On the Two-Dimensional Knapsack Problem for Convex Polygons

Arturo Merino
Technische Universität Berlin, Germany
merino@math.tu-berlin.de

Andreas Wiese
Universidad de Chile, Santiago, Chile
awiese@dii.uchile.cl

Abstract

We study the two-dimensional geometric knapsack problem for convex polygons. Given a set of weighted convex polygons and a square knapsack, the goal is to select the most profitable subset of the given polygons that fits non-overlappingly into the knapsack. We allow to rotate the polygons by arbitrary angles. We present a quasi-polynomial time $O(1)$-approximation algorithm for the general case and a polynomial time $O(1)$-approximation algorithm if all input polygons are triangles, both assuming polynomially bounded integral input data. Also, we give a quasi-polynomial time algorithm that computes a solution of optimal weight under resource augmentation, i.e., we allow to increase the size of the knapsack by a factor of $1 + \delta$ for some $\delta > 0$ but compare ourselves with the optimal solution for the original knapsack. To the best of our knowledge, these are the first results for two-dimensional geometric knapsack in which the input objects are more general than axis-parallel rectangles or circles and in which the input polygons can be rotated by arbitrary angles.

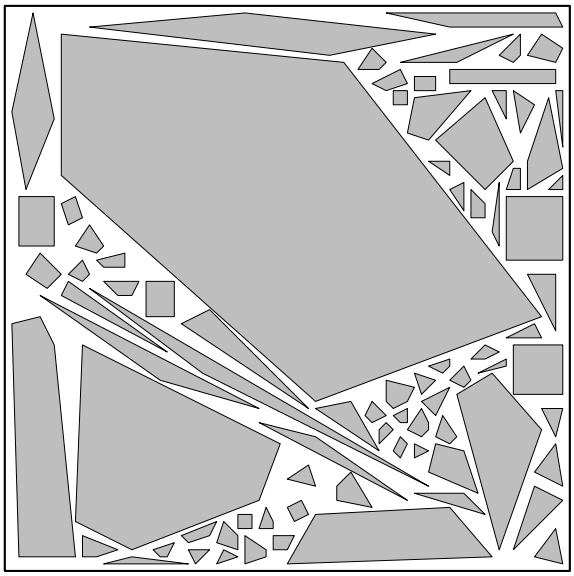
2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases Approximation algorithms, geometric knapsack problem, polygons, rotation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.84

Category Track A: Algorithms, Complexity and Games

Funding *Arturo Merino*: Partially supported by DFG Project 413902284 and ANID Becas Chile 2019-72200522.
Andreas Wiese: Partially supported by FONDECYT Regular grant 1170223.



1 Introduction

In the two-dimensional geometric knapsack problem (2DKP) we are given a square knapsack $K := [0, N] \times [0, N]$ for some integer N and a set of n convex polygons $\mathcal{P}$ where each polygon $P_i \in \mathcal{P}$ has a weight $w_i > 0$; we write $w(\mathcal{P}') := \sum_{P_i \in \mathcal{P}'} w_i$ for any set $\mathcal{P}' \subseteq \mathcal{P}$. The goal is to select a subset $\mathcal{P}' \subseteq \mathcal{P}$ of maximum total weight $w(\mathcal{P}')$ such that the polygons in $\mathcal{P}'$ fit non-overlapping into K if we translate and rotate them suitably (by arbitrary angles). 2DKP is a natural packing problem, the reader may think of cutting items out of a piece of raw material like metal or wood, cutting cookings out of dough, or, in three dimensions, loading cargo into a ship or a truck. In particular, in these applications the respective items can have various kinds of shapes. Also note that 2DKP is a natural geometric generalization of the classical one-dimensional knapsack problem.

Our understanding of 2DKP highly depends on the type of input objects. If all polygons are axis-parallel squares there is a $(1 + \epsilon)$-approximation with a running time of the form $O_\epsilon(1)n^{O(1)}$ (i.e., an EPTAS) [6], and there can be no FPTAS (unless $\mathsf{P} = \mathsf{NP}$) since the problem is strongly NP-hard [11]. For axis-parallel rectangles there is a polynomial time $(17/9 + \epsilon) < 1.89$-approximation algorithm and a $(3/2 + \epsilon)$-approximation if the items can be rotated by exactly 90 degrees [5]. If the input data is quasi-polynomially bounded there is even a $(1 + \epsilon)$-approximation in quasi-polynomial time [2], with and without the possibility to rotate items by 90 degrees. For circles a $(1 + \epsilon)$-approximation is known under resource augmentation in one dimension if the weight of each circle equals its area [12].

To the best of our knowledge, there is no result known for 2DKP for shapes different than axis-parallel rectangles and circles. Also, there is no result known in which input polygons are allowed to be rotated by angles different than 90 degrees. However, in the applications of 2DKP the items might have shapes that are more complicated than rectangles or circles. Also, it makes sense to allow rotations by arbitrary angles, e.g., when cutting items out of some material. In this paper, we present the first results for 2DKP in these settings.

1.1 Our contribution

We study 2DKP for arbitrary convex polygons, allowing to rotate them by arbitrary angles. Note that due to the latter, it might be that some optimal solution places the vertices of the polygons on irrational coordinates, even if all input numbers are integers. Our first results are a quasi-polynomial time $O(1)$-approximation algorithm for general convex polygons and a polynomial time $O(1)$-approximation algorithm for triangles.

By rotation we can assume for each input polygon that the line segment defining its diameter is horizontal. We identify three different types of polygons for which we employ different strategies for packing them, see Figure 1a). First, we consider the *easy* polygons which are the polygons whose bounding boxes fit into the knapsack without rotation. We pack these polygons such that their bounding boxes do not intersect. Using area arguments and the Steinberg's algorithm [13] we obtain a $O(1)$-approximation for the easy polygons. Then we consider the *medium* polygons which are the polygons whose bounding boxes easily fit into the knapsack if we can rotate them by 45 degrees. We use a special type of packing in which the bounding boxes are rotated by 45 degrees and then stacked on top of each other, see Figure 1b). More precisely, we group the polygons by the widths of their bounding boxes and to each group we assign two rectangular containers in the packing. We compute the essentially optimal solution of this type by solving a generalization of one-dimensional knapsack for each group. Our key structural insight for medium polygons is that such a solution is $O(1)$-approximate. To this end, we prove that in OPT the medium polygons of

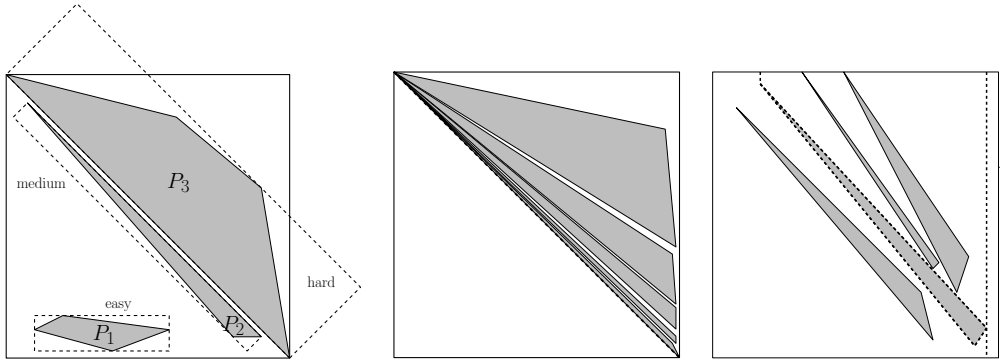


Figure 1 (a) An easy, a medium, and a hard polygon and their bounding boxes (b): Triangles packed in a top-left packing (c) The geometric DP subdivides the knapsack along the dashed lines and then recurses within each resulting area.

each group occupy an area that is by at most a constant factor bigger than the corresponding containers, and that a constant fraction of these polygons fit into the containers. In particular, we show that medium polygons with very wide bounding boxes lie in a very small hexagonical area close to the diagonal of the knapsack. Our routines for easy and medium polygons run in polynomial time.

It remains to pack the *hard* polygons whose bounding boxes just fit into the knapsack or do not fit at all, even under rotation. Note that this does not imply that the polygon itself does not fit. Our key insight is that there can be only $O(\log N)$ such polygons in the optimal solution, at most $O(1)$ from each group. Therefore, we can guess these polygons in quasi-polynomial time, assuming that N is quasi-polynomially bounded. However, in contrast to other packing problems, it is not trivial to check whether a set of given polygons fits into the knapsack since we can rotate them by arbitrary angles and we cannot enumerate all possibilities for the angles. However, we show that by losing a constant factor in the approximation guarantee we can assume that the placement of each hard polygon comes from a precomputable polynomial size set and hence we can guess the placements of the $O(\log N)$ hard polygons in quasi-polynomial time.

