprecise in the sense that they yield a solution whose profit could be by a factor $1 + \epsilon$ lower than the solution due to Lemma 6. However, our resulting boxes are guaranteed to fit inside of the knapsack.

2.2.1 Guessing basic quantities

We start by guessing the number of $\mathcal{N}^*$- and $\mathcal{S}$-boxes, respectively. Then, for each $\mathcal{N}^*$-box $B \in \mathcal{B}$ we guess

- the value j_B indicating that $s_{\max}(B) = k_{j_B}$ and $s_{\min}(B) = k_{j_B-1}$; note that for j_B there are only $O_{\epsilon,d}(1)$ possibilities,
- the value $n_{d'}(B)$ for each dimension $d' \in [d]$; for each value $n_{d'}(B)$ there are only $O(\log n)$ possibilities.

For each $\mathcal{S}$-box $B \in \mathcal{B}$, we guess $\ell_{d'}(B)$ for each $d' \in [d]$. Note that also here, for each of these values there are only $O(\log N)$ possibilities. We denote by the *basic quantities* all these guessed values for all boxes in $\mathcal{B}$.

▶ **Lemma 7.** *In time* $(\log N)^{O_{\epsilon,d}(1)}$ *we can guess all basic quantities.*

We do not know the values $k_1, k_2, \ldots, k_r$. However, recall that they yield a partition of $\mathcal{I}$ with a set $\mathcal{I}_j := \{i \in \mathcal{I} : s_i \in (k_{j-1}, k_j]\}$ for each $j \in [r]$. We next guess additional quantities which provide us with helpful information for the next step of our algorithm, the indirect guessing framework. First, we guess approximately the profit that each set $\mathcal{I}_j$ contributes to OPT. Formally, for each $j \in [r]$ we guess $\hat{p}(j) := \lfloor p(\mathcal{I}_j \cap \mathrm{OPT}) \rfloor_{1+\epsilon}$ if $p(\mathcal{I}_j \cap \mathrm{OPT}) \geq \frac{\epsilon}{r}\mathrm{OPT}$ and $\hat{p}(j) := 0$ otherwise. Since $\mathrm{OPT} \in [p_{\max}, n \cdot p_{\max})$, we have that $\hat{p}_j \in \{0\} \cup [\frac{\epsilon}{r}p_{\max}, n \cdot p_{\max})$; hence, there are only $O(\log N)$ possibilities for $\hat{p}_j$.

▶ **Lemma 8.** *The value* $\hat{p}_j$ *for each* $j \in [r]$ *can be guessed in time* $(\log n)^{O_{\epsilon,d}(1)}$ *and they satisfy* $\sum_{j=1}^{r} \hat{p}(j) \geq (1 - O(\epsilon))\mathrm{OPT}$.

Observe that each $\mathcal{N}^*$-box $B \in \mathcal{B}$ can contain only items from $\mathcal{I}_{j_B}$. However, each $\mathcal{S}$-box $B \in \mathcal{B}$ might contain items from more than one set $\mathcal{I}_j$. For each $\mathcal{S}$-box $B \in \mathcal{B}$ and each set $\mathcal{I}_j$, we guess approximately the fraction of the volume B that is occupied by items from $\mathcal{I}_j$. Formally, for each such pair we define the value $a_{B,j} := \frac{\sum_{i \in \mathcal{I}(B) \cap \mathcal{I}_j} \lceil s_i \rceil_{1+\epsilon}^d}{\mathrm{VOL}_d(B)}$ and guess the value $\hat{a}_{B,j} := \left\lceil \frac{a_{B,j}}{\epsilon/r} \right\rceil \epsilon/r$, i.e., the value $a_{B,j}$ rounded up to the next larger integral multiple of ϵ/r. Note that for each value $\hat{a}_{B,j}$ there are only $O_{\epsilon,d}(1)$ possibilities, and that there are only $O_{\epsilon,d}(1)$ such values that we need to guess.

