A Polynomial Time Approximation Scheme for the Square Packing Problem

Klaus Jansen[1,*] and Roberto Solis-Oba[2,**]

[1] Institut für Informatik
Universität zu Kiel, Kiel, Germany
kj@informatik.uni-kiel.de
[2] Department of Computer Science
The University of Western Ontario, London, Canada
solis@csd.uwo.ca

Abstract. Given a set Q of squares with positive profits, the *square packing problem* is to select and pack a subset of squares of maximum profit into a rectangular bin $\mathcal{R}$. We present a polynomial time approximation scheme for this problem, that for any value $\epsilon > 0$ finds and packs a subset $Q' \subseteq Q$ of profit at least $(1-\epsilon)OPT$, where OPT is the profit of an optimum solution. This settles the approximability of the problem and improves on the previously best approximation ratio of $5/4+\epsilon$ achieved by Harren's algorithm.

1 Introduction

Let $Q = \{Q_1, Q_2, \ldots, Q_n\}$ be a set of squares with positive profits. For a square Q_i, its size s_i is the length of one of its sides, and its profit is denoted as p_i. The *square packing problem* is to select and pack a maximum profit subset of squares into a given rectangular bin $\mathcal{R}$. We consider only orthogonal packings, i.e. packings in which the sides of the squares are parallel to the sides of the bin. Geometric packing problems, like this one, have received a lot of attention recently from the algorithms community [1,6,8], as these problems have numerous applications in stock cutting, VLSI design, advertisement placement, image processing, and scheduling [5,6,8]. Furthermore, some of these problems are fundamental geometric problems that have been extensively studied by the Discrete Geometry community [7,11].

The square packing problem is known to be strongly NP-hard even for the restricted case of packing squares with unit profits [10]. For the version of the problem with unit profits, Jansen and Zhang [6] designed a polynomial time approximation scheme (PTAS). Fishkin et al. [4] studied the so-called square packing problem with resource augmentation and designed an algorithm that

* Research supported in part by the EU project AEOLUS contract number 015964 and by the DAAD German Academic Exchange Service.
** Research supported in part by the Natural Science and Engineering Research Council of Canada grant R3050A01.

packs a subset of squares of profit at least $(1-\epsilon)OPT$ into an augmented square bin $[0, 1+\epsilon] \times [0, 1+\epsilon]$, where OPT is the maximum profit of any subset of squares that can be packed into a unit size square bin. They also gave a PTAS for the problem without resource augmentation for the case when $p_i = s_i^2$ for every $Q_i \in Q$. For the square packing problem with arbitrary profits, the previously best algorithm, by Harren, [5] has performance ratio $\frac{5}{4} + \epsilon$, for any $\epsilon > 0$.

We improve upon Harren's algorithm by presenting a PTAS for the square packing problem that selects and packs in $\mathcal{R}$ squares of profit at least $(1-\epsilon)OPT$, where OPT is the optimum profit and $\epsilon > 0$ is any given positive constant. This is the best possible algorithm for the problem in the sense that the square packing problem is strongly NP-hard [10] and, so, it does not admit a fully polynomial time approximation scheme (FPTAS) unless P = NP.

For presentation simplicity, we only describe our algorithm for the case when $\mathcal{R}$ is a unit size square bin. In the full version of the paper we will show how to deal with the case of a rectangular bin. The description is divided in two parts. First, we show a series of transformations that simplify the structure of an optimum solution P^* while only slightly decreasing the total profit of the squares packed in the bin. The transformations are based on the intuitive observation that large and high profit squares must be accurately positioned in the bin so they all fit; however, small, low profit squares might be positioned less carefully, as small mistakes in their positioning only causes a small loss in profit due to some of them not fitting in the bin. This observation has been used in the past for solving other packing and scheduling problems. One crucial point of our work that is different from past research is that we cannot round up the dimensions of large squares, as then we might need to increase the size of the bin to be able to pack the enlarged squares. This makes our problem a lot harder than problems that allow resource augmentation.

We define a class of canonical packings for the relatively small set of high profit squares in a feasible solution; these packings leave free space in the bin that can be split into a small set of rectilinear polygonal regions. We consider three types of these regions: large, elongated, and small. A large region is rectangular and its two dimensions are much larger than the sizes of the squares packed in it, so a simple shelf-packing algorithm, like NFDS [2], can pack squares of nearly optimum profit there. An elongated region is also rectangular, but it has one dimension much larger than the other, so, a strip packing algorithm can be used to pack squares there to near optimality.

Packing squares in a small region is considerably more difficult though, as there might not be enough room to pack any squares crossing the borders of a small region. We handle this situation by allowing some of the regions to be merged together. The drawback of doing this is that these regions get more complex shapes, and squares of lage size relative to the dimensions of these regions must be packed very carefully. We deal with these regions by applying our transformations recursively on them. This creates a hierarchical organization for the regions, and we show that regions belonging to a constant number of levels

at the top of this hierarchy store squares of nearly optimum profit. This allows us to bound the number of required recursive applications of the transformations.

The result of the transformations is a near optimum solution P^+ with a very regular structure: The packing P^+ can be divided into a constant number of rectangular regions; in each region squares are packed either using the greedy NFDS algorithm, or groups of squares of similar sizes are packed in strips of the same width. Showing that a near-optimum packing with this particular structure exists is the core of our work as it allows a simple enumeration-based algorithm to compute a solution of profit close to that of P^+. The transformations are rather complex and they are described in Sections 2-5.

The second part of our algorithm is described in Section 7 and it shows how to build a very large, but polynomial in n, number of possible solutions with the same structure as P^+. We show that one of these solutions has profit very close to the profit of P^+, thus proving the existence of a PTAS for the square packing problem. The main contribution of this work is to settle the approximability of the square packing problem.

2 Big and Small Squares

Let $\epsilon > 0$ be the required precision and, without loss of generality, let $1/\epsilon$ be integer and $\epsilon \leq 1/3$. Fix an instance Q of the square packing problem and an optimum solution P^* for it. Let Q^* be the set of squares selected by P^* and let OPT be the profit of P^*. We can assume that Q^* has more than $1/\epsilon$ squares, as otherwise we could find Q^* in $O(1)$ time by simply enumerating all subsets of at most $1/\epsilon$ squares from Q and then selecting the subset Q' of largest profit that can be packed in the bin using the upper-left justified packing algorithm of Section 7.