▶ **Theorem 1.** *There is a $O(1)$-approximation algorithm for 2DKP with a running time of $(nN)^{(\log nN)^{O(1)}}$.*

If all hard polygons are triangles we present even a *polynomial* time $O(1)$-approximation algorithm. We split the triangles in OPT into two types, for one type we show that a constant fraction of it can be packed in what we call *top-left-packings*, see Figure 1b). In these packings, the triangles are sorted by the lengths of their longest edges and placed on top of each other in a triangular area. We devise a dynamic program (DP) that essentially computes the most profitable top-left-packing. For proving that this yields a $O(1)$-approximation, we need some careful arguments for rearranging a subset of the triangles with large weight to obtain a packing that our DP can compute. We observe that essentially all hard polygons in OPT must intersect the horizontal line that contains the mid-point of the knapsack. Our key insight is that if we pack a triangle in a top-left-packing then it intersects this line to a similar extent as in OPT. Then we derive a sufficient condition when a set of triangles fits in a top-left-packing, based on by how much they overlap this line.

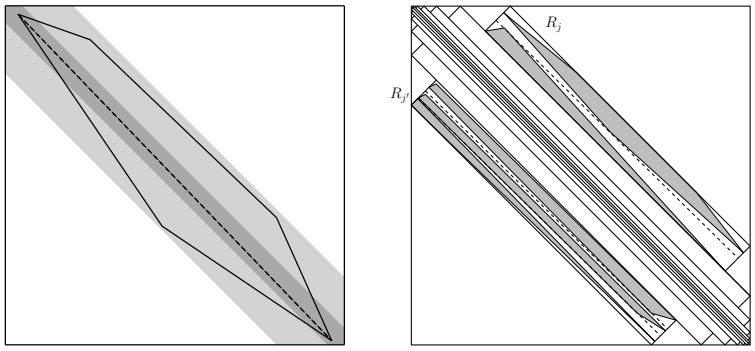


Figure 2 Left: Assume that the polygon (black line segments) is a medium polygon contained in the set $\mathcal{P}_j$. Then the diagonal (dashed) line segment must lie into the dark-gray area and the whole polygon must be contained in the light-gray area. Right: The containers for the medium polygons of the different groups. Within each container, the polygons are stacked on top of each other such that their respective bounding boxes do not intersect.

For the other type of triangles we use a geometric dynamic program. In this DP we recursively subdivide the knapsack into subareas in which we search for the optimal solution recursively, see Figure 1c). In the process we guess the placements of some triangles from OPT. Again, by losing a constant factor we can assume that for each triangle in OPT there are only a polynomial number of possible placements. By exploiting structural properties of this type of triangles we ensure that the number of needed DP-cells is bounded by a polynomial. A key difficulty is that we sometimes split the knapsack into two parts on which we recurse independently. Then we need to ensure that we do not select some (possibly high weight) triangle in both parts. To this end, we globally select at most one triangle from each of the $O(\log N)$ groups (losing a constant factor) and when we recurse, we guess for each subproblem from which of the $O(\log N)$ groups it contains a triangle in OPT. This yields only $2^{O(\log N)} = N^{O(1)}$ guesses.

▶ **Theorem 2.** *There is a $O(1)$-approximation algorithm for 2DKP with a running time of $(nN)^{O(1)}$ if all input polygons are triangles.*

Then we study the setting of resource augmentation, i.e., we compute a solution that fits into a larger knapsack of size $(1+\delta)N \times (1+\delta)N$ for some constant $\delta > 0$ and compare ourselves with a solution that fits into the original knapsack of size $N \times N$. We show that then the optimal solution can contain only *constantly* many hard polygons and hence we can guess them in *polynomial* time.

▶ **Theorem 3.** *There is a $O(1)$-approximation algorithm for 2DKP under $(1+\delta)$-resource augmentation with a running time of $n^{O_\delta(1)}$.*

Finally, we present a quasi-polynomial time algorithm that computes a solution of weight at least $w(\text{OPT})$ (i.e., we do not lose any factor in the approximation guarantee) that is feasible under resource augmentation. This algorithm does not use the above classification of polygons into easy, medium, and hard polygons. Instead, we prove that if we can increase the size of the knapsack slightly we can ensure that for the input polygons there are only $(\log n)^{O_\delta(1)}$ different shapes by enlarging the polygons suitably. Also, we show that we need to allow only a polynomial number of possible placements and rotations for each input polygon, *without* sacrificing any polygons from OPT. Then we use a technique from [1] implying that there is a balanced separator for the polygons in OPT with only $(\log n)^{O_\delta(1)}$ edges and

which intersects polygons from OPT with only very small area. We guess the separator, guess how many polygons of each type are placed inside and outside the separator, and then recurse on each of these parts. Some polygons are intersected by the balanced separator. However, we ensure that they have very small area in total and hence we can place them into the additional space of the knapsack that we gain due to the resource augmentation. This generalizes a result in [2] for axis-parallel rectangles.

▶ **Theorem 4.** *There is an algorithm for 2DKP under $(1+\delta)$-resource augmentation with a running time of $n^{O_\delta(\log n)^{O(1)}}$ that computes a solution of weight at least $w(\mathrm{OPT})$.*

In our approximation algorithms, we focus on a clean exposition of our methodology for obtaining $O(1)$-approximations, rather than on optimizing the actual approximation ratio. Due to space constraints, most proofs and details had to be ommited in this extended abstract.

1.2 Other related work

Prior to the results mentioned above, polynomial time $(2+\epsilon)$-approximation algorithms for 2DKP for axis-parallel rectangles were presented by Jansen and Zhang [10, 9]. For the same setting, a PTAS is known under resource augmentation in one dimension [7] and a polynomial time algorithm computing a solution with optimum weight under resource augmentation in both dimensions [6]. Also, there is a PTAS if the weight of each rectangle equals its area [3]. For squares, Jansen and Solis-Oba presented a PTAS [8].

2 Constant factor approximation algorithms

In this section we present our quasi-polynomial time $O(1)$-approximation algorithm for general convex polygons and our polynomial time $O(1)$-approximation algorithm for triangles., assuming polynomially bounded input data. Our strategy is to partition the input polygons $\mathcal{P}$ into three classes, easy, medium, and hard polygons, and then to devise algorithms for each class separately.

Let $K := [0, N] \times [0, N]$ denote the given knapsack. We assume that each input polygon is described by the coordinates of its vertices which we assume to be integral. First, we rotate each polygon in $\mathcal{P}$ such that its longest diagonal (i.e., the line segment that connects the two vertices of largest distance) is horizontal. For each polygon $P_i \in \mathcal{P}$ denote by $(x_{i,1}, y_{i,1}), ..., (x_{i,k_i}, y_{i,k_i})$ the new coordinates of its vertices. Observe that due to the rotation, the resulting coordinates might not be integral, and possibly not even rational. We will take this into account when we define our algorithms. For each $P_i \in \mathcal{P}$ we define its *bounding box* B_i to be the smallest axis-parallel rectangle that contains P_i. Formally, we define $B_i := [\min_\ell x_{i,\ell}, \max_\ell x_{i,\ell}] \times [\min_\ell y_{i,\ell}, \max_\ell y_{i,\ell}]$. For each polygon P_i let $\ell_i := \max_\ell x_{i,\ell} - \min_\ell x_{i,\ell}$ and $h_i := \max_\ell y_{i,\ell} - \min_\ell y_{i,\ell}$. If necessary we will work with suitable estimates of these values later, considering that they might be irrational and hence we cannot compute them exactly.

We first distinguish the input polygons into *easy*, *medium*, and *hard* polygons. We say that a polygon P_i is *easy* if B_i fits into K without rotation, i.e., such that $\ell_i \le N$ and $h_i \le N$. Denote by $\mathcal{P}_E \subseteq \mathcal{P}$ the set of easy polygons. Note that the bounding box of a polygon in $\mathcal{P} \setminus \mathcal{P}_E$ might still fit into K if we rotate it suitably. Intuitively, we define the medium polygons to be the polygons P_i whose bounding box B_i fits into K with quite some slack if we rotate B_i properly and the hard polygons are the remaining polygons (in particular those polygons whose bounding box does not fit at all into K).

Formally, for each polygon $P_i \in \mathcal{P}$ we define $h'_i := \sqrt{2}N - \ell_i$. The intuition for h'_i is that a rectangle of width ℓ_i and height h'_i is the highest rectangle of width ℓ_i that still fits into K.

▶ **Lemma 5.** *Let $P_i \in \mathcal{P}$. A rectangle of width ℓ_i and height h'_i fits into K (if we rotate it by 45°) but a rectangle of width ℓ_i and of height larger than h'_i does not fit into K.*

Hence, if h_i is much smaller than h'_i then B_i fits into K with quite some slack. Therefore, we define that a polygon $P_i \in \mathcal{P} \setminus \mathcal{P}_E$ is *medium* if $h_i \leq h'_i/8$ and *hard* otherwise. Denote by $\mathcal{P}_M \subseteq \mathcal{P}$ and $\mathcal{P}_H \subseteq \mathcal{P}$ the medium and hard polygons, respectively. We will present $O(1)$-approximation algorithms for each of the sets $\mathcal{P}_E, \mathcal{P}_M, \mathcal{P}_H$ separately. The best of the computed sets will then yield a $O(1)$-approximation overall.