▶ **Lemma 9.** *For each* $\mathcal{S}$*-box* $B \in \mathcal{B}$ *and each* $j \in [r]$ *the value* $\hat{a}_{B,j}$ *can be guessed in time* $(\log n)^{O_{\epsilon,d}(1)}$. *Moreover, for each* $\mathcal{S}$*-box* $B \in \mathcal{B}$ *we have that* $\sum_{j=1}^{r} \hat{a}_{B,j}\mathrm{VOL}_d(B) \leq (1 - 2d \cdot \epsilon)\mathrm{VOL}_d(B) + \epsilon\mathrm{VOL}_d(B)$.

2.2.2 Indirect guessing

The next step of our algorithm is to determine the values $k_1, k_2, \ldots, k_r$. Unfortunately, we cannot guess them directly in polylogarithmic time, since there are N options for each of them. In contrast to the other guessed quantities, it is not sufficient to allow only powers of $1 + \epsilon$ for each of them. If we choose a value for some k_j that is only a little bit too large, then our boxes might not fit into the knapsack anymore, since for each $\mathcal{N}^*$-box B we have that $s_{\max}(B) = k_{j_B}$ and the side length of B in each dimension d' equals $n_{d'}(B)s_{\max}(B)$. On the other hand, if we define some value k_j only a little bit too small, then we might not be able to assign items with enough profit to B, e.g., if B contains only one item which does not fit anymore if we choose k_j too small. Hence, we need a different approach to determine the values $k_1, k_2, \ldots, k_r$. To this end, we modify the *indirect guessing framework* introduced in [18] to fit our purposes.

The main idea is to compute values $\tilde{k}_0, \tilde{k}_1, \tilde{k}_2, \dots, \tilde{k}_r$ that we use instead of the values $k_0, k_1, k_2, \dots, k_r$. They yield a partition of $\mathcal{I}$ into sets $\tilde{\mathcal{I}}_j := \{i \in \mathcal{I} : s_i \in (\tilde{k}_{j-1}, \tilde{k}_j]\}$. Intuitively, for each j we want to pack items from $\tilde{\mathcal{I}}_j$ into the space that is used by items in $\mathcal{I}_j$ in the packing from Lemma 6. We will choose the values $\tilde{k}_1, \tilde{k}_2, \dots, \tilde{k}_r$ such that in this way, we obtain almost the same profit as the packing of Lemma 6. Furthermore, we ensure that $\tilde{k}_j \leq k_j$ for each $j \in [r]$. This implies that the side lengths of the resulting $\mathcal{N}^*$-boxes are at most the side lengths of the $\mathcal{N}^*$-boxes due to Lemma 6 and, therefore, the guessed $\mathcal{N}^*$-boxes can be feasibly packed into the knapsack.

Formally, we perform r iterations, one for each value k_j. We define $\tilde{k}_0 := 0$. Suppose inductively that we have determined ℓ values $\tilde{k}_1, \tilde{k}_2, \dots, \tilde{k}_\ell$ already for some $\ell \in \{0, 1, ..., r-1\}$ such that $\tilde{k}_\ell \leq k_\ell$. We want to compute $\tilde{k}_{\ell+1}$ such that $\tilde{k}_{\ell+1} \leq k_{\ell+1}$. To this end, we do a binary search on the set $S := \{s_i : i \in \mathcal{I} \wedge \tilde{k}_\ell < s_i\}$, using our item data structure. Our first candidate value is the median of S which we can find in time $O(\log n)$ via our binary search tree for the item sizes. For each candidate value $s \in S$, we estimate up to a factor of $1+\epsilon$ the possible profit due to items in $\tilde{\mathcal{I}}_{\ell+1}$ if we define $\tilde{k}_{\ell+1} := s$. We will describe later how we compute such an estimation. The objective is to find the smallest value $s \in S$ such that the estimated profit is at least $(1+\epsilon)^{-1}\hat{p}(j)$. Hence, if for a specific guess s our obtained profit due to s is at least $(1+\epsilon)^{-1}\hat{p}(j)$, we restrict our set S to $\{s_i : i \in \mathcal{I} \wedge \tilde{k}_\ell \leq s_i \leq s\}$ and continue with the next iteration of the binary search. If, however, the estimated profit due to s is strictly less than $(1+\epsilon)^{-1}\hat{p}(j)$, this implies that $\tilde{k}_{\ell+1} > s$ since otherwise the set of items $\tilde{\mathcal{I}}_{\ell+1}$ is a subset of the items considered for guess s and cannot yield a larger profit. We restrict our set S to $\{s_i : i \in \mathcal{I} \wedge s_i > s\}$ and continue with the binary search until $|S| = 1$.

