Online Square-into-Square Packing

Sándor P. Fekete[1] and Hella-Franziska Hoffmann[2]

[1] Department of Computer Science, TU Braunschweig, Germany
s.fekete@tu-bs.de

[2] Cheriton School of Computer Science, University of Waterloo, Canada
hrhoffmann@uwaterloo.ca

Abstract. In 1967, Moon and Moser proved a tight bound on the critical density of squares in squares: any set of squares with a total area of at most 1/2 can be packed into a unit square, which is tight. The proof requires full knowledge of the set, as the algorithmic solution consists in sorting the objects by decreasing size, and packing them greedily into shelves. Since then, the online version of the problem has remained open; the best upper bound is still 1/2, while the currently best lower bound is 1/3, due to Han et al. (2008). In this paper, we present a new lower bound of 11/32, based on a dynamic shelf allocation scheme, which may be interesting in itself.

We also give results for the closely related problem in which the size of the square container is not fixed, but must be dynamically increased in order to accommodate online sequences of objects. For this variant, we establish an upper bound of 3/7 for the critical density, and a lower bound of 1/8. When aiming for accommodating an online sequence of squares, this corresponds to a 2.82...-competitive method for minimizing the required container size, and a lower bound of 1.33... for the achievable factor.

Keywords: Packing, online problems, packing squares, critical density.

1 Introduction

Packing is one of the most natural and common optimization problems. Given a set $\mathcal{O}$ of objects and a container E, find a placement of all objects into E, such that no two overlap. Packing problems are highly relevant in many practical applications, as well as in geometric and abstract settings. Simple one-dimensional variants (such as the PARTITION case with two containers, or the KNAPSACK problem of a largest packable subset) are NP-hard. Additional difficulties occur in higher dimensions: as Leung et al. [9] showed, it is NP-hard even to check whether a given set of squares fits into a unit-square container.

When dealing with an important, but difficult optimization problem, it is crucial to develop a wide array of efficient methods for distinguishing feasible instances from the infeasible ones. In one dimension, a trivial necessary and sufficient criterion is the total size of the objects in comparison to the container. This makes it natural to consider a similar approach for the two-dimensional version: *What is the largest number δ, such that any family of squares with area at most δ can be packed into a unit square?* An upper bound of $\delta \leq 1/2$ is trivial: two squares of size $1/2 + \varepsilon$ cannot be packed. As Moon and Moser showed in 1967 [11], $\delta = 1/2$ is the correct critical bound: sort the objects by decreasing size, and greedily pack them into a vertical stack of one-dimensional "shelves", i.e., horizontal subpackings whose height is defined by the largest object.

This approach cannot be used when the set of objects is not known a priori, i.e., in an online setting. It is not hard to see that a pure shelf-packing approach can be arbitrarily bad. However, other, more sophisticated approaches were able to prove lower bounds for δ: the current best bound (established by Han et al. [4]) is based on a relatively natural recursive approach and shows that $\delta \geq 1/3$.

Furthermore, it may not always be desirable (or possible) to assume a fixed container: the total area of objects may remain small, so a fixed large, square container may be wasteful. Thus, it is logical to consider the size of the container itself as an optimization parameter. Moreover, considering a possibly *larger* container reflects the natural optimization scenario in which the full set of objects *must* be accommodated, possibly by paying a price in the container size. From this perspective, $1/\sqrt{\delta}$ yields a competitive factor for the minimum size of the container, which is maintained at any stage of the process. This perspective has been studied extensively for the case of an infinite strip, but not for an adjustable square.

Our Results. We establish a new best lower bound of $\delta \geq 11/32$ for packing an online sequence of squares into a fixed square container, breaking through the threshold of $1/3$ that is natural for simple recursive approaches based on brick-like structures. Our result is based on a two-dimensional system of multi-directional shelves and buffers, which are dynamically allocated and updated. We believe that this approach is interesting by itself, as it may not only yield worst-case estimates, but also provide a possible avenue for further improvements, and be useful as an algorithmic method.

As a second set of results, we establish the first upper and lower bounds for a square container, which is dynamically enlarged, but must maintain its quadratic shape. In particular, we show that there is an upper bound of $\delta \leq 3/7 < 1/2$ for the critical density, and a lower bound of $1/8 \leq \delta$; when focusing on the minimum size of a square container, these results correspond to a $2.82\ldots$-competitive factor, and a lower bound of $1.33\ldots$ for the achievable factor by any deterministic online algorithm.

Related Work: Offline Packing of Squares. One of the earliest considered packing variants is the problem of finding a dense square packing for a rectangular container. In 1966 Moser [12] first stated the question as follows:

> "What is the smallest number A such that any family of objects with total area at most 1 can be packed into a rectangle of area A?"