For the easy polygons, we construct a polynomial time $O(1)$-approximation algorithm in which we select polygons such that we can pack their bounding boxes as non-overlapping rectangles using Steinberg's algorithm [4], see Section 2.1. The approximation ratio follows from area arguments.

▶ **Lemma 6.** *There is a polynomial time algorithm that computes a solution $\mathcal{P}'_E \subseteq \mathcal{P}_E$ with $w(\mathrm{OPT} \cap \mathcal{P}_E) \leq O(w(\mathcal{P}'_E))$.*

For the medium polygons, we obtain a $O(1)$-approximation algorithm using a different packing strategy, see Section 2.2.

▶ **Lemma 7.** *There is an algorithm with a running time of $n^{O(1)}$ that computes a solution $\mathcal{P}'_M \subseteq \mathcal{P}_M$ with $w(\mathrm{OPT} \cap \mathcal{P}_M) \leq O(w(\mathcal{P}'_M))$.*

The most difficult polygons are the hard polygons. First, we show that in quasi-polynomial time we can obtain a $O(1)$-approximation for them, see Section 2.3.

▶ **Lemma 8.** *There is an algorithm with a running time of $(nN)^{(\log nN)^{O(1)}}$ that computes a solution $\mathcal{P}'_H \subseteq \mathcal{P}_H$ with $w(\mathrm{OPT} \cap \mathcal{P}_H) \leq O(w(\mathcal{P}'_H))$.*

Combining Lemmas 6, 7, and 8 yields Theorem 1. If all polygons are triangles, we obtain a $O(1)$-approximation even in polynomial time. The following lemma is proved in Section 2.3.1 and together with Lemmas 6 and 7 implies Theorem 2.

▶ **Lemma 9.** *If all input polygons are triangles, then there is an algorithm with a running time of $(nN)^{O(1)}$ that computes a solution $\mathcal{P}'_H \subseteq \mathcal{P}_H$ with $w(\mathrm{OPT} \cap \mathcal{P}_H) \leq O(w(\mathcal{P}'_H))$.*

Orthogonal to the characterization into easy, medium and hard polygons, we subdivide the polygons in $\mathcal{P}$ further into classes according to the respective values ℓ_i. More precisely, we do this according to their difference between ℓ_i and the diameter of K, i.e., $\sqrt{2}N$. Formally, for each $j \in \mathbb{Z}$ we define $\mathcal{P}_j := \{P_i \in \mathcal{P} | \ell_i \in [\sqrt{2}N - 2^j, \sqrt{2}N - 2^{j-1})\}$ and additionally $\mathcal{P}_{-\infty} := \{P_i \in \mathcal{P} | \ell_i = \sqrt{2}N\}$. Note that for each polygon $P_i \in \mathcal{P}$ we can compute the group $\mathcal{P}_j$ even though ℓ_i might be irrational.

2.1 Easy polygons

We present a $O(1)$-approximation algorithm for the polygons in $\mathcal{P}_E$. First, we show that the area of each polygon is at least half of the area of its bounding box. We will use this later for defining lower bounds using area arguments.

▶ **Lemma 10.** *For each $P_i \in \mathcal{P}$ it holds that* $\mathrm{area}(P_i) \geq \frac{1}{2}\mathrm{area}(B_i)$.

On the other hand, it is known that we can pack any set of axis-parallel rectangles into K, as long as their total area is at most $\mathrm{area}(K)/2$ and each single rectangle fits into K.

▶ **Theorem 11** ([13]). *Let $r_1, ..., r_k$ be a set of axis-parallel rectangles such that $\sum_{i=1}^{k} \text{area}(r_i) \leq \text{area}(K)/2$ and each individual rectangle r_i fits into K. Then there is a polynomial time algorithm that packs $r_1, ..., r_k$ into K.*

We first compute (essentially) the most profitable set of polygons from $\mathcal{P}_E$ whose total area is at most $\text{area}(K)$ via a reduction to one-dimensional knapsack.

▶ **Lemma 12.** *In time $(\frac{n}{\epsilon})^{O(1)}$ we can compute a set of polygons $\mathcal{P}' \subseteq \mathcal{P}_E$ such that $w(\mathcal{P}') \geq (1-\epsilon) w(\text{OPT} \cap \mathcal{P}_E)$ and $\sum_{P_i \in \mathcal{P}_E} \text{area}(P_i) \leq \text{area}(K)$.*

The idea is now to partition $\mathcal{P}'$ into at most 7 sets $\mathcal{P}'_1, ..., \mathcal{P}'_7$. Hence, one of these sets must contain at least a profit of $w(\mathcal{P}')/7$. We define this partition such that each set $\mathcal{P}'_j$ contains only one polygon or its polygons have a total area of at most $\text{area}(K)/4$.

▶ **Lemma 13.** *Given a set $\mathcal{P}' \subseteq \mathcal{P}_E$ with $\sum_{P_i \in \mathcal{P}_E} \text{area}(P_i) \leq \text{area}(K)$. In polynomial time we can compute a set $\mathcal{P}'' \subseteq \mathcal{P}'$ with $w(\mathcal{P}'') \geq \frac{1}{7} w(\mathcal{P}')$ and additionally $\sum_{P_i \in \mathcal{P}''} \text{area}(P_i) \leq \text{area}(K)/4$ or $|\mathcal{P}''| = 1$.*

If $|\mathcal{P}''| = 1$ we simply pack the single polygon in $\mathcal{P}''$ into the knapsack. Otherwise, using Lemmas 10 and 12 and Theorem 11 we know that we can pack the bounding boxes of the polygons in $\mathcal{P}''$ into K. Note that their heights and widths might be irrational. Therefore, we slightly increase them such that these values become rational, before applying Theorem 11 to compute the actual packing. If as a result the total area of the bounding boxes exceeds $\text{area}(K)/2$ we partition them into two sets where each set satisfies that the total area of the bounding boxes is at most $\text{area}(K)/2$ or it contains only one polygon and we keep the more profitable of these two sets (hence losing a factor of 2 in the approximation ratio). This yields a $O(1)$-approximation algorithm for the easy polygons and thus proves Lemma 6.

2.2 Medium polygons

We describe a $O(1)$-approximation algorithm for the polygons in $\mathcal{P}_M$. In its solution, for each $j \in \mathbb{Z}$ we will define two rectangular containers R_j, R'_j for polygons in $\mathcal{P}_M \cap \mathcal{P}_j$, each of them having width $\sqrt{2}N - 2^{j-1}$ and height 2^{j-3}, see Figures 2. Let $\mathcal{R} := \cup_j \{R_j, R'_j\}$. First, we show that we can pack all containers in $\mathcal{R}$ into K (if we rotate them by 45°).

▶ **Lemma 14.** *The rectangles in $\mathcal{R}$ can be packed non-overlappingly into K.*

For each $j \in \mathbb{Z}$ we will compute a set of polygons $\mathcal{P}'_j \subseteq \mathcal{P}_M \cap \mathcal{P}_j$ of large weight. Within each container R_j, R'_j we will stack the bounding boxes of the polygons in $\mathcal{P}'_j$ on top of each other and then place the polygons in $\mathcal{P}'_j$ in their respective bounding boxes, see Figure 2. In particular, a set of items $\mathcal{P}''_j \subseteq \mathcal{P}_j$ fits into R_j (or R'_j) using this strategy if and only if $h(\mathcal{P}''_j) := \sum_{P_i \in \mathcal{P}''_j} h_i \leq 2^{j-3}$. Observe that for a polygon $P_i \in \mathcal{P}_j$ with $P_i \in \mathcal{P}_H$ it is not necessarily true that $h_i \leq 2^{j-3}$ and hence for hard polygons this strategy is not suitable. We compute the essentially most profitable set of items $\mathcal{P}'_j$ that fits into R_j and R'_j with the above strategy. For this, we need to solve a variation of one-dimensional knapsack with two knapsacks (instead of one) that represent R_j and R'_j. The value h_i for a polygon P_i might be irrational, therefore we work with a $(1+\epsilon)$-estimate of h_i instead. This costs only a factor $O(1)$ in the approximation guarantee.