We denote by $\mathcal{B}_{\mathcal{N}^*}$ and $\mathcal{B}_{\mathcal{S}}$ the set of $\mathcal{N}^*$- and $\mathcal{S}$-boxes in $\mathcal{B}$ due to Lemma 6, respectively. Additionally, we denote by $\mathcal{B}_{\mathcal{N}^*}(\ell+1)$ the set of $\mathcal{N}^*$-boxes B with $j_B = \ell+1$ and by $\mathcal{B}_{\mathcal{S}}(\ell+1)$ the set of $\mathcal{S}$-boxes B with $\hat{a}_{B,\ell+1} > 0$. Note that those are the only boxes that are relevant for the current iteration in which we want to determine $k_{\ell+1}$ (approximately). We also define $\mathcal{B}(\ell+1) := \mathcal{B}_{\mathcal{S}}(\ell+1) \cup \mathcal{B}_{\mathcal{N}^*}(\ell+1)$.

We describe now how we estimate the obtained profit for one specific candidate choice of $s \in S$. We try to pack items from $\tilde{\mathcal{I}}_{\ell+1}(s) := \left\{i \in \mathcal{I} : s_i \in (\tilde{k}_\ell, s]\right\}$ into

- the $\mathcal{N}^*$-boxes $\mathcal{B}_{\mathcal{N}^*}(\ell+1)$ and
- the $\mathcal{S}$-boxes $\mathcal{B}_{\mathcal{S}}(\ell+1)$, where for each $\mathcal{S}$-box $\mathcal{B}_{\mathcal{S}}(\ell+1)$, we use a volume of at most $\hat{a}_{B,\ell+1} \cdot \mathrm{VOL}(B)$ and ensure that we pack only items $i \in \tilde{\mathcal{I}}_{\ell+1}(s)$ for which $s_i \leq \epsilon \ell_{\min}(B)$.

We solve this subproblem approximately via the following integer program (IP(s)). Intuitively, we group items such that all items in the same group have the same size and profit, up to a factor of $1+\epsilon$. Formally, we define

- a size class $\mathcal{Q}_t = \{i \in \tilde{\mathcal{I}}_{\ell+1}(s) : s_i \in [(1+\epsilon)^t, (1+\epsilon)^{t+1})\}$ for each $t \in \mathcal{T}_Q = \{\lfloor \log_{1+\epsilon}(\tilde{k}_\ell) \rfloor, \dots, \lceil \log_{1+\epsilon}(s) \rceil\}$; we denote by $\hat{s}(t) := (1+\epsilon)^{t+1}$ the corresponding "rounded" size,
- a profit class $\mathcal{P}_{t'} = \{i \in \tilde{\mathcal{I}}_{\ell+1}(s) : p_i \in [(1+\epsilon)^{t'}, (1+\epsilon)^{t'+1})\}$ for each $t' \in \mathcal{T}_P = \{\lfloor \log_{1+\epsilon} \epsilon \frac{p_{\max}}{n} \rfloor, \dots, \lceil \log_{1+\epsilon} p_{\max} \rceil\}$; we denote by $\hat{p}(t') := (1+\epsilon)^{t'+1}$ the corresponding "rounded" profit and
- a set of pairs $\mathcal{T} := \{(t, t') : t \in \mathcal{T}_Q \wedge t' \in \mathcal{T}_P\}$.