The offline case has been widely studied since 1966; there is a long list of results for packing squares into a rectangle. Already in 1967, Moon and Moser [11] gave the first bounds for A: any set of squares with total area at most 1 can be packed into a square with side lengths $\sqrt{2}$, which shows $A \leq 2$, and thus $\delta \geq 1/2$; they also proved $A \geq 1.2$. Meir and Moser [10] showed that any family of squares each with side lengths $\leq x$ and total area A can be packed into a rectangle of width w and height h, if $w, h \geq x$ and $x^2 + (w-x)(h-x) \geq A$; they also proved that any family of k-dimensional cubes with side lengths $\leq x$ and total volume V can be packed into a rectangular parallelepiped with edge lengths $a_1, \ldots, a_k$ if $a_i \geq x$ for $i = 1, \ldots, k$ and $x^k + \prod_{i=1}^{k} (a_i - x) \geq V$. Kleitman and Krieger improved the upper bound on A to $\sqrt{3} \approx 1.733$ [7] and to $4/\sqrt{6} \approx 1.633$ [8] by showing that any finite family of squares with total area 1 can be packed into a rectangle of size $\sqrt{2} \times 2/\sqrt{3}$. Novotný further improved the bounds to $1.244 \approx (2+\sqrt{3})/3 \leq A < 1.53$ in 1995 [13] and 1996 [14]. The current best known upper bound of 1.3999 is due to Hougardy [5].

Online Packing of Squares. In 1997, Januszewski and Lassak [6] studied the online version of the dense packing problem. In particular, they proved that for $d \geq 5$, every online sequence of d-dimensional cubes of total volume $2(\frac{1}{2})^d$ can be packed into the unit cube. For lower dimensions, they established online methods for packing (hyper-) cubes and squares with a total volume of at most $\frac{3}{2}(\frac{1}{2})^d$ and $\frac{5}{16}$ for $d \in \{3, 4\}$ and $d = 2$, respectively. The results are achieved by performing an online algorithm that subsequently divides the unit square into rectangles with aspect ratio $\sqrt{2}$. In the following, we call these rectangles *bricks*. The best known lower bound of $2(\frac{1}{2})^d$ for any $d \geq 1$ was presented by Meir and Moser [10].

Using a variant of the brick algorithm, Han et al. [4] extended the result to packing a 2-dimensional sequence with total area $\leq 1/3$ into the unit square.

A different kind of online square packing was considered by Fekete et al. [2,3]. The container is an unbounded strip, into which objects enter from above in a Tetris-like fashion; any new object must come to rest on a previously placed object, and the path to its final destination must be collision-free. Their best competitive factor is $34/13 \approx 2.6154\ldots$, which corresponds to an (asymptotic) packing density of $13/34 \approx 0.38\ldots$.

2 Packing into a Fixed Container

As noted in the introduction, it is relatively easy to achieve a dense packing of squares in an offline setting: sorting the items by decreasing size makes sure that a shelf-packing approach places squares of similar size together, so the loss of density remains relatively small. This line of attack is not available in an online setting; indeed, it is not hard to see that a brute-force shelf-packing method can be arbitrarily bad if the sequence of items consists of a limited number of medium-sized squares, followed by a large number of small ones. Allocating different size classes to different horizontal shelves is not a remedy, as we may end up saving space for squares that never appear, and run out of space for smaller squares in the process; on the other hand, fragmenting the space for large squares by placing small ones into it may be fatal when a large one does appear after all.

Previous approaches (in particular, the brick-packing algorithm) have sidestepped these difficulties by using a recursive subdivision scheme. While this leads to relatively good performance guarantees (such as the previous record of 1/3 for a competitive ratio), it seems impossible to tighten the lower bound; in particular, 1/3 seems to be a natural upper bound for this relatively direct approach. Thus, making progress on this natural and classical algorithmic problem requires less elegant, but more powerful tools.

In the following we present a different approach for overcoming the crucial impediment of mixed square sizes, and breaking through the barrier of 1/3. Our *Recursive Shelf Algorithm* aims at subdividing the set of squares into different size classes called *large*, *medium* and *small*, which are packed into pre-reserved shelves. The crucial challenge is to dynamically update regions when one of them gets filled up before the other ones do; in particular, we have to protect against the arrival of one large square, several medium-sized squares, or many small ones. To this end, we combine a number of new techniques:

- Initially, we assign carefully chosen horizontal strips for shelf-packing each size class.
- We provide rules for dynamically updating shelf space when required by the sequence of items. In particular, we accommodate a larger set of smaller squares by inserting additional *vertical* shelves into the space for larger squares whenever necessary.
- In order to achieve the desired overall density, we maintain a set of buffers for overflowing strips. These buffers can be used for different size classes, depending on the sequence of squares.

With the help of these techniques, and a careful analysis, we are able to establish $\delta \geq 11/32$. It should be noted that the development of this new technique may be more significant than the numerical improvement of the density bound: we are convinced that tightening the remaining gap towards the elusive 1/2 will be possible by an extended (but more complicated) case analysis.