▶ **Lemma 15.** *Let $\epsilon > 0$. For each $j \in \mathbb{Z}$ there is an algorithm with a running time of $n^{O(\frac{1}{\epsilon})}$ that computes two disjoint sets $\mathcal{P}'_{j,1}, \mathcal{P}'_{j,2} \subseteq \mathcal{P}_j \cap \mathcal{P}_M$ such that $h(\mathcal{P}'_{j,1}) \leq 2^{j-3}$ and $h(\mathcal{P}'_{j,2}) \leq 2^{j-3}$ and $w(\mathcal{P}^*{}_{j,1} \cup \mathcal{P}^*{}_{j,2}) \leq O(w(\mathcal{P}'_{j,1} \cup \mathcal{P}'_{j,2}))$ for any disjoint sets $\mathcal{P}^*{}_{j,1}, \mathcal{P}^*{}_{j,2} \subseteq \mathcal{P}_j \cap \mathcal{P}_M$ such that $h(\mathcal{P}^*_{j,1}) \leq 2^{j-3}$ and $h(\mathcal{P}^*_{j,2}) \leq 2^{j-3}$.*

For each $j \in \mathbb{Z}$ with $\mathcal{P}_j \cap \mathcal{P}_M \neq \emptyset$ we apply Lemma 15 and obtain sets $\mathcal{P}'_{j,1}, \mathcal{P}'_{j,2}$. We pack $\mathcal{P}'_{j,1}$ into R_j and $\mathcal{P}'_{j,2}$ into R'_j, using that $h(\mathcal{P}'_{j,1}) \leq h(R_j)$ and $h(\mathcal{P}'_{j,2}) \leq h(R'_j)$. Then we pack all containers R_j, R'_j for each $j \in \mathbb{Z}$ into K, using Lemma 14.

Let $\mathcal{P}'_M := \bigcup_j \mathcal{P}'_{j,1} \cup \mathcal{P}'_{j,2}$ denote the selected polygons. We want to show that $\mathcal{P}'_M$ has large weight; more precisely we want to show that $w(\mathrm{OPT} \cap \mathcal{P}_M) \leq O(w(\mathcal{P}'_M))$. First, we show that for each $j \in \mathbb{Z}$ the polygons in $\mathcal{P}_j \cap \mathcal{P}_M \cap \mathrm{OPT}$ have bounded area. To this end, we show that they are contained inside a certain (irregular) hexagon (see Figure 2) which has small area if the polygons $P_i \in \mathcal{P}_j$ are wide, i.e., if ℓ_i is close to $\sqrt{2}N$. The reason is that then P_i must be placed close to the diagonal of the knapsack and on the other hand h_i is relatively small (since P_i is medium), which implies that all of P_i lies close to the diagonal of the knapsack. For any object $O \subseteq \mathbb{R}^2$ we define $\mathrm{area}(O)$ to be its area.

▶ **Lemma 16.** *For each j it holds that* $\mathrm{area}(\mathcal{P}_j \cap \mathcal{P}_M) \leq O(\mathrm{area}(R_j \cup R'_j))$.

Using this, we can partition $\mathcal{P}_j \cap \mathcal{P}_M \cap \mathrm{OPT}$ into at most $O(1)$ subsets such that for each subset $\mathcal{P}'$ it holds that $h(\mathcal{P}') \leq 2^{j-3}$ and hence $\mathcal{P}'$ fits into R_j (and R'_j) using our packing strategy above. Here we use that each medium polygon $P_i \in \mathcal{P}_j$ satisfies that $h_i \leq 2^{j-3}$.

▶ **Lemma 17.** *For each $j \in \mathbb{Z}$ there are disjoint set $\mathcal{P}^*{}_{j,1}, \mathcal{P}^*{}_{j,2} \subseteq \mathcal{P}_j \cap \mathcal{P}_M \cap \mathrm{OPT}$ with $w(\mathcal{P}_j \cap \mathcal{P}_M \cap \mathrm{OPT}) \leq O(w(\mathcal{P}^*{}_{j,1} \cup \mathcal{P}^*{}_{j,2}))$ such that $h(\mathcal{P}^*_{j,1}) \leq 2^{j-3}$ and $h(\mathcal{P}^*_{j,2}) \leq 2^{j-3}$.*

By combining Lemmas 14, 15 and 17 we obtain the proof of Lemma 7.

2.3 Hard polygons

We first show that for each class $\mathcal{P}_j$ there are at most a constant number of polygons from $\mathcal{P}_j \cap \mathcal{P}_H$ in OPT, and that for only $O(\log N)$ classes $\mathcal{P}_j$ it holds that $\mathcal{P}_j \cap \mathcal{P}_H \neq \emptyset$.

▶ **Lemma 18.** *For each $j \in \mathbb{Z}$ it holds that $|\mathcal{P}_j \cap \mathcal{P}_H \cap \mathrm{OPT}| \leq O(1)$. Also, if $\mathcal{P}_j \cap \mathcal{P}_H \neq \emptyset$ then $j \in \{j_{\min}, ..., j_{\max}\}$ with $j_{\min} := -\lceil \log N \rceil$ and $j_{\max} := 1 + \lceil \log((\sqrt{2}-1)N) \rceil$.*

We describe now a quasi-polynomial time algorithm for hard polygons, i.e., we want to prove Lemma 8. Lemma 18 implies that $|\mathcal{P}_H \cap \mathrm{OPT}| \leq O(\log N)$. Therefore, we can enumerate all possibilities for $\mathcal{P}_H \cap \mathrm{OPT}$ in time $n^{O(\log N)}$. For each for each enumerated set $\mathcal{P}'_H \subseteq \mathcal{P}_H$ we need to check whether it fits into K. We cannot try all possibilities for placing $\mathcal{P}'_H$ into K since we are allowed tof rotate the polygons in $\mathcal{P}'_H$ by arbitrary angles. To this end, we show that there is a subset of $\mathcal{P}_H \cap \mathrm{OPT}$ of large weight which contains only a single polygon or it does not use the complete space of the knapsack but leaves some empty space. We use this empty space to move the polygons slightly and rotate them such that each of them is placed in one out of $(nN)^{O(1)}$ different positions that we can compute beforehand. Hence, we can guess all positions of these polygons in time $(nN)^{O(\log N)}$. We define that a *placement* of a polygon $P_i \in \mathcal{P}$ inside K is a polygon $\tilde{P}_i$ such that $d + \mathrm{rot}_\alpha(P_i) = \tilde{P}_i \subseteq K$ where $d \in \mathbb{R}^2$ and $\mathrm{rot}_\alpha(P_i)$ is the polygon that we obtain when we rotate P_i by an angle α clockwise around its first vertex.

▶ **Lemma 19.** *For each polygon $P_i \in \mathcal{P}_H$ we can compute a set of $(nN)^{O(1)}$ possible placements $\mathcal{L}_i$ in time $(nN)^{O(1)}$ such that there exists a set $\mathcal{P}'_H \subseteq \mathcal{P}_H \cap \mathrm{OPT}$ with $w(\mathcal{P}_H \cap \mathrm{OPT}) \leq O(w(\mathcal{P}'_H))$ which can be packed into K such that each polygon P_i is packed according to a placement in $\mathcal{L}_i$.*

This yields the proof of Lemma 8.

2.3.1 Hard triangles

In this section we present a $O(1)$-approximation algorithm in *polynomial* time for hard polygons assuming that they are all triangles, i.e., we prove Lemma 9. Slightly abusing notation, denote by OPT the set $\mathcal{P}'_H$ obtained by applying Lemma 19. We distinguish the triangles in OPT into two types: *edge-facing* triangles and *corner-facing* triangles. Let $P_i \in \text{OPT} \cap \mathcal{P}_H$, let e_1, e_2 denote the two longest edges of P_i, and let v_i^* the vertex of P_i adjacent to e_1 and e_2. Let $R_i^{(1)}$ and $R_i^{(2)}$ be the two rays that originate at v_i^* and that contain e_1 and e_2, respectively, in the placement of P_i in OPT. We have that $R_i^{(1)} \setminus \{v_i^*\}$ and $R_i^{(2)} \setminus \{v_i^*\}$ intersect at most one edge of the knapsack each. If $R_i^{(1)} \setminus \{v_i^*\}$ and $R_i^{(2)} \setminus \{v_i^*\}$ intersect the same edge of the knapsack then we say that P_i is *edge-facing,* if one of them intersects a horizontal edge and the other one intersects a vertical edge we say that P_i is *corner-facing.* The next lemma shows that there can be only $O(1)$ triangles in $\text{OPT} \cap \mathcal{P}_H$ that are neither edge- nor corner-facing, and therefore we compute a $O(1)$-approximation with respect to the total profit of such triangles by simply selecting the input triangle with maximum weight.

▶ **Lemma 20.** *There can be at most $O(1)$ triangles in $\text{OPT} \cap \mathcal{P}_H$ that are neither edge-facing nor corner-facing.*

Let p_{TL}, p_{TR}, p_{BL}, and p_{BR} denote the top left, top right, bottom left, and bottom right corners of K, respectively, and let $p_M := \binom{N/2}{N/2}$, $p_L := \binom{0}{N/2}$, and $p_R := \binom{N}{N/2}$, see Figure 3. By losing a factor $O(1)$ we assume from now on that that OPT contains at most one hard triangle from each group $\mathcal{P}_j$, using Lemma 18.