Our subproblem is now equivalent (up to a factor of $1+\epsilon$ in the profit) to choosing how many items from each group are packed into which box, while maximizing the total profit of these items, such that

- for each $\mathcal{N}^*$-box $B \in \mathcal{B}_{\mathcal{N}^*}(\ell+1)$ the number of items packed into B is at most the number of grid cells denoted by $n(B) := \prod_{d'=1}^{d} \hat{n}_{d'}(B)$,

- for each $\mathcal{S}$-box $B \in \mathcal{B}_{\mathcal{S}}(\ell+1)$ the total volume of items packed into B does not exceed the designated volume for items in $\mathcal{I}_j$ reserved in B and
- for each pair (t,t') the number of items packed into all boxes is at most the number of available items in the corresponding group.

$$
\begin{aligned}
(\mathrm{IP}(s)) \quad \max \quad & \sum_{(t,t')\in\mathcal{T}} \sum_{B\in\mathcal{B}(\ell+1)} x_{t,t',B}\, p(t') & & \\
\text{s.t.} \quad & \sum_{(t,t')\in\mathcal{T}} x_{t,t',B} \le n(B) & & \forall B \in \mathcal{B}_{\mathcal{N}^*}(\ell+1) \\
& \sum_{(t,t')\in\mathcal{T}} x_{t,t',B}\, s(t)^d \le a_{B,\ell+1}\mathrm{VOL}_d(B) & & \forall B \in \mathcal{B}_{\mathcal{S}}(\ell+1) \\
& \sum_{B\in\mathcal{B}(\ell+1)} x_{t,t',B} \le n_{t,t'} & & \forall (t,t') \in \mathcal{T} \\
& x_{t,t',B} \in \mathbb{N}_0 & & \forall (t,t') \in \mathcal{T}, B \in \mathcal{B}(\ell+1)
\end{aligned}
$$

We cannot solve $(\mathrm{IP}(s))$ directly; however, we show that we can solve it approximately, losing only a factor of $1+\epsilon$. We describe now how to do this in time $(\log_{1+\epsilon}(N))^{O_\epsilon(1)}$. We start by guessing the $|\mathcal{B}(\ell+1)| \cdot |\mathcal{T}|/\epsilon = O_{\epsilon,d}(1)$ most profitable items in an optimal solution of $(\mathrm{IP}(s))$. To do this, we guess the profit type and size type of each of these items as well as which box they are packed in. For a single item this yields a total amount of $O_{\epsilon,d}((\log_{1+\epsilon}(N))^2)$ many possibilities, and hence at most $(\log_{1+\epsilon}(N))^{O_\epsilon(1)}$ possibilities overall. We adjust $(\mathrm{IP}(s))$ accordingly (i.e., reduce the right-hand sides of our constraints) and solve the LP relaxation of the remaining problem in time $\left(\log_{1+\epsilon}(N)\right)^{O(1)}$, yielding a solution x^*. We round it by simply defining $\bar{x}_{t,t',B} := \lfloor x^*_{t,t',B} \rfloor$ for each $(t,t') \in \mathcal{T}$ and each $B \in \mathcal{B}(\ell+1)$. This yields a solution consisting of the guessed items together with $\bar{x}$, where $\bar{x}$ represents the remaining items in our solution. Since we guessed the $|\mathcal{B}(\ell+1)| \cdot |\mathcal{T}|/\epsilon$ most profitable items before but there are only $|\mathcal{B}(\ell+1)| \cdot |\mathcal{T}|$ variables, we solve $(\mathrm{IP}(s))$ up to a factor of $1+\epsilon$.