In the following Section 2.1, we describe the general concept of shelf-packing. In Section 2.2 we give an overview of the algorithm. Section 2.3 sketches the placement of large objects, while Section 2.4 describes the packing created with medium-sized squares. The packing of small squares is discussed in Section 2.5. The overall performance is analyzed in Section 2.6. Due to limited space, various proof details and detailed pseudocode had to be omitted and can be found in the full version of the paper.

2.1 Shelf Packing

For a given subset of squares with maximum size h, a *shelf* $\mathcal{S}$ is a subrectangle of the container that has height h; a *shelf packing* places the squares into a shelf next to each other, until some object no longer fits; see Fig. 1(a). When that happens, the shelf is closed, and a new shelf gets opened.

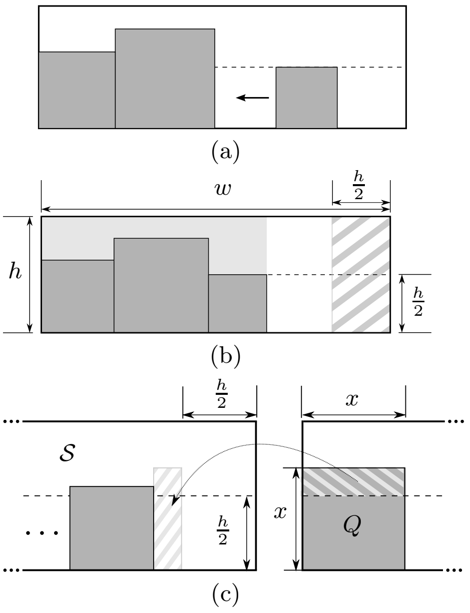


Fig. 1. (a) A shelf packed with squares of one height class. (b) Different areas of a shelf $\mathcal{S}$. *end($\mathcal{S}$)*: hatched region to the right, *occupied*($\mathcal{S}$): total area of squares (dark gray) and *usedSection*($\mathcal{S}$): region with light gray background (incl. *occupied*($\mathcal{S}$)). (c) Assignment of the extra area of Q (hatched) to $\mathcal{S}$ when square Q causes on overflow of shelf $\mathcal{S}$.

In the following, we will subdivide the set of possible squares into subsets, according to their size: We let H_k denote the height class belonging to the interval $(2^{-(k+1)}, 2^{-k}]$. In particular, we call all squares in H_0 *large*, all squares in H_1 *medium*, and all other squares (in $H_{\geq 2}$) *small*.

The proof for the following useful lemma (according to Moon and Moser [11]) is straightforward and omitted.

Lemma 1. *Let $\mathcal{S}$ be a shelf of height class H_k with width w and height h that is packed with a set $\mathcal{P}$ of squares all belonging to H_k. Let Q be an additional square of H_k with side length x that does not fit into $\mathcal{S}$. Then the total area of all the squares packed into $\mathcal{S}$ plus the area of Q is greater than $\frac{\|\mathcal{S}\|}{2} - (h/2)^2 + x \cdot \frac{h}{2}$.*

Notation. In the following, we let $w_\mathcal{S}$ denote the width of a shelf $\mathcal{S}$, $h_\mathcal{S}$ denote its height and $\mathcal{P}(\mathcal{S})$ denote the set of squares packed into it. We define *used-Section*($\mathcal{S}$) as the horizontal section of $\mathcal{S}$ that contains $\mathcal{P}(\mathcal{S})$; see Fig. 1(b). We denote the (possibly empty) section with width $h_\mathcal{S}/2$ and height $h_\mathcal{S}$ at the end of $\mathcal{S}$ by *end($\mathcal{S}$)*. The total area of the squares actually packed into a shelf $\mathcal{S}$ is *occupied*($\mathcal{S}$). The part of the square Q extending over the upper half of $\mathcal{S}$ is the *extra area* of Q. When Q causes a shelf $\mathcal{S}$ to be closed, we assign *extra*(Q) to $\mathcal{S}$; see Fig. 1(c). The total area assigned this way is referred to as *assigned*($\mathcal{S}$).

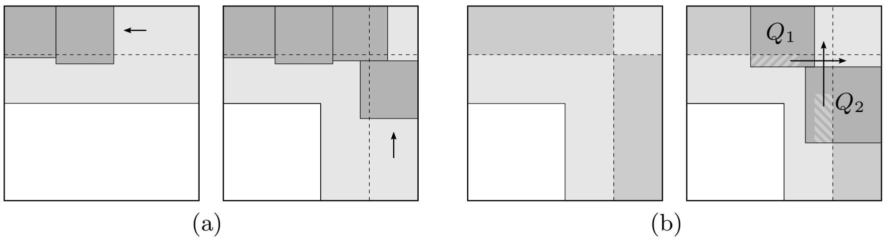


Fig. 2. (a): The L-shaped packing created with the squares of height class H_1. (b) Analysis of the Ceiling-Packing Algorithm: The algorithm packs at least as much as the gray area shown on the left. The parts of the packed squares that are used to fill the gaps appearing in the top right corner are depicted as hatched regions.