As mentioned above, our algorithm deals differently with small and large squares. Thus, we first need to define the notion of "small" and "large". Define $\rho_0 = 1$, and $\rho_k = \rho_{k-1}^4 (\epsilon/4)^{5+2^{\eta+1}}$ for all integers $k \geq 1$, where $\eta = \log_{1-\epsilon} \epsilon$. Then, it is not hard to see that $\rho_k = (\epsilon/4)^{(5+2^{\eta+1})(4^k-1)/3}$ for all $k \geq 0$.

For each integer $k \geq 1$ we define $\mathcal{Q}_k = \{Q_i \in Q \mid s_i \in (\rho_k, \rho_{k-1}]\}$. Let τ be the smallest index such that the total profit of the squares in $\mathcal{Q}_\tau^* = \mathcal{Q}_\tau \cap Q^*$ is at most ϵOPT. Observe that $\tau \leq 1/\epsilon$.

Partition Q into three groups: the *big* squares $\mathcal{B} = \{Q_i \in Q \mid s_i > \rho_{\tau-1}\}$, the *medium* squares $\mathcal{M} = \{Q_i \in Q \mid \rho_\tau < s_i \leq \rho_{\tau-1}\}$, and the *small* squares $\mathcal{S} = \{Q_i \in Q \mid s_i \leq \rho_\tau\}$. Similarly, define $\mathcal{B}^* = \mathcal{B} \cap Q^*$, $\mathcal{M}^* = \mathcal{M} \cap Q^*$, and $\mathcal{S}^* = \mathcal{S} \cap Q^*$. Note that $\mathcal{M}^* = \mathcal{Q}_\tau^*$, so the total profit of the medium squares in $\mathcal{M}^*$ is at most ϵOPT. In the next sections we show how to modify P^* to produce a near-optimum solution P^+ with a regular structure.

To construct P^+ we need to consider three cases: (i) when $\mathcal{B}^* = \emptyset$, (ii) when every big square has profit larger than ϵOPT, and (iii) when at least one big square has profit no larger than ϵOPT. Case (ii) is the most complex since, as we will show, it requires us to recursively partition the bin into successively smaller

blocks where squares are re-classified as big, medium, and small depending on their relative sizes with respect to the dimensions of the blocks.

3 Instances without Big Squares

If $\mathcal{B}^* = \emptyset$, then we just remove $\mathcal{M}^*$ from P^* to get a solution P^+ of profit at least $(1-\epsilon)OPT$. Note that all remaining squares have size at most $\rho_\tau \leq (\epsilon/4)^{5+2^{\eta+1}}$, as $\tau \geq 1$. Furthermore, since $\epsilon \leq 1/3$, then $\eta \geq \log_{2/3}(1/3) > 2.7$, and so

$$\rho_\tau = \rho_{\tau-1}^4 (\epsilon/4)^{5+2^{\eta}+1} \leq \rho_{\tau-1}^4 (\epsilon/4)^{5+2^{3.7}} < \epsilon/10^{10}, \tag{1}$$

which is very small compared to the bin. Hence, we can use NFDS [2] to re-pack small squares of profit at least $(1-2\epsilon)OPT$ in the bin.

4 Instances with High Profit Big Squares

We now consider the case when every big square has profit larger than ϵOPT. Note that in this case, $|\mathcal{B}^*| < 1/\epsilon$.

4.1 Blocks

We temporarily remove from P^* all medium and small squares. The empty space created is partitioned by the big squares into a set of disconnected regions $r_1, r_2, \ldots, r_\kappa$. Each r_i is a polygonal region that might contain holes and it is delimited by a set p_i of polygonal boundaries. Let N_i be the number of vertices of p_i.

We divide each r_i into rectangular sub-regions by tracing a horizontal cutting line passing through each horizontal side of p_i and a vertical cutting line through each vertical side of p_i. These cutting lines define a grid of size at most $N_i/2 \times N_i/2$ that splits r_i into $O(N_i^2)$ rectangular regions called *blocks* (see Fig. 1).

Now we add back the medium and small squares in the same positions where they appear in the optimum solution P^*. Note that some of these squares might cross the block boundaries; we wish to simplify the packing so no square crosses any block boundaries. To achieve this, some squares might need to be re-arranged and/or discarded. The exact procedure for eliminating intersections between squares and block boundaries depends on the dimensions and shape of each block. Accordingly, we partition the blocks in three classes as follows. For each block b_j let $\widehat{Q}_j$ be the largest square whose interior intersects b_j and let $\hat{s}_j$ be the size of $\widehat{Q}_j$. Note that $\widehat{Q}_j \in \mathcal{M}^* \cup \mathcal{S}^*$. Let

$$\alpha = 10/\epsilon \text{ and } \beta = 24/\epsilon^3. \tag{2}$$

Definition 1.

- *A block b_j is* **large** *if it has length and width at least $\alpha \hat{s}_j$.*

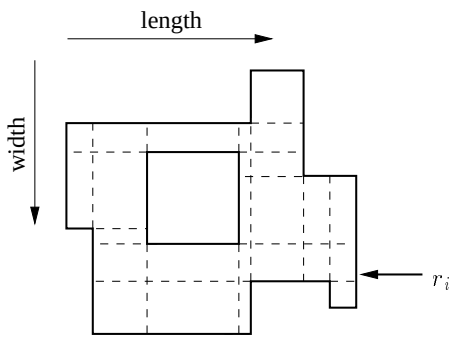

Fig. 1. Grid splitting region r_i into blocks.

- *A block b_j is* **elongated** *if either its length is at least $\beta\hat{s}_j$ and its width is smaller than $\alpha\hat{s}_j$ (***horizontal elongated block***), or its length is smaller than $\alpha\hat{s}_j$ and its width is at least $\beta\hat{s}_j$ (***vertical elongated block***).*
- *A block b_j is* **small** *if it is neither long nor elongated.*

To modify P^* so that no square crosses any block boundaries, first we need to eliminate all *bad block boundaries* (see Fig. 2). The common boundary I between two blocks b_i and b_j is *bad* if there is some square Q_r crossing I and either

1. I is common to the long sides of two elongated blocks, or
2. I is common to a small block and to a long side of an elongated block, or
3. I is common to two small blocks.