▶ **Lemma 10.** *There is an algorithm with a running time of* $(\log_{1+\epsilon}(N))^{O(1)}$ *that computes a* $(1+\epsilon)$*-approximate solution for* $(\mathrm{IP}(s))$ *for each* s*; we denote by* $q(s)$ *the value of this solution. For two values* s, s' *with* $s \le s'$ *we have that* $q(s) \le q(s')$*.*

At the end of our binary search, we define $\tilde{k}_{\ell+1}$ to be the smallest value $s \in S$ for which $q(s) \ge (1-\epsilon)\hat{p}(\ell+1)$. Let $x^*_{\ell+1}$ denote the computed solution to $(\mathrm{IP}(s))$ corresponding to $s = \tilde{k}_{\ell+1}$. Based on the inductive assumption $\tilde{k}_\ell \le k_\ell$ and our choice of $\tilde{k}_{\ell+1}$, we can show $\tilde{k}_{\ell+1} \le k_{\ell+1}$. This is crucial as $\tilde{k}_{\ell+1}$ determines the side lengths of an $\mathcal{N}^*$-box B with $j_B = \ell+1$. Thus, in order to be able to pack our guessed boxes into the knapsack we must guarantee that these side lengths are not larger than the side lengths of the corresponding box in the packing underlying Lemma 6.

▶ **Lemma 11.** *We have that* $\tilde{k}_{\ell+1} \le k_{\ell+1}$*.*

After completing the r rounds of our indirect guessing framework, we have obtained values $\tilde{k}_1, \tilde{k}_2, \ldots, \tilde{k}_r$ and r integral solutions to $(\mathrm{IP}(\tilde{k}_1)), ..., (\mathrm{IP}(\tilde{k}_r))$. We can construct a feasible solution ALG by combining these solutions: for each $j \in [r]$ and each $\mathcal{N}^*$-box $B \in \mathcal{B}$ we define $s_{\max}(B) := \tilde{k}_{j_B}$ and we assign the items of each set $\tilde{\mathcal{I}}_j$ into the boxes in $\mathcal{B}$ according to our solution to $\mathrm{IP}(\tilde{k}_j))$.

For each $j \in \{1, \ldots, r\}$, each size class $\mathcal{Q}_t$, and each profit class $\mathcal{P}_{t'}$, there is a certain number of items from the set $\tilde{\mathcal{I}}_j \cap \mathcal{Q}_t \cap \mathcal{P}_{t'}$ that our solution contains; we denote this number by $z_{j,t,t'}$. It is irrelevant which exact items from this set we pick (up to a factor of $1+\epsilon$ in

the profit). Thus, we simply order the items from the set $\tilde{\mathcal{I}}_j \cap \mathcal{Q}_t \cap \mathcal{P}_{t'}$ in non-decreasing order of side lengths, select the first $z_{j,t,t'}$ items, and assign them to the corresponding boxes. Finally, we pack each $\mathcal{S}$-box B using the NFDH-algorithm (Lemma 5) and each $\mathcal{N}^*$-box B by placing each item into a single cell of the d-dimensional grid with side length $s_{\max}(B)$.

This yields an algorithm with a running time of $n \cdot (\log^2 n) + (\log N)^{O_{\epsilon,d}(1)}$. In the full version [7] we explain how we can improve it to $n \cdot (\log^2 n) + (\log n)^{O_{\epsilon,d}(1)}$. Key to this is to guess the basic quantities and to solve each integer program (IP(s)) in time $(\log n)^{O_{\epsilon,d}(1)}$, using additional technical improvements such as restricting the range of the considered item sizes and profits when solving (IP(s)). We would like to remark that the exponent of $\log n$ depends on the value $C_{\text{boxes}}(\epsilon, d)$ in Lemma 6 which in turn depends on the (not precisely specified) number of boxes used in the structured packing in [22].

▶ **Theorem 12.** *There is a $(1+\epsilon)$-approximation algorithm for the d-dimensional hypercube knapsack problem with a running time of $n \cdot (\log^2 n) + (\log n)^{O_{\epsilon,d}(1)}$.*

2.3 Dynamic algorithm

The algorithmic techniques above can be combined with our item data structure to derive a dynamic algorithm for the d-dimensional hypercube knapsack problem. Our algorithm supports the following operations:

- **(i)** Insertion and deletion of an item into our data structure,
- **(ii)** Output a $(1+\epsilon)$-estimate of the value of the optimal solution,
- **(iii)** Output a $(1+\epsilon)$-approximate solution ALG and
- **(iv)** Query where a given item is contained in ALG.