We let $\widetilde{\mathcal{A}}(\mathcal{S})$ denote the total of occupied and assigned area of $\mathcal{S}$ minus the extra area of the squares in $\mathcal{P}(\mathcal{S})$.

2.2 Algorithm Overview

We construct a shelf-based packing in the unit square by packing *small*, *medium* and *large squares* separately. We stop the Recursive Shelf Algorithm when the packings of two different subalgorithms would overlap. As it turns out, this can only happen when the total area of the given squares is greater than 11/32; details are provided in the "Combined Analysis" of Section 2.6, after describing the approach for individual size classes.

2.3 Packing Large Squares

The simplest packing subroutine is applied to large squares, i.e., of size greater than 1/2. We pack a square $Q_0 \in H_0$ into the top right corner of the unit square U. Clearly, only one large square can be part of a sequence with total area $\leq 11/32$. Hence, this single location for the squares in H_0 is sufficient.

2.4 Packing Medium Squares

We pack all medium squares (those with side lengths in $(1/4, 1/2]$) separately; note that there can be at most 5 of these squares, otherwise their total area is already bigger than $3/8 > 11/32$. Moreover, if there is a large square, there can be at most one medium square (otherwise the total area exceeds 3/8), and both can be packed next to each other.

We start with packing the H_1-squares from left to right coinciding with the top of the unit square U. If a square would cross the right boundary of U, we continue by placing the following squares from top to bottom coinciding with the right boundary; see Fig. 2(a).

We call the corresponding subroutine the *Ceiling Packing Algorithm.* Without interference of other height classes, the algorithm succeeds in packing any sequence of H_1-squares with total area $\leq 3/8$.

Theorem 2. *The Ceiling Packing Subroutine packs any sequence of medium squares with total area at most* $3/8$ *into the unit square.*

2.5 Packing Small Squares

As noted above, the presence of one large or few medium squares already assigns a majority of the required area, without causing too much fragmentation. Thus, the critical question is to deal with small squares in a way that leaves space for larger ones, but allows us to find extra space for a continuing sequence of small squares.

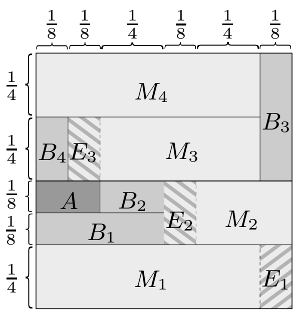
In the Recursive Shelf Algorithm we pack all small squares according to the *packSmall* subroutine, independent of the large and medium square packings. This method is also based on shelf packings. We partition the unit square into shelves for different height classes, such that for certain subsections we maintain a total packing density of 1/2.

Distribution of the Shelves. The general partition of the unit square is depicted in Fig. 3(a). The regions $M_1, \dots, M_4$ (in that order) act as shelves for height class H_2. We call the union M of the M_i the *main packing area*; this is the part of U that will definitely be packed with squares by our algorithm. The other regions may stay empty, depending on the sequence of incoming squares. The regions $B_1, \dots, B_4$ provide shelves for H_3. We call the union B of the B_j the *buffer area.* In the region A we reserve H_k-shelf space for every $k \geq 4$. We call A the *initial buffer area.* The ends E_i of the main packing regions M_i serve as both: parts of the main packing region and additional buffer areas. We call the union E of the E_i the *end buffer area.*

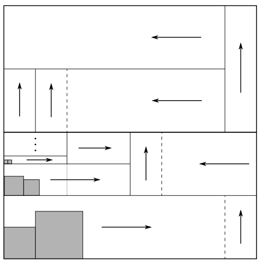
Packing Approach. During the packing process, we maintain open shelves for all the height classes for which we already received at least one square as input; we pack them according to the shelf-packing scheme described above. We start the packing of small squares in the lower half $\mathcal{H}_\ell$ of U. The region M_1 serves as the first H_2-shelf. The left half of B_1 serves as the *initial buffer shelf* for H_3. As soon as we receive the first square of a height class H_k with $k \geq 4$, we open an *initial buffer shelf* with width 1/4 and height 2^{-k} on top of the existing shelves in A. All of these shelves are packed from left to right; see Fig. 3(b) for the packing directions. In case the initial buffer for a height class H_k with ≥ 3 cannot accommodate another H_k-square, we continue packing H_k-squares into *vertical shelves* that we cut out of the main packing region. To achieve a packing density of 1/2 for these vertical shelves, we assign and occupy additional *buffer space* in $B \cup E$.

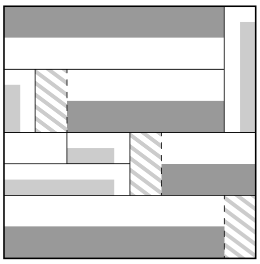
The reason for choosing this unit-square subdivision and packing scheme is to establish the following lemma.

Lemma 3. *In each step of the algorithm, the total area of the small squares packed into* U *is at least* $\|usedSection(M) \setminus E\|/2$.