Intuitively, bad block boundaries might be problematic because if a square Q_r crosses the long side of a small or elongated block b_j, Q_r cannot be packed in b_j if it is larger than the smaller dimension of b_j.

We eliminate bad block boundaries by merging some of the blocks as follows. Consider one by one the columns of the grid splitting r_i (see Fig. 1). If some column has two adjacent horizontal elongated blocks sharing a bad boundary, these blocks are merged (see Fig. 2(a)). The resulting block is either horizontal elongated or large. This process eliminates all bad boundaries between horizontal elongated blocks. Next, we consider one by one the rows of the grid. If in some row two adjacent vertical elongated blocks share a bad boundary, they are merged (see Fig. 2(b)). The resulting block is either vertical elongated or large.

After merging blocks as described, the only bad boundaries remaining might be between two small blocks, or between a small and an elongated block. If a group of blocks share bad boundaries, they are all merged into a *composite small block* (see Fig. 2(c)). For convenience, a small block is also composite small.

Lemma 1. *A region r_i can be split into at most $N_i^2/4$ blocks avoiding bad block boundaries. Each composite small block b_j has area at most $\alpha\beta(\hat{s}_j N_i/2)^2$, where $\hat{s}_j$ is the size of the largest square completely packed inside b_j; furthermore, the set p_j of polygonal boundaries delimiting b_j has at most $N_i^2/4$ vertices.*

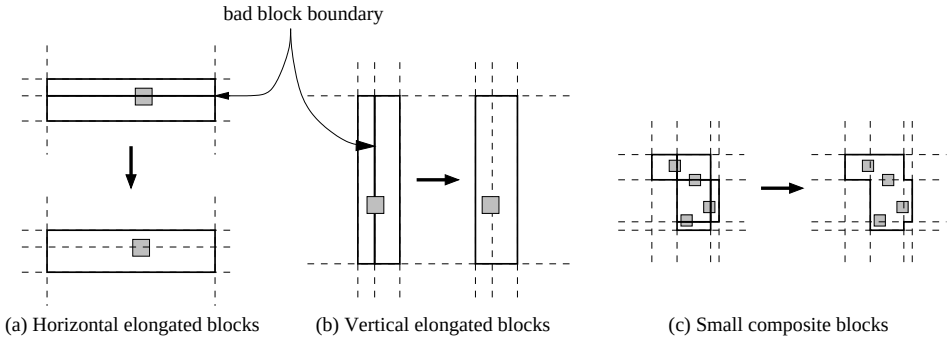

Fig. 2. Merging blocks.

4.2 Assigning Squares to Blocks

Consider a region r_i and let C_i be the set of squares that cross the boundaries of r_i's blocks. Each square completely contained in a block is assigned to that block. As for the squares in C_i, we assign them to blocks as follows.

1. A square in C_i intersecting a large block b_j is assigned to b_j; if a square intersects several large blocks, it is assigned to any one of them.
2. For the remaining squares in C_i, if a square Q_r intersects several elongated blocks, it is assigned to a block that intersects Q_r with its short side only; we can guarantee that this block exists because of the way in which blocks have been merged.

Note that every square in C_i is allocated to a large or elongated block, as no square crosses the common boundary of two small blocks (otherwise they would have been merged). We now modify P^* so that no square crosses any block boundaries.

4.3 Large Blocks

Consider a large block b_j. We only consider the case when the width w_j of b_j is smaller than its length l_j. Let S_j be the set of squares assigned to b_j and let $\widehat{Q}_j$ be the largest square in S_j. The following shifting technique allows us to (i) round the dimensions of b_j down to the nearest multiple of $\hat{s}_j$ (the size of $\widehat{Q}_j$) and (ii) also allows us to remove the squares crossing the boundaries of b_j, while losing only a very small fraction of the total profit of the squares originally packed by P^* in b_j.

Let D_j be the set of squares crossing any of the boundaries of block b_j, plus the squares whose top or right sides are at distance smaller than $\hat{s}_j$ from the top or the right side of b_j (see Fig. 3). D_j is momentarily removed from the packing. This creates an empty region of width at least $\hat{s}_j$ at the top of b_j and an empty

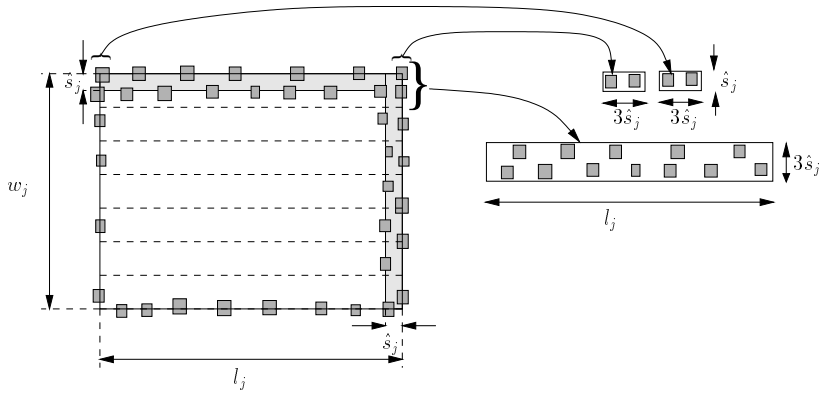

Fig. 3. The shifting technique. Set D_j appears in dark shade.

region of length at least $\hat{s}_j$ at the right side of b_j. These empty regions allow us to round the width and length of b_j down to the nearest multiple of $\hat{s}_j$ without having to remove any more squares from the packing.

Split b_j into $\alpha/10$ horizontal strips of the same width, as shown in Fig. 3. Each square Q_k packed inside b_j is assigned to one of the strips as follows: If Q_k intersects only one strip t, then Q_k is assigned to t; if Q_k intersects two strips t and t', then Q_k is assigned to the strip on the top. Since every strip has width at least $w_j/(\alpha/10) \geq 10\hat{s}_j$, by Def. 1, then, no square intersects three strips.