For operation (i) we simply add or delete an item from our item data structure (see Lemma 1) and our balanced binary search trees, which takes time $O(\log^2 n)$. In order to execute operation (ii), we run our algorithm described above, except for computing the explicit packing of the items in the end. Instead, we simply return the total profit of our solutions to the integer programs (IP(s)) that correspond the solution that we output at the end. This takes time $(\log n)^{O_{\epsilon,d}(1)}$ in total. If a $(1+\epsilon)$-approximate solution ALG is queried (operation (iii)), we also compute the exact set of items and their packing as described previously. Since their total number is $|\text{ALG}|$, we can compute and output ALG in time $O(|\text{ALG}| \cdot (\log n)) + (\log n)^{O_{\epsilon,d}(1)}$.

Finally, if it is queried whether a given item $i \in \mathcal{I}$ is in contained in ALG (operation (iv)), we determine the value $j \in \{1, \dots, r\}$, the size class $\mathcal{Q}_t$, and the profit class $\mathcal{P}_{t'}$ for which $i \in \tilde{\mathcal{I}}_j \cap \mathcal{Q}_t \cap \mathcal{P}_{t'}$. Recall that $z_{j,t,t'}$ denotes the total number of items from this set we select and we select the $z_{j,t,t'}$ items in this set of shortest side length. Hence, we output "$i \in$ ALG" if i is among the shortest $z_{j,t,t'}$ items in $\tilde{\mathcal{I}}_j \cap \mathcal{Q}_t \cap \mathcal{P}_{t'}$, and "$i \notin$ ALG" otherwise. This ensures that we give consistent answers between consecutive updates of the set $\mathcal{I}$.

▶ **Theorem 13.** *There is a dynamic algorithm for the d-dimensional hypercube knapsack problem that supports the following operations:*

- **(i)** *insertion or deletion of an item in time $O(\log^2 n)$,*
- **(ii)** *output a $(1+\epsilon)$-estimate of the value of the optimal solution in time $(\log n)^{O_{\epsilon,d}(1)}$,*
- **(iii)** *output a $(1+\epsilon)$-approximate solution* ALG *in time $O(|\text{ALG}| \cdot (\log n)) + (\log n)^{O_{\epsilon,d}(1)}$,*
- **(iv)** *query whether an item is contained in* ALG *in time $(\log n)^{O_{\epsilon,d}(1)}$.*

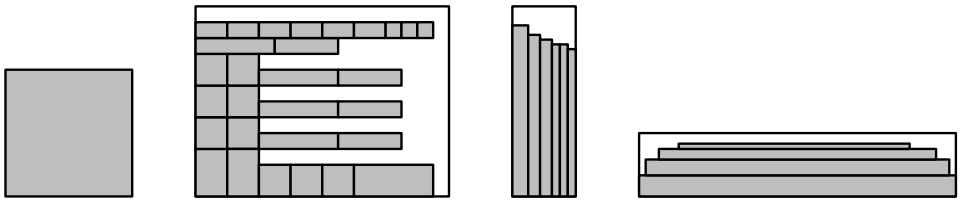


Figure 3 Visualization of packing of $\mathcal{L}$-, $\mathcal{S}$-,$\mathcal{H}$- and $\mathcal{V}$-box, respectively.

3 Algorithms for rectangles

In this section we give an overview of our algorithms for two-dimensional knapsack for rectangles (see [7] for a detailed description). First, we classify items into four groups: we say that an item is *large* if it is large compared to edge length of the knapsack in both dimensions and *small* if it is small in both dimensions.