(a)





(b) (c)

Fig. 3. (a) Region types in packingSmall. (b) Packing directions. (c) Area occupation in the main packing region (dark gray) and the usable buffer area (light gray).

We will argue that when the algorithm terminates, we have packed as much area as marked in Fig. 3(c). In the following, we describe the packings for the different small height classes in more detail.

Packing H_2-Squares. In the main packing area, we always maintain an open shelf for height class H_2 that is packed with H_2-squares, as described in Section 2.1. In order to avoid early collisions with large and medium squares, we start with packing M_1 from left to right, continuing with packing M_2 from right to left. Then we alternately treat M_3 and M_4 as the current main packing region, placing H_2-squares into the region whose *usedSection* is smaller. When the length of *usedSection*(M_4) becomes larger than 3/8, we prefer M_3 over M_4 until M_3 is full. This way, we can use the end E_3 as an additional buffer for vertical shelves in M_4 while ensuring that no overlap with the packing of H_1-squares can occur, unless the total area of the input exceeds 11/32. We know that each square of H_2 has a side length of at least half the height of the H_2-shelves. Consequently, if we only pack H_2-squares into the main packing area, the used sections of the M_i regions will be at least half full. However, if we receive a large number of $H_{\geq 3}$-squares as input, we may use these regions to accommodate $H_{\geq 3}$ as well.

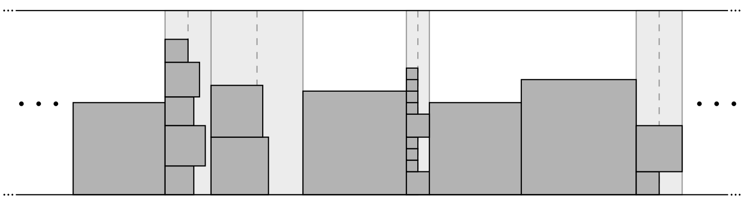


Fig. 4. Sample packing of square into a part of the main packing region according to the vertical shelf packing scheme

Packing of $H_{\geq 3}$-Squares. For $H_{k \geq 4}$ we reserve space in the initial buffer area A. As soon as we receive the first square of a height class H_k with $k \geq 4$, we open an H_k-shelf of length $1/4$ on top of the existing shelves in A. We call this shelf *initialBuffer*(k) and pack it from left to right with all subsequent H_k-squares according to the shelf-packing concept. If the *initialBuffer*(k) is full, we start allocating space in form of vertical H_k-shelves in the main packing region.

The same initial buffer packing is performed with H_3-squares, with the difference that we use B instead of A as the initial buffer packing location. As soon as the placement of a square Q would cause the initial buffer packing for H_3 in B to exceed a length of $1/4$, we open a vertical H_3-shelf with Q in the main packing region and stop using B for the initial buffer packing.

Vertical Shelves. In contrast to the situation in general shelf packings, we do not only pack H_2-squares into the H_2-shelves in the main packing region, but also cut out rectangular slices that serve as shelves for smaller height classes with a full initial buffer. We treat these vertical shelves as usual shelves for their corresponding height class. The width of each vertical main shelf for a height class H_k is always 2^{-k} and its height is always $1/4$, as it was formed by a slice of an H_2-shelf; see Fig. 4 for a sample packing. When closed, the resulting vertical shelf packings have a density of at least $3/8$; see Lemma 1. To achieve a total packing density of $1/2$ also for closed vertical shelves, we use the additional space reserved in the buffer area.

Buffer Assignments. There are three different ways to assign a buffer to a vertical shelf $\mathcal{S}$ for H_k with width $w_{\mathcal{S}}$.

1. If there is enough unassigned but occupied buffer area in region B, we simply assign a $w_{\mathcal{S}}/2$-wide slice of this area to $\mathcal{S}$.
2. Otherwise, we occupy further parts of the buffer area B by either
   (a) opening a $w_{\mathcal{S}}/2$-wide buffer subshelf $\mathcal{B}$ for height class H_k (if $k > 3$) or
   (b) simply packing the square into the buffer area (if $k = 3$).
   In both cases we assign a $(w_{\mathcal{S}}/2)^2$ part of the newly occupied area to $\mathcal{S}$.

If an overflow occurs at a vertical buffer subshelf for height class H_k, we want to pack the non-fitting square into the corresponding vertical main packing shelf $\mathcal{S}$. Hence, we start allocating buffer space for $\mathcal{S}$ when there is just enough space left in $\mathcal{S}$ to place another H_k-square. In particular, we open buffer subshelf $\mathcal{B}$ as soon as the next square would intersect *head($\mathcal{S}$)*, the top $w_{\mathcal{S}} \times w_{\mathcal{S}}$ part of $\mathcal{S}$.

In the following, we explain why this assignment is sufficient to establish the desired density in the main packing region.