Let $\text{OPT}_{\text{EF}} \subseteq \text{OPT} \cap \mathcal{P}_H$ denote the edge-facing hard triangles in OPT and denote by $\text{OPT}_{\text{CF}} \subseteq \text{OPT} \cap \mathcal{P}_H$ the corner-facing hard triangles in OPT. In the remainder of this section we present now $O(1)$-approximation algorithms for edge-facing and for corner-facing triangles in $\mathcal{P}_H$. By selecting the best solution among the two we obtain the proof of Lemma 9.

Edge-facing triangles

We define a special type of solutions called *top-left-packings* that our algorithm will compute. We will show later that there are solutions of this type whose profit is at least a constant fraction of the profit of OPT_{EF}.

For each $t \in \mathbb{N}$ let $p_t := p_M + \frac{t}{N^2}\binom{1}{0}$. Let $\mathcal{P}' = \{P_{i_1}, ..., P_{i_k}\}$ be a set of triangles that are ordered according to the groups $\mathcal{P}_j$, i.e., such that for any $P_{i_\ell}, P_{i_{\ell+1}} \in \mathcal{P}'$ with $P_{i_\ell} \in \mathcal{P}_j$ and $P_{i_{\ell+1}} \in \mathcal{P}_{j'}$ for some j, j' it holds that $j \leq j'$. We define a placement of $\mathcal{P}'$ that we call a *top-left-packing.* First, we place P_{i_1} such that $v_{i_1}^*$ concides with p_{TL} and one edge of P_{i_1} lies on the diagonal of K that connects p_{TL} and p_0. Note that there is a unique way to place P_{i_1} in this way. Iteratively, suppose that we have packed triangles $\{P_{i_1}, ..., P_{i_\ell}\}$ such that for each triangle $P_{i_{\ell'}}$ in this set its respective vertex $v_{i_{\ell'}}^*$ coincides with p_{TL}, see Figure 1c). Intuitively, we pack $P_{i_{\ell+1}}$ on top of P_{i_ℓ} such that $v_{i_{\ell+1}}^*$ coincides with p_{TL}. Let t be the smallest integer such that the line segment connecting p_t and p_R has empty intersection with each triangle $P_{i_1}, ..., P_{i_\ell}$ according to our placement. We place $P_{i_{\ell+1}}$ such that $v_{i_{\ell+1}}^*$ concides with p_{TL} and one of its edges lies on the line that contains p_{TL} and p_t. There is a unique way to place $P_{i_{\ell+1}}$ in this way. We continue until we placed all triangles in $\mathcal{P}'$. If all of them are placed completely inside K we say that the resulting solution is a *top-left-packing* and that $\mathcal{P}'$ is *top-left-packable*. We define *bottom-right-packing* and *bottom-right-packable* symmetrically, mirroring the above definition along the line that contains p_{BL} and p_{TR}.

In the next lemma we show that there is always a top-left-packable or a bottom-right-packable solution with large profit compared to $\mathcal{P}_H \cap \mathrm{OPT}$ or there is a single triangle with large profit.

▶ **Lemma 21.** *There exists a solution $\mathcal{P}_H^* \subseteq \mathcal{P}_H \cap \mathrm{OPT}_{\mathrm{EF}}$ such that $w(\mathcal{P}_H \cap \mathrm{OPT}_{\mathrm{EF}}) \leq O(w(\mathcal{P}_H^*))$ and*

- *$\mathcal{P}_H^*$ is top-left-packable or bottom-right-packable and for each j we have that $|\mathcal{P}_H^* \cap \mathcal{P}_j| \leq 1$,*
- *or it holds that $|\mathcal{P}_H^*| = 1$.*

Proof sketch. Let L be the line segment connecting p_L with p_R. Essentially, we can assume that the length of the longest edge of each triangle is close to $\sqrt{2}N$ (the length of the diagonal of the knapsack), e.g., $(1-\epsilon)\sqrt{2}N$ for some $\epsilon > 0$. One can show that this holds for all but $O_\epsilon(1)$ hard triangles in OPT. Our key observation is that in *any* packing each hard triangle P_i intersects L by the same amount (up to constant factors). Using this, we partition $\mathrm{OPT}_{\mathrm{EF}}$ into $O(1)$ groups such that in any packing each group intersects L by a smaller amount than $\mathrm{OPT}_{\mathrm{EF}}$ in the original packing. This guarantees that the bottom edge of each triangle fits in the top-left-packing for each group. Using that in the original packing the triangles in $\mathrm{OPT}_{\mathrm{EF}}$ were edge-facing, we show that also the entire triangle fits. ◀

We describe now a polynomial time algorithm that computes the most profitable solution that satisfies the properties of Lemma 21. To find the most profitable solution $\mathcal{P}_H^*$ that satisfies that $|\mathcal{P}_H^*| = 1$ we simply take the triangle with maximum weight, let P_{i^*} be this triangle. We establish now a dynamic program that computes the most profitable top-left-packable solution; computing the most profitable bottom-right-packable solution works analogously. Our DP has a cell corresponding to pairs (j,t) with $j,t \in \mathbb{Z}$. Intuitively, (j,t) represents the subproblem of computing a set $\mathcal{P}'_H \subseteq \mathcal{P}_H$ of maximum weight such that $\mathcal{P}'_H \cap \mathcal{P}_{j'} = \emptyset$ for each $j' < j$ and $|\mathcal{P}'_H \cap \mathcal{P}_{j''}| \leq 1$ for each $j'' \geq j$ and such that $\mathcal{P}'_H$ is top-left-packable inside the triangular area T_t defined by the line that contains p_{TL} and p_t, the top edge of K, and the right edge of K. Given a cell (j,t) we want to compute a solution $DP(j,t)$ associated with (j,t). Intuitively, we guess whether the optimal solution $\mathcal{P}'_H$ to (j,t) contains a triangle from $\mathcal{P}_H \cap \mathcal{P}_j$. Therefore, we try each triangle $P_i \in \mathcal{P}_H \cap \mathcal{P}_j$ and place it inside T_t such that v_i^* concides with p_{TL} and one of its edges lies on the line containing p_{TL} and p_t. Let $t'(P_i)$ denote the smallest integer such that $t'(P_i) \geq t$ and $p_{t'(P_i)}$ is not contained in the resulting placement of P_i inside T_t. We associate with P_i the solution $P_i \cup DP(j+1, t'(P_i))$. Finally, we define $DP(j,t)$ to be the solution of maximum profit among the solutions $P_i \cup DP(j+1, t'(P_i))$ for each $P_i \in \mathcal{P}_H \cap \mathcal{P}_j$ and the solution $DP(j+1,t)$.

We introduce a DP-cell $DP(j,t)$ for each pair $(j,t) \in \mathbb{Z}^2$ where $j_{\min} \leq j \leq j_{\max}$ and $0 \leq t \leq \log_{1+1/n}\left(\frac{N}{2}\right)$. Note that due to Lemma 18 for all other values of j we have that $\mathcal{P}_j \cap \mathcal{P}_H = \emptyset$. Also note that $p_t \notin K$ if $t \geq N^2/2$. This yields at most $(nN)^{O(1)}$ cells in total. Finally, we output the solution $DP(j_{\min}, 0)$.

In the next lemma we prove that our DP computes the optimal top-left-packable solution with the properties of Lemma 21.

▶ **Lemma 22.** *There is an algorithm with a running time of $(nN)^{O(1)}$ that computes the optimal solution $\mathcal{P}' \subseteq \mathcal{P}_H$ such that $\mathcal{P}'$ is top-left-packable or bottom-right-packable and such that for each j we have that $|\mathcal{P}' \cap \mathcal{P}_j| \leq 1$.*

We execute the above DP and its counterpart for bottom-right-packable solutions to obtain a top-left-packable solution $\mathcal{P}'_1$ and a bottom-right-packable solution $\mathcal{P}'_2$. We output the most profitable solution among $\{P_{i^*}\}, \mathcal{P}'_1, \mathcal{P}'_2$. Due to Lemma 21 this yields a solution with weight at least $\Omega(w(\mathcal{P}_H \cap \mathrm{OPT}))$.

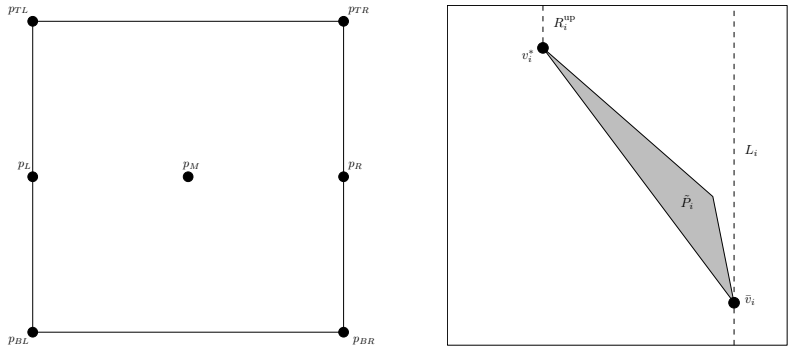


Figure 3 Left: The points $p_{TL}, p_{TR}, p_{BL}, p_{BR}, p_L, p_M, p_R$. Right: A corner-facing triangle, its vertices v_i^* and $\bar{v}_i$, and the lines L_i and R_i^{up}.