An item of relatively large width (height) and relatively small height (width) is referred to as *vertical* (*horizontal*). We construct *easily guessable packing* using four types of boxes. The first type are *$\mathcal{L}$-boxes* which contain only one large item each. Also, we use *$\mathcal{H}$-boxes* inside which horizontal items are stacked on top of each other (see Figure 3), and correspondingly *$\mathcal{V}$-boxes* for vertical items. Finally, we use $\mathcal{S}$-boxes which are defined in the same manner as in the case of (hyper-)cubes. We prove that there always exists a $(2+\epsilon)$-approximate packing using these types of boxes.

▶ **Lemma 14** (Informal). *There exists an easily guessable packing for packing rectangles into a two-dimensional knapsack with a profit of at least* $(1/2-\epsilon)\mathrm{OPT}$.

In these packings, we can guess the height of each box (of each type) in $O_\epsilon(1)$ time and the width of each $\mathcal{S}$- and $\mathcal{V}$-box in $O(\mathrm{poly}(\log n))$ time (and additionally some other basic quantities). Then, we apply the indirect guessing framework in order to determine the widths of the $\mathcal{L}$- and $\mathcal{H}$-boxes.

▶ **Theorem 15.** *There is a $(2+\epsilon)$-approximation algorithm for the geometric knapsack problem for rectangles with a running time of $n\cdot(\log n)^4+(\log n)^{O_\epsilon(1)}$. Also, there is a dynamic $(2+\epsilon)$-approximation algorithm for the problem which supports the following operations:*

- *insert or delete an item in time $O(\log^4 n)$,*
- *output a $(2+\epsilon)$-estimate of the value of the optimal solution, or query whether an item is contained in* ALG*, in time $(\log n)^{O_\epsilon(1)}$, and*
- *output a $(2+\epsilon)$-approximate solution* ALG *in time $O(|\mathrm{ALG}|\cdot(\log n))+(\log n)^{O_\epsilon(1)}$.*

A natural open question is to improve our approximation ratio to $2-\delta$ for some constant $\delta>0$. This seems difficult since there is provably no corresponding structured packing with only $O_\epsilon(1)$ boxes [12]. The known polynomial time $(17/9+\epsilon)$-approximation uses an L-shaped container which is packed by a DP with a running time of $n^{\Omega_\epsilon(1)}$ [12]. It is not clear how to improve this to near-linear running time. However, if we are allowed to rotate the rectangles by 90 degrees, then it is possible to construct an easily guessable packing with $O_\epsilon(1)$ boxes and an approximation ratio of only $17/9+\epsilon$. We use here that we have more freedom to modify the optimal packing by rotating some of its items.

▶ **Lemma 16** (Informal). *If we are allowed to rotate the input rectangles, there exists an easily guessable packing into a two-dimensional knapsack with a profit of at least* $(9/17-\epsilon)\mathrm{OPT}$.

On the other hand, it becomes harder to compute a solution that corresonds to our easily guessable packing. The reason is that a horizontal or vertical item can now be assigned to an $\mathcal{H}$- *or* to a $\mathcal{V}$-box. This is particularly problematic since these two types of boxes are not treated symmetrically. Like before, we can guess the height of each box in time $O_\epsilon(1)$. However, for $\mathcal{V}$-boxes this yields a different kind of restriction than for $\mathcal{H}$-boxes.

▶ **Theorem 17.** *There is a* $(17/9+\epsilon)$*-approximation algorithm for the geometric knapsack problem for rectangles with rotations with a running time of* $n \cdot (\log n)^4 + (\log n)^{O_\epsilon(1)}$*. Also, there is a dynamic* $(17/9+\epsilon)$*-approximation algorithm for the problem which supports the following operations:*

- *insert or delete an item in time* $O(\log^4 n)$*,*
- *output a* $(17/9+\epsilon)$*-estimate of the value of the optimal solution, or query whether an item is contained in* ALG*, in time* $(\log n)^{O_\epsilon(1)}$*,*
- *output a* $(17/9+\epsilon)$*-approximate solution* ALG *in time* $|\text{ALG}| \cdot (\log n)^3 + (\log n)^{O_\epsilon(1)}$*.*