By Lemma 1, we have $\widetilde{\mathcal{A}}(\mathcal{S}) \geq \|\mathcal{S} \setminus \mathit{end(\mathcal{S})}\|/2$ for any closed shelf $\mathcal{S}$ of our packing. Thus, for each vertical shelf $\mathcal{S}$ with width $w_{\mathcal{S}}$, we need to reserve an area of at most $\|\mathit{end(\mathcal{S})}\|/2 = (w_{\mathcal{S}}/2)^2$ in the buffer region in order to establish Lemma 3.

In case we create a vertical buffer subshelf $\mathcal{B}$ of width $w_{\mathcal{B}}$ $(=w_{\mathcal{S}})$ and height $h_{\mathcal{B}}$, we assign a $(w_{\mathcal{B}}/2)^2$-sized part of $\mathcal{B}$ to the corresponding vertical shelf $\mathcal{S}$ and have an area of $w_{\mathcal{B}}/2 \cdot h_{\mathcal{B}}/2$ to spare. We release this extra space as an occupied but unassigned buffer slice of density $1/2$. We do the same with the extra area of an H_3-square that was placed in the buffer region. This way we ensure that every vertical shelf of width w effectively allocates a buffer of total length $w/2$ and that the area assigned in case 1 is sufficient. Thus, we get the following property of closed vertical main packing shelves; see the full paper for a detailed proof.

Lemma 4. *Let $\mathcal{S}$ be a closed vertical shelf for H_k that has a buffer part assigned to it. Then $\widetilde{\mathcal{A}}(\mathcal{S}) \geq \|\mathcal{S}\|/2$.*

The gaps remaining in an open vertical main packing shelf $\mathcal{S}$ may be larger. However, whenever we open a vertical shelf $\mathcal{S}$ for a height class H_k, we assign the occupied area of the initial buffer of H_k to it. This way we establish the following property.

Lemma 5. *If $\mathcal{S}$ is an open vertical shelf for H_k, then $\widetilde{\mathcal{A}}(\mathcal{S}) \geq \|\mathcal{S}\|/2$.*

Restricting the Used Buffer Region. To avoid early collisions with large and medium squares, we only pack new squares into the buffer region if there is not enough unassigned, occupied area remaining in the buffer regions.

Let $\mathcal{S}$ be a vertical main shelf for H_3, *bufferlength* be the total width of the occupied buffer sections and *requiredBufferLength* be the total buffer width required for the vertical shelves. If *bufferlength* $\geq$ *requiredBufferLength*, there must be an occupied but unassigned buffer slice, which we assign to $\mathcal{S}$. Otherwise, we need the area of an H_3-square Q to get the remaining buffer part. We distinguish two cases:

1. If *bufferlength*$+x \leq$ *requiredBufferLength*$+1/16$, we pack Q into B, ensuring the used buffer section to be at most $1/16$ ahead of the section required.
2. Otherwise, the placement of Q in the buffer region would result in an undesirable buffer growth. In this case, we pack Q instead into its corresponding vertical main shelf and assign its potential extra space of $(x - 1/16) \cdot x$ to the buffer region. This way, a $(x - 1/16)$-wide buffer part is gained from Q, and the new buffer length is *bufferlength*$+ x - 1/16 >$ *requiredBufferLength.*

Buffer Region Overflow. A packing overflow may also occur in the buffer regions $B_1, \ldots, B_4$. In this case, we proceed as in every other shelf: we pack the current H_3-square or buffer subshelf into the next B_i-region in order and assign potential extra space from H_3-squares to the overflowing buffer shelf.

If an $H_{\geq 4}$-square cannot be placed at the end of *usedSection*(B_4), we pack it into the end of any of the other buffer regions of $B \cup E$. We claim that there must be enough space for this in at least one of the B_j-regions, as long as the total input area does not exceed 11/32.

By construction, we only need a $w/2$-wide buffer slice for each vertical main shelf with width w. Thus, the total buffer length required for $M \setminus E$ is at most 22/16. Unfortunately, the buffer region $B = B_1 \cup \cdots \cup B_4$ does only provide a total usable buffer length of 17/16 in the worst case. To get the missing buffer space of length 5/16, we also use the initial buffers and allocate extra space at the end of the main packing regions as buffer area.

Additional End Buffer Usage. The actual buffer space available in the E_i-regions depends on the lengths of the main packing sections that overlap with them. We call the part of *usedSection*(M_i) that overlaps with the considered E_i region $\mathcal{L}$. We denote the width of $\mathcal{L}$ by ℓ. Depending on ℓ, we choose between two different packing schemes for using E_i as a buffer.

1. *If* $\ell > w_{E_i}/2$*:* Then close the buffer shelf E_i (without having actively used it as an H_3-buffer-shelf) and simply use $\widetilde{\mathcal{A}}(\mathcal{L})$ as additional buffer area.
2. *If* $\ell \leq w_{E_i}/2$*:* Then treat $E_i \setminus \mathcal{L}$ similar to the normal buffer regions B_j; see Fig. 5(a).