Let t_j be the strip for which the total profit, $\mathit{profit}(t_j)$, of the squares assigned to it is minimum. Clearly, $\mathit{profit}(t_j) \leq \frac{10}{\alpha}\mathit{profit}(S_j) \leq \epsilon \times \mathit{profit}(S_j)$, by (2). Hence, if we remove from P^* all squares assigned to t_j the profit loss will be very small. Furthermore, this creates inside t_j an empty region of width at least $9\hat{s}_j$. We use this empty space to reintroduce the previously removed squares D_j:

- All squares in D_j that were taken off the top of b_j can be packed in a strip of length $l_j + 2\hat{s}_j$ and width $3\hat{s}_j$ (see Fig. 3). These squares can be re-arranged so they fit in two strips: one of length l_j and width $3\hat{s}_j$ and the other of length $6\hat{s}_j$ and width $\hat{s}_j$ (see right hand side of Fig. 3).
- All remaining squares in D_j that were removed from the right side of b_j can be packed in a strip of length $3\hat{s}_j$ and width $w_j < l_j$ (see Fig. 3).
- The other squares that crossed the bottom of b_j can be packed in a strip of length l_j and width $\hat{s}_j$, while the squares that crossed the left side of b_j can be packed in a strip of length $w_j < l_j$ and width $\hat{s}_j$.

The total profit of the squares that remain in b_j is at least $(1-\epsilon)\mathit{profit}(S_j)$. Since every square packed in b_j has size at most $\hat{s}_j$ (which is very small compared to the dimensions of b_j), we can use the NFDS algorithm to produce a simple-structured packing for b_j. The total profit of the squares that the NFDS algorithm can pack in b_j is at least $(1-2\epsilon)\mathit{profit}(S_j)$.

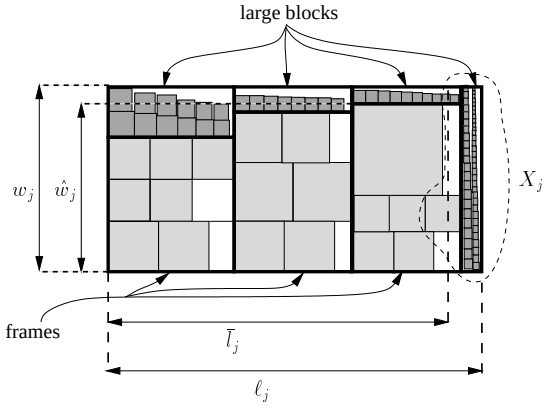

Fig. 4. Packing produced by the KR algorithm. Set $S_{\epsilon j}$ appears in light shading.

4.4 Elongated Blocks

To understand how we will modify the structure of the packing for elongated blocks, we first give a brief review of the strip packing algorithm of Kenyon and Rémila [8], which we will refer to as the KR algorithm.

4.4.1 The KR Algorithm. The KR algorithm uses a two-stage process to pack a set S_j of squares in an elongated block b_j. For convenience, we assume that the length l_j of b_j is larger than its width w_j. Let $\widehat{Q}_j$ be the largest square packed in b_j and let $\hat{s}_j$ be its size. In the first stage, the KR algorithm packs in b_j the set $S_{\epsilon j} \subseteq S_j$ of squares of size at least $\epsilon \times w_j/(2+\epsilon)$ using a linear programming approach. To do this, a (multi)subset $T_j \subseteq S_{\epsilon j}$ of $h = \left(\frac{2+\epsilon}{\epsilon}\right)^2$ squares is selected; the width of each square in $S_{\epsilon j}$ is rounded up to the width of the nearest square from T_j. Rounded squares are grouped into h clusters, $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_h$, called *configurations*. Configuration $\mathcal{C}_i$ has width $W_i \leq w_j$. The solution for the linear program determines the length L_i of each $\mathcal{C}_i$.

Each configuration $\mathcal{C}_i$ is allocated a rectangular region R_i of width W_i and length L_i. The rectangular regions $R_1, R_2, \ldots, R_h$ are called *frames*. If configuration $\mathcal{C}_i$ is composed of squares of widths $w_{i1}, w_{i2}, \ldots, w_{ir}$, then frame R_i is divided into r rows of widths $w_{i1}, w_{i2}, \ldots, w_{ir}$. Rectangles from $S_{\epsilon j}$ of width w_{ij} are packed in the row of width w_{ij} as long as their total length does not exceed $L_i + \hat{s}_j$ (see Fig. 4).

In the second stage the space remaining in b_j is divided into $h+1$ rectangular blocks, $R'_1, R'_2, \ldots, R'_h, R'_{h+1}$, that for convenience we call large blocks; R'_i, $1 \leq i \leq h$, has width $w_j - W_i$ and length L_i. Block R'_{h+1} has width w_j and length $L_j - \sum_{i=1}^{h} L_i$. Squares in $S_j \setminus S_{\epsilon j}$ are packed in these blocks with the NFDS algorithm. The packing produced has width no larger than w_j and length [8]

$$\ell_j \leq l_j(1+\epsilon) + (4(2+\epsilon)^2/\epsilon^2 + 1)\hat{s}_j < l_j(1+\epsilon) + 22\hat{s}_j/\epsilon^2, \text{ as } \epsilon \leq 1/3. \quad (3)$$

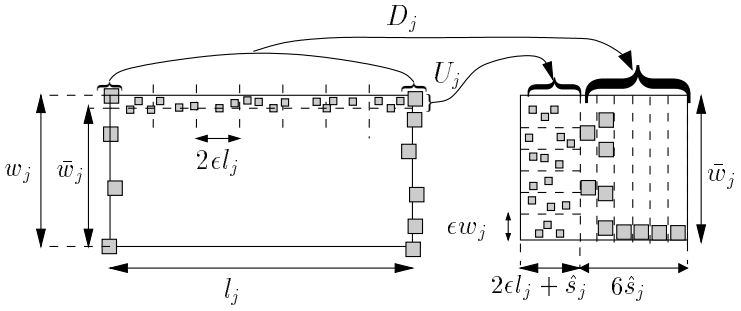

Fig. 5. Re-packing the squares $D_j \cup U_j$.

4.4.2 Modifying the Packing in an Elongated Block. Consider an elongated block b_j. We only look at the case when the length l_j of b_j is larger than its width w_j. Let S_j be the set of squares assigned to b_j and let $\hat{s}_j$ be the size of the largest square in S_j. All squares in S_j have size at most w_j as they are either completely contained inside b_j or they intersect the short side of b_j.