▶ **Lemma 23.** *There is an algorithm with a running time of $(nN)^{O(1)}$ that computes a solution $\mathcal{P}'_H \subseteq \mathcal{P}_H$ such that $w(\text{OPT}_{\text{EF}}) \leq O(w(\mathcal{P}'_H))$.*

Corner-facing triangles

We present now a $O(1)$-approximation algorithm for the corner-facing triangles in OPT, i.e., our algorithm computes a solution $\mathcal{P}' \subseteq \mathcal{P}$ of profit at least $\Omega(w(\text{OPT}_{\text{CF}}))$. We first establish some properties for OPT_{CF}. We argue that by losing a constant factor we can assume that each triangle in OPT_{CF} intuitively faces the bottom-right corner.

▶ **Lemma 24.** *By losing a factor 4 we can assume that for each triangle $P_i \in \text{OPT}_{\text{CF}}$ we have that $R_i^{(1)} \setminus \{v_i^*\}$ intersects the bottom edge of the knapsack and and $R_i^{(2)} \setminus \{v_i^*\}$ intersects the right edge of the knapsack, or vice versa.*

In the following lemma we establish a property that will be crucial for our algorithm. For each $P_i \in \text{OPT}_{\text{CF}}$ let R_i^{up} denote the ray originating at v_i^* and going upwards. We establish that we can assume that R_i^{up} does not intersect with any triangle $P_{i'} \in \text{OPT}_{\text{CF}}$, see Figure 3.

▶ **Lemma 25.** *By losing a factor $O(1)$ we can assume that for each $P_i, P_{i'} \in \text{OPT}_{\text{CF}}$ it holds that $R_i^{\text{up}} \cap P_{i'} = \emptyset$.*

Our algorithm is a dynamic program that intuively guesses the placements of the triangles in OPT_{CF} step by step. To this end, each DP-cell corresponds to a subproblem that is defined via a part $K' \subseteq K$ of the knapsack and a subset of the groups $J \subseteq \{j_{\min}, ..., j_{\max}\}$. The goal is to place triangles from $\bigcup_{j \in J} \mathcal{P}_j$ of maximum profit into K'. Formally, each DP-cell is defined by up to two triangles $P_i, P_{i'}$, placements $\tilde{P}_i, \tilde{P}_{i'}$ for them, and a set $J \subseteq \{j_{\min}, ..., j_{\max}\}$; if the cell is defined via exactly one triangle P_i then there is also a value $\text{dir} \in \{\text{left}, \text{mid}\}$. The corresponding region K' is defined as follows: if the cell is defined via zero triangles then the region is the whole knapsack K. Otherwise, let $\bar{v}_i$ denote the right-most vertex of $\tilde{P}_i$, i.e., the vertex of $\tilde{P}_i$ that is closest to the right edge of the knapsack (see Figure 3). Let L_i denote the vertical line that goes through $\bar{v}_i$ (and thus intersects the top and the bottom edge of the knapsack). If the cell is defined via one triangle P_i then observe that $K \setminus (\tilde{P}_i \cup R_i^{\text{up}} \cup L_i)$ has three connected components,

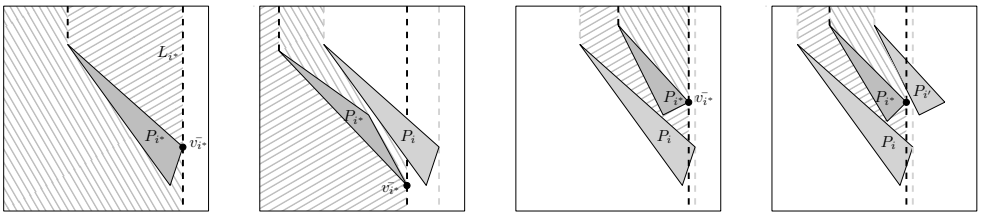


Figure 4 The cases in the transition of the DP for corner-facing triangles (see Lemma 26).

- one on the left, surrounded by R_i^{up}, parts of $\tilde{P}_i$, the left edge of the knapsack, and parts of the top and bottom edge of the knapsack
- one on the right, surrounded by L_i, the right edge of the knapsack, and parts of the top and bottom edge of the knapsack, and
- one in the middle, surrounded by the top edge of the knapsack, $\tilde{P}_i$, and L_i.

If dir = left then the region of the cell equals the left component, if dir = mid then the region of the cell equals the middle component. Assume now that the cell is defined via two triangles $P_i, P_{i'}$. Assume w.l.o.g. that $\bar{v}_i$ is closer to the right edge of the knapsack than $\bar{v}_{i'}$. Then $K \setminus (\tilde{P}_i \cup \tilde{P}_{i'} \cup R_i^{\text{up}} \cup R_{i'}^{\text{up}} \cup L_{i'})$ has one connected component that is surrounded by $\tilde{P}_i, \tilde{P}_{i'}, R_i^{\text{up}}, R_{i'}^{\text{up}}, L_{i'}$ and we define the region of the cell to be this component. Observe that the total number of DP-cells is bounded by $(nN)^{O(1)}$, using that there are only $(nN)^{O(1)}$ possible placements for each triangle.

We describe now a dynamic program that computes the optimal solution to each cell. Assume that we are given a cell C for which we want to compute the optimal solution. We guess the triangle P_{i^*} in the optimal solution to this cell such that $\bar{v}_{i^*}$ is closest to the right edge of the knapsack, and its placement $\tilde{P}_{i^*}$ in the optimal solution to C. Let j^* such that $P_{i^*} \in \mathcal{P}_{j^*}$. We will prove in the next lemma that the optimal solution to C consists of P_{i^*} and the optimal solutions to two other DP-cells, see Figure 4.

▶ **Lemma 26.** *Let C be a DP-cell, let $J \subseteq \{j_{\min}, ..., j_{\max}\}$, and let $P_i \in \mathcal{P}_\ell, P_{i'} \in \mathcal{P}_{\ell'}$ be two triangles with $\ell < \ell'$ and let $\tilde{P}_i, \tilde{P}_{i'}$ be placements for them. Then there are disjoint sets $J', J'' \subseteq J$ such that*

1. *if $C = (J)$, then its optimal solution consists of P_{i^*} and the optimal solutions to the cells $(J', P_{i^*}, \tilde{P}_{i^*}, \text{left})$ and $(J'', P_{i^*}, \tilde{P}_{i^*}, \text{mid})$,*
2. *if $C = (J, P_i, \tilde{P}_i, \text{left})$ then its optimal solution consists of P_{i^*} and the optimal solutions to the cells $(J', P_{i^*}, \tilde{P}_{i^*}, \text{left})$ and $(J'', P_i, \tilde{P}_i, P_{i^*}, \tilde{P}_{i^*})$,*
3. *if $C = (J, P_i, \tilde{P}_i, \text{mid})$ then its optimal solution consists of P_{i^*} and the optimal solutions to the cells $(J', P_{i^*}, \tilde{P}_{i^*}, \text{mid})$ and $(J'', P_i, \tilde{P}_i, P_{i^*}, \tilde{P}_{i^*})$,*
4. *if $C = (J, P_i, \tilde{P}_i, P_{i'}, \tilde{P}_{i'})$ then the optimal solution to C consists of P_{i^*} and the optimal solutions to the cells $(J', P_i, \tilde{P}_i, P_{i^*}, \tilde{P}_{i^*})$ and $(J'', P_{i^*}, \tilde{P}_{i^*}, P_{i'}, \tilde{P}_{i'})$.*

We guess the sets $J', J'' \subseteq J$ according to Lemma 26 and store in C the solution consisting of P_{i^*}, and the solutions stored in the two cells according to the lemma. At the end, the cell $C = (\{j_{\min}, ..., j_{\max}\})$ (whose corresponding region equals to K) contains the optimal solution. By combining Lemmas 23 and 27 we obtain the proof of Lemma 9.

▶ **Lemma 27.** *There is an algorithm with a running time of $(nN)^{O(1)}$ that computes a solution $\mathcal{P}' \subseteq \mathcal{P}$ such that $w(\text{OPT}_{\text{CF}}) \leq O(w(\mathcal{P}'))$.*

2.4 Hard polygons under resource augmentation

Let $\delta > 0$. We consider the setting of $(1+\delta)$-resource augmentation, i.e., we want to compute a solution $\mathcal{P}' \subseteq \mathcal{P}$ that is feasible for a knapsack of size $(1+\delta)N \times (1+\delta)N$ and such that $w(\mathrm{OPT}) \leq O(w(\mathcal{P}'))$ where OPT is the optimal solution for the original knapsack of size $N \times N$. Note that increasing K by a factor of $1+\delta$ is equivalent to shrinking the input polygons by a factor of $1+\delta$.