We keep packing squares into $E_i \setminus \mathcal{L}$, according to the buffer-packing scheme, until the packing reaches a length of 2/16. We combine the area occupied this way with the area $\widetilde{\mathcal{A}}(\mathcal{L})$ of $\mathcal{L}$ to get proper buffers; see Fig. 5(b) and the full paper for pseudocode. We use 0.5/16 of the gained buffer length to achieve a density of 1/2 for $\mathcal{L}$ and hence get an additional buffer length of 1.5/16 from each of the E_i-regions. We start using (a part of) E_i as an additional buffer region, as soon as the corresponding main shelf M_i is closed. With the extra buffer length gained in E, we provide enough buffer space to fill the gaps remaining in all vertical main packing shelves in $M \setminus E$.

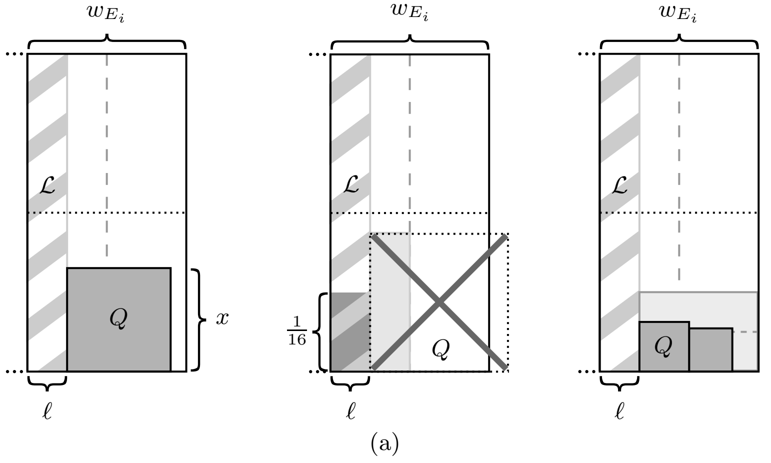
Lemma 6. *The total buffer space provided in* $A \cup B \cup E$ *is sufficient for the vertical shelves in* $M \setminus E$*.*

Lemmas 4, 5 and 6 directly prove the invariant of Lemma 3. That is, we can assume a density of 1/2 for the used sections of $M \setminus E$. By construction, the algorithm successfully packs all small squares, until a square Q would intersect the top left corner of U in M_4. At this time the total area of small input squares must be greater than $\|\bigcup_{i=1}^{3}(M_i \setminus E_i) \cup M_4\|/2 = 11/32$.

Theorem 7. *The packSmall Subroutine packs any sequence of small squares with total area at most* 11/32 *into the unit square.*

2.6 Combined Analysis

In the previous sections we proved that the algorithm successfully packs large, medium and large squares separately, as long as input has a total area of at most



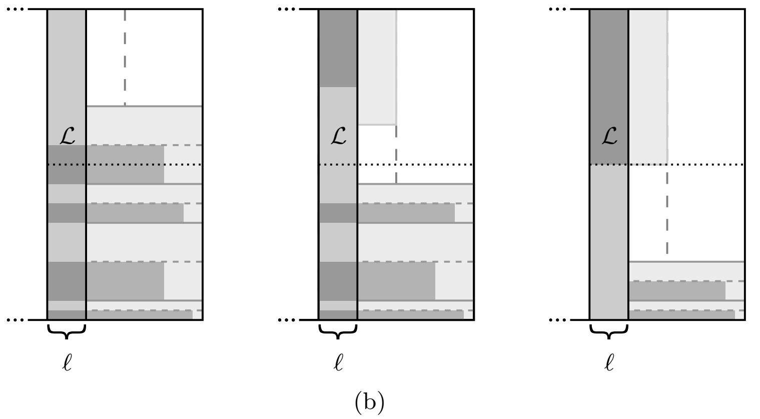


Fig. 5. (a) The packing performed in the end buffer regions: (left) packing of a fitting H_3-square, (center) extra area assignment from a non-fitting H_3-square and (right) subshelf packing of $H_{\geq 4}$ squares. (b) The assignment of $\widetilde{\mathcal{A}}(\mathcal{L})$, when E_i is closed: (left) we use all of $\widetilde{\mathcal{A}}(\mathcal{L})$ to extend the horizontal subshelves to a total length of w_{E_i}, (center) we use one half of $\widetilde{\mathcal{A}}(\mathcal{L})$ for the horizontal subshelf extensions and the other for an H_3-buffer or (right) we must use all of $\widetilde{\mathcal{A}}(\mathcal{L})$ for H_3-buffer assignment.

11/32. A case distinction over all possible collisions that may appear between the packings of these height classes can be used to prove the main result. An important property for this proof is the following; see the full paper for details.