Let D_j be the set of squares crossing the short sides of b_j. These squares are, for the time being, removed from b_j. Let S'_j be the squares remaining in b_j. We re-pack S'_j by using the KR algorithm. In this new packing, let $\hat{w}_j$ be the width of the widest configuration used (see Fig. 4). Let i be the largest integer such that $\hat{w}_j + i\epsilon\hat{s}_j \leq w_j$. We round the width of b_j down to $\bar{w}_j = \max\{\hat{s}_j, \hat{w}_j + i\epsilon\hat{s}_j\}$. By doing this, some subset $U_j \subseteq S'_j$ of squares of size smaller than $\epsilon w_j/(2+\epsilon)$ might not fit at the top of b_j (see Fig. 4; the dark shaded squares have size smaller than $\epsilon w_j/(2+\epsilon)$). The total area of the squares in U_j is smaller than $2\epsilon w_j l_j/(2+\epsilon)$. The squares U_j are removed from b_j; they will be re-introduced to the packing later.

Now, we round the length of b_j down to the nearest multiple $\bar{l}_j$ of $\hat{s}_j$ (see Fig. 4). The KR algorithm might produce a packing for S'_j of length larger than l_j, but we can reduce the length to at most $\bar{l}_j$ by using the shifting technique described in Section 4.3: Let X_j be the set of squares that are not completely packed by the KR algorithm inside the rounded block b_j (see Fig. 4). We divide b_j into $1/(4\epsilon)$ strips of the same length and width $\bar{w}_j$. For at least one of these strips t_j, the total profit of the squares completely packed in it is at most $4\epsilon \times \mathit{profit}(S'_j)$. Remove these squares from t_j creating an empty region of width $\bar{w}_j$ and length at least $4\epsilon l_j - 2\hat{s}_j \geq 3\epsilon l_j + \epsilon\beta\hat{s}_j - 2\hat{s}_j$, since by Def. 1, $l_j \geq \beta\hat{s}_j$.

By (3), X_j can be packed in a rectangle of width $\bar{w}_j$ and length $\epsilon l_j + 22\hat{s}_j/\epsilon^2 + \hat{s}_j$. We can pack X_j in t_j and still have an empty region of width $\bar{w}_j$ and length

$$\ell \geq 3\epsilon l_j + \epsilon\beta\hat{s}_j - 2\hat{s}_j - (\epsilon l_j + 22\hat{s}_j/\epsilon^2 + \hat{s}_j) = 2\epsilon l_j + \hat{s}_j(\frac{2}{\epsilon^2} - 3), \text{ by (2)}, \ \geq 2\epsilon l_j + 15\hat{s}_j.$$

The squares in $D_j \cup U_j$ can be added back and packed inside t_j in a region of width $\bar{w}_j$ and length $2\epsilon l_j + 5\hat{s}_j < \ell$. To see this, observe that the squares from

D_j can be packed in two strips of width $w_j + 2\hat{s}_j$ and length $\hat{s}_j$ (see Fig. 5). Since $\bar{w}_j \geq w_j - \epsilon\hat{s}_j$ and $\bar{w}_j \geq \hat{s}_j$, then D_j can also be packed in six strips of width $\bar{w}_j$ and length $\hat{s}_j$. The squares in U_j can be packed in a horizontal strip of length l_j and width $2\epsilon w_j/(2+\epsilon) < \epsilon w_j$. These squares can be re-arranged so they fit in at most $1/(2\epsilon)$ strips of length $2\epsilon l_j + \hat{s}_j$ and width ϵw_j. The total width of these strips is at most $\epsilon w_j/(2\epsilon) = w_j/2 < w_j - \hat{s}_j/2 < w_j - \epsilon\hat{s}_j \leq \bar{w}_j$, as $\hat{s}_j \leq w_j$ and $\epsilon \leq 1/3$.

The above process packs in the rounded block b_j all the squares of S_j, except those originally placed in strip t_j. The total profit of the packed squares is at least $(1-4\epsilon)\mathit{profit}(S_j)$, but now no square crosses the boundaries of b_j and, furthermore, the squares are packed in a very regular manner.

4.5 Composite Small Blocks

Let B_1 be the set of composite small blocks and let S_1 be the set of squares packed in the blocks of B_1. Recall that we are considering the case when every big square has profit larger than ϵOPT. Since S_1 contains only medium and small squares, then, $\mathit{profit}(S_1) \leq (1-\epsilon)OPT$.

Let $b_{j1} \in B_1$ be a composite small block and let $S_{j1} \subseteq S_1$ be the set of squares packed in b_{j1}. We will modify the way in which S_{j1} is packed inside b_{j1} by recursively applying to b_{j1} the transformations described in Sections 2, 4, 5, i.e., we will consider that now b_{j1} is the bin $\mathcal{R}$ where the squares need to be packed (however, now the bin might not be rectangular). Accordingly, we define $\rho'_0 = 1$, $\rho'_k = (\rho'_{k-1})^4(\epsilon/4)^{5+2^{\eta+1}}$, and $\rho_k = \rho'_k \times \hat{s}_{j1}$, for all integers $k \geq 1$, where $\hat{s}_{j1}$ is the size of the largest square in S_{j1}. Then, we define the sets $\mathcal{Q}^*_{kj} = \{Q_i \in S_{j1} \mid s_i \in (\rho_k, \rho_{k-1}]\}$, $k \geq 1$. Let $\tau \geq 2$ be the smallest index for which $\mathit{profit}(\mathcal{Q}^*_{\tau j}) \leq \epsilon \times \mathit{profit}(S_{j1})$. The set S_{j1} is then partitioned into big $\mathcal{B}^*_{j1} = \{Q_i \in S_{j1} \mid s_i > \rho_{\tau-1}\}$, medium $\mathcal{M}^*_{j1} = \{Q_i \in S_{j1} \mid \rho_\tau < s_i \leq \rho_{\tau-1}\}$, and small $\mathcal{S}^*_{j1} = \{Q_i \in S_{j1} \mid s_i \leq \rho_\tau\}$ squares as we did in Section 2. For simplicity, we required $\tau \geq 2$ to ensure that the set $\mathcal{B}^*_{j1}$ is not empty. We now need to consider two cases:

(*i*) The set $\mathcal{B}^*_{j1}$ contains a square Q_ℓ of profit no larger than $\epsilon \times \mathit{profit}(S_{j1})$. In this case we modify the packing for S_{j1} as indicated in Section 5.

(*ii*) Every square in $\mathcal{B}^*_{j1}$ has profit larger than $\mathit{profit}(S_{j1})$. We recursively partition b_{j1} into smaller blocks using the procedure described in Section 4.