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Quadratic-time hardness of lcs and other sequence similarity measures. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2015)*, pages 59–78, 2015.
2. Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-SUM lower bounds for approximate distance oracles via additive combinatorics. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*, pages 391–404, 2023.
3. Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the two-dimensional geometric knapsack problem. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 1491–1505, 2014.
4. Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th Annual Symposium on Foundations of Computer Science (FOCS 2014)*, pages 661–670, 2014.
5. Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proceedings of the 56th Annual Symposium on Foundations of Computer Science (FOCS 2015)*, pages 79–97, 2015.
6. Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 1216–1235, 2018.
7. Moritz Buchem, Paul Deuker, and Andreas Wiese. Approximating the geometric knapsack problem in near-linear time and dynamically, 2024. `arXiv:2403.00536`.
8. Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.
9. Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. On problems equivalent to (min,+)-convolution. *ACM Transactions on Algorithms (TALG)*, 15(1):1–25, 2019.
10. Mingyang Deng, Ce Jin, and Xiao Mao. Approximating knapsack and partition via dense subset sums. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 2961–2979, 2023.
11. Franziska Eberle, Nicole Megow, Lukas Nölke, Bertrand Simon, and Andreas Wiese. Fully Dynamic Algorithms for Knapsack Problems with Polylogarithmic Update Time. In *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*, pages 18:1–18:17, 2021.
12. Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, Sandy Heydrich, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via L-packings. *ACM Transactions on Algorithms (TALG)*, 17(4):1–67, 2021.

13 Waldo Gálvez, Fabrizio Grandoni, Arindam Khan, Diego Ramírez-Romero, and Andreas Wiese. Improved Approximation Algorithms for 2-Dimensional Knapsack: Packing into Multiple L-Shapes, Spirals, and More. In *Proceedings of the 37th International Symposium on Computational Geometry (SoCG 2021)*, pages 39:1–39:17, 2021.

14 Leo J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science (SFCS 1978)*, pages 8–21, 1978.

15 Rolf Harren. Approximation algorithms for orthogonal packing problems for hypercubes. *Theoretical Computer Science*, 410(44):4504–4532, 2009.

16 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC 2015)*, pages 21–30, 2015.

17 Monika Henzinger, Stefan Neumann, Harald Räcke, and Stefan Schmid. Dynamic Maintenance of Monotone Dynamic Programs and Applications. In *Proceedings of the 40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, pages 36:1–36:16, 2023.

18 Sandy Heydrich and Andreas Wiese. Faster approximation schemes for the two-dimensional knapsack problem. *ACM Transactions on Algorithms (TALG)*, 15(4):1–28, 2019.

19 Klaus Jansen. On rectangle packing: maximizing benefits. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, 2004.

20 Klaus Jansen. Parameterized approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 39(4):1392–1412, 2010.

21 Klaus Jansen. A fast approximation scheme for the multiple knapsack problem. In *Proceedings of the 38th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2012)*, pages 313–324, 2012.

22 Klaus Jansen, Arindam Khan, Marvin Lira, and K. V. N. Sreenivas. A PTAS for Packing Hypercubes into a Knapsack. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, pages 78:1–78:20, 2022.

23 Klaus Jansen and Guochuan Zhang. Maximizing the number of packed rectangles. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT 2004)*, pages 362–371, 2004.

24 Hans Kellerer. A polynomial time approximation scheme for the multiple knapsack problem. In *Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (APPROX 1999)*, pages 51–62, 1999.

25 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the Fine-Grained Complexity of One-Dimensional Dynamic Programming. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, pages 21:1–21:15, 2017.

26 D. T. Lee and Franco P. Preparata. Computational geometry - a survey. *IEEE Transactions on Computers*, 100(12):1072–1101, 1984.

27 Arturo Merino and Andreas Wiese. On the Two-Dimensional Knapsack Problem for Convex Polygons. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, pages 84:1–84:16, 2020.

28 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 515–524, 2013.

29 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS 2010)*, pages 645–654, 2010.