Given a polygon P defined via coordinates $(x_1, y_1), ..., (x_k, y_k) \in \mathbb{R}^2$ we define $\mathrm{shr}_{1+\delta}(P)$ to be the polygon with coordinates $(\bar{x}_1, \bar{y}_1), ..., (\bar{x}_k, \bar{y}_k) \in \mathbb{R}^2$ where $\bar{x}_{k'} = x_{k'}/(1+\delta)$ and $\bar{y}_{k'} = y_{k'}/(1+\delta)$ for each k'. For each input polygon $P_i \in \mathcal{P}$ we define its shrunk counterpart to be $\bar{P}_i := \mathrm{shr}_{1+\delta}(P_i)$. Based on $\bar{\mathcal{P}}$ we define sets $\bar{\mathcal{P}}_E, \bar{\mathcal{P}}_M, \bar{\mathcal{P}}_H$ and the set $\bar{\mathcal{P}}_j$ for each $j \in \mathbb{Z}$ in the same way as we defined $\mathcal{P}_E, \mathcal{P}_M, \mathcal{P}_H$ and $\mathcal{P}_j$ based on $\mathcal{P}$ above.

For the sets $\bar{\mathcal{P}}_E$ and $\bar{\mathcal{P}}_M$ we use the algorithms due to Lemmas 6 and 7 as before. For the hard polygons $\bar{\mathcal{P}}_H$ we can show that there are only $O_\delta(1)$ groups $\bar{\mathcal{P}}_j$ that are non-empty, using that we obtained them via shrinking the original input polygons. Intuitively, this is true since $\bar{\ell}_i \leq \frac{\sqrt{2}N}{1+\delta}$ for each $\bar{P}_i \in \bar{\mathcal{P}}$ where $\bar{\ell}_i$ denotes the length of the longest diagonal of $\bar{P}_i$, and hence $\bar{\mathcal{P}}_j \cap \bar{\mathcal{P}}_H = \emptyset$ if $j < \log\left(\frac{\delta}{1+\delta}\sqrt{2}N\right)$.

▶ **Lemma 28.** *We have that $\bar{\mathcal{P}}_j = \emptyset$ if $j < \log\left(\frac{\delta}{1+\delta}\sqrt{2}N\right)$. Hence, there are only $\log\left(\frac{1+\delta}{\delta}\right)+1$ values $j \in \mathbb{Z}$ such that $\bar{\mathcal{P}}_j \neq \emptyset$.*

Lemmas 18 and 28 imply that $|\overline{\mathrm{OPT}} \cap \bar{\mathcal{P}}_H| \leq O(\log\left(\frac{1+\delta}{\delta}\right))$ where $\overline{\mathrm{OPT}}$ denotes the optimal solution for the polygons in $\bar{\mathcal{P}}$. Let $\bar{\mathcal{P}}'_H \subseteq \bar{\mathcal{P}}_H$ denote the set due to Lemma 19 when assuming that $\bar{\mathcal{P}}_H$ are the hard polygons in the given instance. Therefore, we guess $\bar{\mathcal{P}}'_H$ in time $n^{O(\log\left(\frac{1+\delta}{\delta}\right))}$. Finally, we output the solution of largest weight among $\bar{\mathcal{P}}'_H$ and the solutions due applying to Lemmas 6 and 7 to the input sets $\bar{\mathcal{P}}_E$ and $\bar{\mathcal{P}}_M$, respectively. This yields the proof of Theorem 3.

3 Optimal profit under resource augmentation

In this section we also study the setting of $(1+\delta)$-resource augmentation, i.e., we want to compute a solution $\mathcal{P}'$ which is feasible for an enlarged knapsack of size $(1+\delta)N \times (1+\delta)N$, for any constant $\delta > 0$. We present an algorithm with a running time of $n^{(\log(n)/\delta)^{O(1)}}$ that computes such a solution $\mathcal{P}'$ with $w(\mathcal{P}') \geq \mathrm{OPT}$ where OPT is the optimal solution for the original knapsack of size $N \times N$. In particular, we here do not lose any factor in our approximation guarantee.

First, we prove a set of properties that we can assume "by $(1+\delta)$-resource augmentation" meaning that if we increase the size of K by a factor $1+\delta$ then there exists a solution of weight $w(\mathrm{OPT})$ with the mentioned properties, or that we can modify the input in time $n^{O(1)}$ such that it has these properties and there still exists a solution of weight $w(\mathrm{OPT})$.

3.1 Few types of items

We want to establish that the input polygons have only $(\log(n)/\delta)^{O(1)}$ different shapes. Like in Section 2 for each polygon $P_i \in \mathcal{P}$ denote by B_i its bounding box with width ℓ_i and height h_i. Note that $\ell_i \geq h_i$. The bounding boxes of all polygons $P_i \in \mathcal{P}$ such that $h_i \leq \delta\frac{N}{n}$ have a total height of at most δN. Therefore, we can pack all these polygons into the extra capacity that we gain by increasing the size of K by a factor $1+\delta$ and therefore ignore them in the sequel.

▶ **Lemma 29.** *By $(1+\delta)$-resource augmentation we can assume for each $P_i \in \mathcal{P}$ that $\ell_i \geq h_i \geq \delta N/n$ and that* $\text{area}(P_i) = \Omega(\text{area}(K)\delta^2/n^2)$.

Next, intuitively we stretch the optimal solution OPT by a factor $1+\delta$ which yields a container C_i for each polygon $P_i \in \text{OPT}$ which contains P_i and which is slightly bigger than P_i. We define a polygon P_i' such that $P_i \subseteq P_i' \subseteq C_i$ and that globally there are only $(\log(n)/\delta)^{O_\delta(1)}$ different ways P_i' can look like, up to translations and rotations. We refer to those as a set $\mathcal{S}$ of *shapes* of input objects. Hence, due to the resource augmentation we can replace each input polygon P_i by one of the shapes in $\mathcal{S}$.

▶ **Lemma 30.** *By $(1+\delta)$-resource augmentation we can assume that there is a set of shapes $\mathcal{S}$ with $|\mathcal{S}| \leq (\log(n)/\delta)^{O_\delta(1)}$ such that for each $P_i \in \mathcal{P}$ there is a shape $S \in \mathcal{S}$ such that $P_i = S$ and S has only $\Lambda = (1/\delta)^{O(1)}$ many vertices.*

Finally, we ensure that for each polygon $P_i \in \mathcal{P}$ we can restrict ourselves to only $(n/\delta)^{O(1)}$ possible placements in K.

▶ **Lemma 31.** *By $(1+\delta)$-resource augmentation, for each polygon $P_i \in \mathcal{P}$ we can compute a set $\mathcal{L}_i$ of at most $(n/\delta)^{O(1)}$ possible placements for P_i in time $(n/\delta)^{O(1)}$ such that if $P_i \in \text{OPT}$ then in* OPT *the polygon P_i is placed inside K according to one placement $\tilde{P}_i \in \mathcal{L}_i$.*

3.2 Recursive algorithm

We describe our main algorithm. First, we guess how many polygons of each of the shapes in $\mathcal{S}$ are contained in OPT. Since there are only $(\log(n)/\delta)^{O_\delta(1)}$ different shapes in $\mathcal{S}$ we can do this in time $n^{(\log(n)/\delta)^{O_\delta(1)}}$. Once we know how many polygons of each shape we need to select, it is clear which polygons we should take since if for some shape $S_i \in \mathcal{S}$ we need to select n_i polygons with that shape then it is optimal to select the n_i polygons in $\mathcal{P}$ of shape S_i with largest weight. Therefore, in the sequel assume that we are given a set of polygons $\mathcal{P}' \subseteq \mathcal{P}$ and we want to find a packing for them inside K.

Our algorithm is recursive and it generalizes a similar algorithm for the special case of axis-parallel rectangles in [1]. On a high level, we guess a partition of K given by a separator Γ which is a polygon inside K. It has the property that at most $\frac{2}{3}|\text{OPT}|$ of the polygons of OPT lie inside Γ and at most $\frac{2}{3}|\text{OPT}|$ of the polygons of OPT lie outside Γ. We guess how many polygons of each shape are placed inside and outside Γ in OPT. Then we recurse separately inside and outside Γ. For our partition, we are looking for a polygon Γ according to the following definition.