With each recursive partitioning, the composite small blocks might get more complex shapes, since composite small blocks are constructed by merging several rectangular regions together. Fortunately, we only need to recursively partition these blocks a constant number of times, as we show below.

Let B_2 be the set of composite small blocks created after recursively partitioning all the blocks in B_1. Since in each block $b_{j1} \in B_1$ that needs to be partitioned there must be at least one square $Q \in \mathcal{B}^*_{j1}$ of profit larger than $\epsilon \times \mathit{profit}(S_{j1})$, then the total profit of the squares packed in the smaller blocks created after partitioning b_{j1} is at most $(1-\epsilon)\mathit{profit}(S_{j1})$. Therefore, the total

profit of the set S_2 of squares packed in all the blocks of B_2 is $\mathit{profit}(S_2) \leq (1-\epsilon)\sum_{b_{j1}\in B_1} \mathit{profit}(S_{j1}) = (1-\epsilon)\mathit{profit}(S_1) \leq (1-\epsilon)^2 OPT$.

Let B_i be the set of composite small blocks after i recursive applications of the partitioning procedure and let S_i be the set of squares packed in B_i. By the above discussion, by recursively partitioning the blocks in B_i we create a new set of composite small blocks B_{i+1} such that the total profit of the set S_{i+1} of squares packed in B_{i+1} is $\mathit{profit}(S_{i+1}) \leq (1-\epsilon)\mathit{profit}(S_i) \leq (1-\epsilon)^i OPT$.

If we perform this recursive partitioning $\eta = \log_{1-\epsilon}(\epsilon)$ times, the total profit of the squares packed inside the blocks B_η is at most $(1-\epsilon)^\eta OPT = \epsilon \times OPT$. Thus, we simply discard all squares packed by the optimum solution P^* in the blocks B_η, losing only profit at most $\epsilon \times OPT$. We can show that

$$|B_\eta| < \left(\frac{4}{\epsilon}\right)^{2^{\eta+2}} \quad \text{and} \quad n_\eta < \left(\frac{4}{\epsilon}\right)^{2^{\eta+1}}, \tag{4}$$

where n_η is the maximum number of vertices in any polygonal region delimiting a composite small block (we omit the proofs, due to space limitations).

This recursive partitioning creates a packing with a hierarchical structure for the composite small blocks. Large squares relative to the size of a composite small block split it into large, elongated, and small sub-blocks where squares are packed as indicated in this section. As we show below, the number and shape of the composite small blocks affects the time complexity of our algorithm.

5 Instances with a Low Profit Big Square

Let b_j be either the unit size square bin, or one of the composite small blocks created during the recursive partitioning procedure described above. Let S_j be the set of squares packed in b_j by the optimum solution P^*. We partition S_j into 3 sets: $\mathcal{B}_j^*$, $\mathcal{M}_j^*$, and $\mathcal{S}_j^*$, of big, medium, and small squares as described in Section 4.5. If b_j is a composite small block then, b_j was obtained by merging at most n_η rectangular blocks, each of width and length smaller than $\alpha\hat{s}_j$, where $\hat{s}_j$ is the size of the largest square in S_j. So, the total area of b_j is smaller than $n_\eta(\alpha\hat{s}_j)^2$ and the (constant) number n_j^* of big squares in $\mathcal{B}_j^*$ is

$$n_j^* < n_\eta(\alpha\hat{s}_j)^2/\rho_{\tau-1}^2 < n_\eta\alpha^2/\rho_{\tau-1}^2 = n_\eta\alpha^2(4/\epsilon)^{2(5+2^{\eta+1})(4^{1/\epsilon}-1)/3}. \tag{5}$$

Let us remove the medium squares $\mathcal{M}_j^*$ from b_j; causing a profit loss of value at most $\epsilon \times \mathit{profit}(S_j)$. This creates a size gap between the big and small squares. We also remove the small squares from b_j, but we will add them back later.

Let $Q_{\min} \in \mathcal{B}_j^*$ be a big square of profit at most $\epsilon \times \mathit{profit}(S_j)$ and p_j be the polygon delimiting b_j. The empty space in b_j left by the n_j^* big squares can be split with horizontal cutting lines starting at the vertices of p_j and at the vertices of the squares in $\mathcal{B}_j^*$ into m rectangles $\mathcal{R}_j = \{R_1, R_2, \ldots, R_m\}$. For convenience, we call these rectangles, large blocks. By (4), p_j has at most n_η vertices and since the total number of vertices of the big squares in $\mathcal{B}_j^*$ is $4n_j^*$, then,

$$m < n_\eta + 4n_j^*. \tag{6}$$

Now, add back the small squares and, then, round the width and length of each large block R_i down to the nearest value of the form $k\hat{s}$, where k is an integer value and $\hat{s} \leq \rho_\tau$ is the size of the largest small square in $\mathcal{S}_j^*$. After doing this, a set S_j' of small squares might not completely fit in the blocks $\mathcal{R}_j$. Let l_i be the length of block R_i and w_i be its width. We can show that the total area of the squares in S_j' is

$$A_j' < \frac{1}{5}\rho_{\tau-1}^2 \tag{7}$$

Let us remove $Q_{\min}$ from b_j, losing additional profit of value at most ϵOPT. This creates an empty square region E in b_j of area at least $\rho_{\tau-1} \times \rho_{\tau-1}$ where we can pack the small squares S_j' by using the NFDS algorithm. Let s' be the size of the largest square in S_j' and let s_E be the size of any side of E. Round s_E down to the nearest multiple of s'. Note that after rounding, $s_E > \rho_{\tau-1} - s' \geq \rho_{\tau-1} - \rho_\tau$.

Lemma 2. *[2] Let S be a set of squares, each of size at most δ. NFDS can pack S in a rectangular bin of length $\ell > \delta$ and width $w \leq (AREA(S)+\delta\ell-\delta^2)/(\ell-\delta)$, where $AREA(S)$ is the total area of the rectangles in S.*

By Lemma 2 and (7), S_j' can be packed in a bin of length s_E and width $w_j' \leq (\rho_{\tau-1}^2/5 + \rho_\tau s_E - \rho_\tau^2)/(s_E - \rho_\tau) < \left(\rho_{\tau-1}^2/5 + \rho_\tau\right)/(\rho_{\tau-1} - 2\rho_\tau)$. Since $\rho_\tau = \rho_{\tau-1}^4 \left(\frac{\epsilon}{4}\right)^{5+2^{\eta+1}} < 10^{-10}\epsilon\rho_{\tau-1}^4$, by (1), $< 10^{-10}\rho_{\tau-1}$, then

$$w_j' < (\rho_{\tau-1}^2/5 + \rho_{\tau-1}^4 \left(\frac{\epsilon}{4}\right)^{5+2^{\eta+1}})/(\rho_{\tau-1} - 2 \times 10^{-10}\rho_{\tau-1}) < \rho_{\tau-1}/4. \tag{8}$$

Therefore, all the squares in S_j' fit in the empty space left by $Q_{\min}$. Furthermore, after doing this we still have left an empty region of length at least s_E and width at least $s_E - \rho_{\tau-1}/4 \geq \frac{3}{4}\rho_{\tau-1} - \rho_\tau$. We now use NFDS to repack the squares contained in each block $R_i \in R_j^*$. Some of the small squares might not fit in the blocks R_i after being re-arranged by NFDS, but their total area [2] is at most $\sum_{R_i \in R_j^*}(2\rho_\tau w_i + \rho_\tau l_i) < A_j' < \rho_{\tau-1}^2/5$, where w_i is the width of R_i and l_i is its length. By Lemma 2 and (8), these squares can be packed in a bin of length s_E and width at most $\rho_{\tau-1}/4$. Hence, they fit in the empty space that remains in the area vacated by $Q_{\min}$.

6 Structure of a Near Optimum Solution

As shown in Section 4.5, the total number of composite small blocks that our recursive splitting procedure creates is at most $|B_\eta| < (4/\epsilon)^{2^{\eta+2}}$. Since each composite small block that needs to be recursively split must have at most $1/\epsilon$ big squares packed in it, the number of big squares involved in these η recursive splittings is at most $|B_\eta|/\epsilon$. In the final set $\tilde{S}$ of blocks produced by the recursive partitioning algorithm, each one of these blocks could contain a large square of low profit, and therefore, these blocks might need to be further split as described in Section 5: By (5) each block $b_j \in \tilde{S}$ could store n_j^* big squares, which by (6),

divide b_j into no more than $n_\eta + 4n_j^*$ blocks. Therefore, $\tilde{S}$ could be further split into $N_B \leq |B_\eta|(n_\eta + 4n_j^*)$ blocks. The total number of big squares packed in the bin is $N_S \leq |B_\eta|(\frac{1}{\epsilon} + n_j^*)$.

The transformations described in Sections 2-5 prove that for any instance Q of the square packing problem there is a near optimum solution P^+ that selects a subset Q^+ of squares of total profit $(1-O(\epsilon))OPT$ and packs them in the unit size square bin in a very regular manner:

- The packing P^+ labels a constant number $N_S \leq |B_\eta|(n_j^* + 1/\epsilon)$ of squares as *big squares*.
- The space not occupied by the big squares can be partitioned into a constant number $N_B \leq |B_\eta|(n_\eta + 4n_j^*)$ of large and elongated rectangular blocks.
- The remaining, unlabelled, squares are packed in the blocks so that none crosses any block boundaries.
- A large block b_i has length l_i and width w_i that are multiples of the size $\hat{s}_i$ of the largest square packed in b_i. Squares of total area at most $(l_i - 2\hat{s}_i)(w_i - \hat{s}_i)$ are packed in b_i using the NFDS algorithm.
- An elongated block b_j has one of its dimensions much larger (by at least a factor β/α) than the other one. The larger dimension of b_j is a multiple of the size $\hat{s}_j$ of the largest square packed in b_j; the smaller dimension is a multiple of $\epsilon \times \hat{s}_j$. Squares are packed in b_j using the KR algorithm.

7 The Algorithm

Our approximation algorithm for the square packing problem is relatively simple, as it just enumerates a polynomial number of packings with the structure described above and then it selects a packing with the highest profit. Some care is needed when selecting the squares to pack in each block to ensure that the solution produced has profit close to that of P^*.

Let n_L and n_E be the number of large and elongated blocks in P^+, respectively. The algorithm tries all non-negative values for n_L and n_E such that $n_L + n_E \leq N_B$. For every choice of n_L and n_E the algorithm must select the dimensions of each long and elongated block. To determine the dimensions of a block b_i, we first choose a square $\widehat{Q}_i \in Q$ to be the largest square packed in that block. If b_i is a large block, then its width w_i and length l_i are multiples of the size $\hat{s}_i$ of $\widehat{Q}_i$. Note that w_i and l_i are at most $n\hat{s}_i$, so there are $O(n^2)$ choices for the dimensions of b_i. If b_i is an elongated block, then its smaller dimension is a multiple of $\epsilon\hat{s}_i$ and its larger dimension is a multiple of $\hat{s}_i$. Hence, there are up to n^2/ϵ possible choices for the dimensions of b_i in this case.

For each choice of the number of blocks and the dimensions of each block, we need to select the set $\mathcal{G}$ of big squares to be packed in the bin. Since $|\mathcal{G}| \leq N_S$, the algorithm tries for $\mathcal{G}$ all subsets of Q of up to N_S squares. The number of possible subsets is $O(n^{N_S})$, which is polynomial in n as N_S is constant.

Let $\mathcal{L}$ be the set of blocks selected by the algorithm. The algorithm takes this set $\mathcal{L} \cup \mathcal{G}$ of rectangles and tries to pack them in the unit size bin $\mathcal{R}$. Since $|\mathcal{L} \cup \mathcal{G}|$

is constant, we can in $O(1)$ time either find a packing for $\mathcal{L} \cup \mathcal{G}$ or decide that no packing for them exists. To do this, let us first define an *upper-left justified packing* for a set S of rectangles as a packing in which no rectangle can be shifted up or to the left without overlapping other rectangles or crossing the boundaries of the bin. Clearly, any packing for S can be transformed into an upper-left justified packing by repeatedly shifting the rectangles up and/or to the left until no rectangle can be further moved. In an upper-left justified packing for S, every square $Q_j \in S$ has its upper left corner at a distance x_j (which is the sum of lengths of at most $|S|$ rectangles) from the left side of the bin and at a distance y_j (which is the sum of widths of at most $|S|$ rectangles) from the top of the bin. Therefore, for each square Q_j there are at most $2^{|S|}$ possible values for x_j and $2^{|S|}$ possible values for y_j.