▶ **Definition 32.** *Let $\ell \in \mathbb{N}$ and $\epsilon > 0$. Let $\bar{\mathcal{P}}$ be a set of pairwise non-overlapping polygons in K. A polygon Γ is a* balanced $\hat{\epsilon}$-cheap ℓ-cut *if*

- *Γ has at most ℓ edges,*
- *the polygons contained in Γ have a total area of at most $2/3 \cdot \text{area}(\bar{\mathcal{P}})$,*
- *the polygons contained in the complement of Γ, i.e., in $K \setminus \Gamma$, have a total area of at most $2/3 \cdot \text{area}(\bar{\mathcal{P}})$, and*
- *the polygons intersecting the boundary of Γ have a total area of at most $\hat{\epsilon} \cdot \text{area}(\bar{\mathcal{P}})$.*

In order to restrict the set of balanced cheap cuts to consider, we will allow only polygons Γ such that each of its vertices is contained in a set Q of size $(n/\delta)^{O(1)}$ defined as follows. We fix a triangulation for each placement $P_i' \in \mathcal{L}_i$ of each polygon $P_i \in \mathcal{P}'$. We define a set Q_0 where for each placement $P_i' \in \mathcal{L}_i$ for P_i we add to Q_0 the positions of the vertices of P_i'. Also, we add the four corners of K to Q_0. Let $\mathcal{V}$ denote the set of vertical lines $\{(\bar{x}, \bar{y}) | \bar{y} \in \mathbb{R}\}$ such that $\bar{x}$ is the x-coordinate of one point in Q_0. We define a set Q_1 where

for each placement $P_i' \in \mathcal{L}_i$ of each $P_i \in \mathcal{P}'$, each edge e of a triangle in the triangulation of P_i', and each vertical line $L \in \mathcal{V}$ we add to Q_1 the intersection of e and L. Also, we add to Q_1 the intersection of each line in $L \in \mathcal{V}$ with the two boundary edges of K. Let Q_2 denote the set of all intersections of pairs of line segments whose respective endpoints are in $Q_0 \cup Q_1$. We define $Q := Q_0 \cup Q_1 \cup Q_2$. A result in [1] implies that there exists a balanced cheap cut whose vertices are all contained in Q.

▶ **Lemma 33** ([1]). *Let $\epsilon > 0$ and let $\mathcal{P}'$ be a set of pairwise non-intersecting polygons in the plane with at most Λ edges each such that* $\text{area}(P) < \text{area}(\mathcal{P}')/3$ *for each $P \in \mathcal{P}$. Then there exists a balanced $O(\epsilon\Lambda)$-cheap $\Lambda(\frac{1}{\epsilon})^{O(1)}$-cut Γ whose vertices are contained in Q.*

Our algorithm is recursive and places polygons from $\mathcal{P}'$, trying to maximize the total area of the placed polygons. In each recursive call we are given an area $\bar{K} \subseteq K$ and a set of polygons $\bar{\mathcal{P}} \subseteq \mathcal{P}'$. In the main call these parameters are $\bar{K} = K$ and $\bar{\mathcal{P}} = \mathcal{P}'$. If $\bar{\mathcal{P}} = \emptyset$ then we return an empty solution. If there is a polygon $P_i \in \bar{\mathcal{P}}$ with $\text{area}(P_i) \geq \text{area}(\bar{\mathcal{P}})/3$ then we guess a placement $P_i' \in \mathcal{L}_i$ and we recurse on the area $K \setminus P_i'$ and on the set $\bar{\mathcal{P}} \setminus \{P_i\}$. Otherwise, we guess the balanced cheap cut Γ due to Lemma 33 with $\epsilon := \frac{\delta}{\Lambda \log(n/\delta)}$ and for each shape $S \in \mathcal{S}$ we guess how many polygons of $\mathcal{P}'$ with shape S are contained in $\Gamma \cap \bar{K}$, how many are contained in $\bar{K} \setminus \Gamma$, and how many cross the boundary of Γ (i.e., have non-empty intersection with the boundary of Γ). Note that there are only $n^{(\Lambda \log(n/\delta))^{O(1)}}$ possibilities to enumerate. Let $\bar{\mathcal{P}}_{\text{in}}, \bar{\mathcal{P}}_{\text{out}}$, and $\bar{\mathcal{P}}_{\text{cross}}$ denote the respective sets of polygons. Then we recurse on the area $\Gamma \cap \bar{K}$ with input polygons $\bar{\mathcal{P}}_{\text{in}}$ and on the area $\bar{K} \setminus \Gamma$ with input polygons $\bar{\mathcal{P}}_{\text{out}}$. Suppose that the recursive calls return two sets of polygons $\bar{\mathcal{P}}'_{\text{in}} \subseteq \bar{\mathcal{P}}_{\text{in}}$ and $\bar{\mathcal{P}}'_{\text{out}} \subseteq \bar{\mathcal{P}}_{\text{out}}$ that the algorithm managed to place inside the respective areas $\Gamma \cap \bar{K}$ and $\bar{K} \setminus \Gamma$. Then we return the set $\bar{\mathcal{P}}'_{\text{in}} \cup \bar{\mathcal{P}}'_{\text{out}}$ for the guesses of Γ, $\bar{\mathcal{P}}_{\text{in}}, \bar{\mathcal{P}}_{\text{out}}$, and $\bar{\mathcal{P}}_{\text{cross}}$ that maximize $\text{area}(\bar{\mathcal{P}}'_{\text{in}} \cup \bar{\mathcal{P}}'_{\text{out}})$. If we guess the (correct) balanced cheap cut due to Lemma 33 in each iteration then our recursion depth is $O(\log_{3/2}(n^2/\delta^2)) = O(\log(n/\delta))$ since the cuts are balanced and each polygon has an area of at least $\Omega(\text{area}(K)\delta^2/n^2)$ (see Lemma 29). Therefore, if in a recursive call of the algorithm the recursion depth is larger than $O(\log(n/\delta))$ then we return the empty set and do not recurse further. Also, if we guess the correct cut in each node of the recursion tree then we cut polygons whose total area is at most a $\frac{\delta}{\log(n/\delta)}$-fraction of the area of all remaining polygons. Since our recursion depth is $O(\log(n/\delta))$, our algorithm outputs a packing for a set of polygons in $\mathcal{P}'$ with area at least $(1 - \frac{\delta}{\log(n/\delta)})^{O(\log(n/\delta))} \bar{w}(\bar{\mathcal{P}}) = (1 - O(\delta))\text{area}(\bar{\mathcal{P}})$. This implies the following lemma.

▶ **Lemma 34.** *Assume that there is a non-overlapping packing for $\mathcal{P}'$ in K. There is an algorithm with a running time of $n^{(\Lambda \log(n/\delta))^{O(1)}}$ that computes a placement of a set of polygons $\bar{\mathcal{P}}' \subseteq \mathcal{P}'$ inside K such that* $\text{area}(\bar{\mathcal{P}}') \geq (1 - O(\delta))\text{area}(\mathcal{P}')$.

It remains to pack the polygons in $\tilde{\mathcal{P}}' := \mathcal{P}' \setminus \bar{\mathcal{P}}'$. The total area of their bounding boxes is bounded by $\sum_{P_i \in \tilde{\mathcal{P}}'} B_i \leq 2\text{area}(\tilde{\mathcal{P}}') \leq O(\delta)\text{area}(\mathcal{P}') \leq O(\delta)\text{area}(K)$. Therefore, we can pack them into additional space that we gain via increasing the size of K by a factor $1 + O(\delta)$, using the Next-Fit-Decreasing-Height algorithm [4].

▶ **Theorem 35.** *There is an algorithm with a running time of $n^{(\log(n)/\delta))^{O(1)}}$ that computes a set $\mathcal{P}'$ with $w(\mathcal{P}') \geq \text{OPT}$ such that $\mathcal{P}'$ fits into K under $(1 + \delta)$-resource augmentation.*

References

1 Anna Adamaszek and Andreas Wiese. A QPTAS for maximum weight independent set of polygons with polylogarithmically many vertices. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 645–656. SIAM, 2014.

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

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

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

5 Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via l-packings. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 260–271, 2017. doi:10.1109/FOCS.2017.32.

6 Sandy Heydrich and Andreas Wiese. Faster approximation schemes for the two-dimensional knapsack problem. *ACM Trans. Algorithms*, 15(4):47:1–47:28, 2019. doi:10.1145/3338512.

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

8 Klaus Jansen and Roberto Solis-Oba. A polynomial time approximation scheme for the square packing problem. In *Integer Programming and Combinatorial Optimization (IPCO 2008)*, volume 5035 of *LNCS*, pages 184–198. Springer, 2008.

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

10 Klaus Jansen and Guochuan Zhang. On rectangle packing: maximizing benefits. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, pages 204–213. SIAM, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982822.

11 Joseph YT Leung, Tommy W Tam, CS Wong, Gilbert H Young, and Francis YL Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990.

12 Carla Negri Lintzmayer, Flávio Keidi Miyazawa, and Eduardo Candido Xavier. Two-dimensional knapsack for circles. In *Latin American Symposium on Theoretical Informatics*, pages 741–754. Springer, 2018.

13 A Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.