To pack $\mathcal{L} \cup \mathcal{G}$, we can simply compute for each rectangle $r \in \mathcal{L} \cup \mathcal{G}$ the at most $2^{|\mathcal{L} \cup \mathcal{G}|} \times 2^{|\mathcal{L} \cup \mathcal{G}|}$ possible coordinates for its upper-left corner. If for one of these positionings all the rectangles fit in the bin without overlapping, then we have found a valid packing for them, otherwise no packing exists.

For each set $\mathcal{L}$ of blocks and $\mathcal{G}$ of big squares that can be packed in the bin, we consider the set $\mathcal{E} \subseteq \mathcal{L}$ of elongated blocks. Each elongated block $b_j \in \mathcal{E}$ needs to be split into frames and large blocks as described in Section 4.4. To show how this splitting is done, let us consider a block $b_i \in \mathcal{E}$. For simplicity we assume that $w_i < l_i$. First, we select a (multi)subset T_i of $h = \left(\frac{2+\epsilon}{\epsilon}\right)^2$ squares that will be used to split b_i into h frames and $h+1$ large blocks as explained in Section 4.4.1. The length of each frame is a multiple of $\hat{s}_j$ and its width is the sum of widths of a subset of squares from T_j; therefore, for each frame there are at most $n2^h$ possible values for its dimensions. Since there are h frames and n^h possible choices for T_j, there are $(n2^h)^h n^h$ ways of selecting frames for block b_j. For each choice of frames there is only one way of splitting the remaining space of b_j into large blocks.

The above process eliminates the elongated blocks from $\mathcal{L}$, but adds a new set of large blocks. Let $\mathcal{L}$ be the resulting set of large blocks and $\mathcal{F}$ be the set of frames created by splitting the elongated blocks. The final number N_L of large blocks is then $N_L = N_B + |\mathcal{E}|(h+1)$ and the number N_F of frames is $N_F = |\mathcal{E}|h$.

7.1 Selecting the Squares

The final step is to allocate a set of squares to each block and frame, and to pack them there. For each block $b_j \in \mathcal{L}$, of length l_j and width w_j, we select a square $\widehat{Q}_j \in Q$, of size $\hat{s}_j$, to be the largest square packed in it. We pack in b_j only squares of size at most $\hat{s}_j$, and total area at most $(l_j - 2\hat{s}_j)(w_j - \hat{s}_j)$. For each frame $f_j \in \mathcal{F}$ we choose two squares $\check{Q}_j$ and $\widehat{Q}_j$ of sizes $\check{s}_j$ and $\hat{s}_j$, $\check{s}_j \leq \hat{s}_j$. We can pack in f_j only squares of size at least $\check{s}_j$ and at most $\hat{s}_j$. If the total length l_j of f_j is smaller than its width, then the length of each square allocated to f_j is rounded up to $\hat{s}_j$, otherwise, the width of each square to be packed in f_j is rounded up to $\hat{s}_j$. The total area of the squares assigned to f_j must be at most $w_j l_j$.

Now, we must allocate to the blocks and frames $\mathcal{L} \cup \mathcal{F}$ a maximum profit subset of squares that satisfy the above conditions. This allocation problem is a special instance of the generalized assignment problem [3]. However, since $|\mathcal{L} \cup \mathcal{F}|$ is constant, a straightforward extension of the $O(n^2/\epsilon)$ FPTAS of Lawler [9] for the knapsack problem can be used to make the allocation. The only change that we need to make to the algorithm in [9] is that instead of computing single pairs $(profit(S), area(S))$ for candidate subsets S of squares, we need to compute $N_L + N_B$ pairs $(profit(S_1), area(S_1)), \ldots, (profit(S_{N_L+N_B}), area(S_{N_L+N_B}))$ for those candidate sets indicating how the squares are allocated to the blocks and frames. This change increases the time complexity of the algorithm to $O(n^2(N_L + N_B)/\epsilon)$.

The total profit of the squares allocated by this algorithm is at least $(1-\epsilon)p^+$, where p^+ is the total profit of the squares allocated by P^+ to the large blocks and frames. As discussed above, all squares allocated to a large block b_j are packed using the NFDS algorithm. Furthermore, all squares assigned to a frame f_j are packed by simply placing the squares side by side in the frame.

Theorem 1. *There is a PTAS for the square packing problem.*

References

1. N. Bansal and M. Sviridenko: Two-dimensional bin packing with one dimensional resource augmentation, to appear in *Discrete Optimization.*
2. E.G. Coffman, M.R. Garey, D.S. Johnson and R.E. Tarjan: Performance bounds for level-oriented two-dimensional packing algorithms, *SIAM Journal on Computing*, 9 (1980), 808-826.
3. R. Cohen, L. Katzir, and D. Raz: An Efficient Approximation for the Generalized Assignment Problem, *Information Processing Letters*, 100 (4), 162 – 166.
4. A.V. Fishkin, O. Gerber, K. Jansen and R. Solis-Oba: Packing weighted rectangles into a square, *Symposium on Mathematical Foundation of Computer Science*, (MFCS 2005), LNCS 3618, 352-363.
5. R. Harren: Approximating the orthogonal knapsack problem for hypercubes, *International Colloquium on Automata Languages and Programming* (ICALP 2006), 238–249.
6. K. Jansen and G. Zhang: Maximizing the total profit of rectangles packed into a rectangle, *Algorithmica*, Volume 47, 323–342, 2007.
7. D.J. Kleitman and M.M. Krieger: An optimal bound for two dimensional bin packing, *Proceedings of the 16th Annual IEEE Symposium on Foundations o f Computer Science (FOCS 1975)*, 163–168.
8. C. Kenyon and E. Rémila: A near-optimal solution to a two-dimensional cutting stock problem, *Mathematics of Operations Research* 25 (2000), 645-656.
9. E. Lawler: Fast approximation algorithms for knapsack problems, *Mathematics of Operations Research* 4 (1979), 339-356.
10. J.Y.-T. Leung, T.W. Tam, C.S. Wong, G.H. Young, and F.Y.L. Chin: Packing squares into a square, *Journal of Parallel and Distributed Computing*, 10 (1990), 271-275.
11. P. Novotny: On packing of squares into a rectangle, *Archivum Mathematicum*, 32 (1996), 75–83.