Lemma 8. *Let Q be a small square with side length x in the buffer region B. Let ℓ be the total usable length of the buffer (value of bufferlength) right before Q was packed into B and let $\mathcal{P}$ be the set of small squares packed into U. Then the total area of the small input squares $\|\mathcal{P}\|$ is greater than $(\ell + x - 1/16) \cdot 1/4$.*

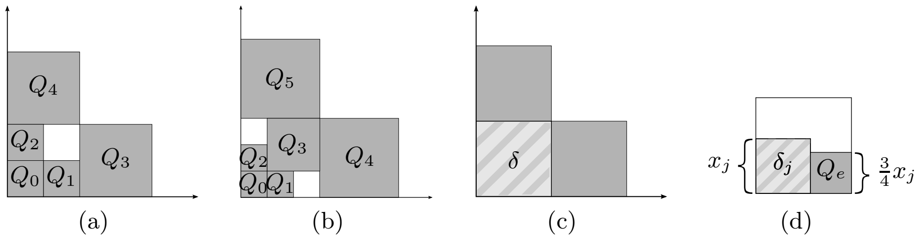


Fig. 6. Different choices in the lower-bound sequence: (a) Packing after choosing a side position. (b) Packing after choosing a center position. (c) The recurring pattern. (d) Packing a last square.

Theorem 9. *The Recursive Shelf Algorithm packs any sequence of squares with total area at most* 11/32 *into the unit square.*

3 Packing into a Dynamic Container

Now we discuss the problem of online packing a sequence of squares into a dynamic square container. At each stage, the container must be large enough to accommodate all objects; this requires keeping the container tight early on, but may require increasing its edge length appropriately during the process.

In the following, we give a non-trivial family of instances, which prove that no online algorithm can maintain a packing density greater than 3/7 for an arbitrary input sequence of squares and introduce an online square packing algorithm that maintains a packing density of 1/8 for an arbitrarily input sequence of squares.

3.1 An Upper Bound on δ

If the total area of the given sequence is unknown in advance, the problem of finding a dense online packing becomes harder. As it turns out, a density of 1/2 can no longer be achieved.

Theorem 10. *There are sequences for which no deterministic online packing algorithm can maintain a density strictly greater than* $3/7 \approx 0.4286$*.*

Proof. We construct an appropriate sequence of squares, depending on what choices a deterministic player makes; see Fig. 6. At each stage, the player must place a square Q_3 into a corner position (Fig. 6(a)) or into a center position (Fig. 6(b)); the opponent responds by either requesting another square of the same size (a), or two of the size of the current spanning box. This is repeated. A straightforward computation shows that the asymptotic density becomes 2/3, if the player keeps choosing side positions, and 3/4, if he keeps choosing center positions; for mixed choices, the density lies in between. Once the density is arbitrarily close to 3/4, with the center position occupied, the opponent can request a final square of size 3/4 of the current spanning box, for a density close

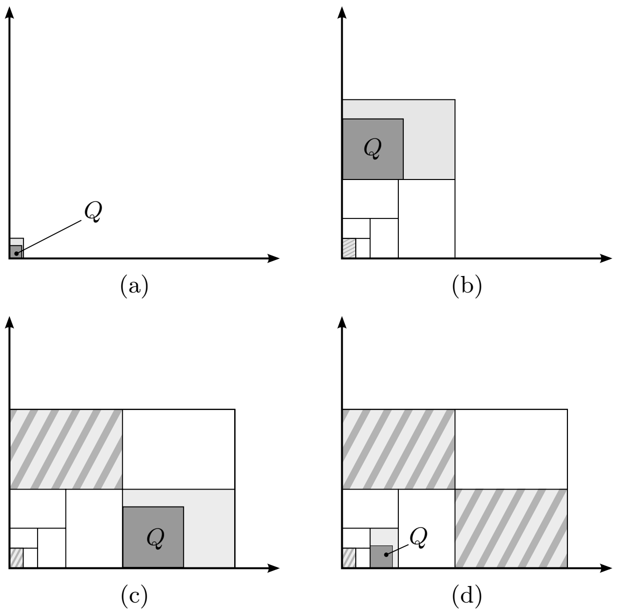


Fig. 7. The modified Brick-Packing algorithm for an input square Q. Occupied bricks are hatched, free bricks are blank. (a) A first square gets placed into the lower left corner, $B_{max} = S(Q)$. (b) If $S(Q) > B_{max}$, we double B_{max} until Q fits. (c) If Q does not fit into B_{max}, but $\|S(Q)\| < \|B_{max}\|$, we double B_{max} and subdivide the resulting brick. (d) If Q fits into B_{max}, we pack it into the smallest free fitting subbrick.

to $\frac{3/4+(3/4)^2}{(7/4)^2} = 3/7$; if the center position does not get occupied, the density is even worse. □

3.2 A Lower Bound on δ

When placing squares into a dynamic container, we cannot use our Recursive Shelf Algorithm, as it requires allocating shelves from all four container boundaries, which are not known in advance. However, we can adapt the Brick Algorithm by [6]: we consider bricks with side lengths equal to a power of $\sqrt{2}$ (and aspect ratio $1/\sqrt{2}$ or $\sqrt{2}$). We let B_k denote the brick of size $(\sqrt{2}^k, \sqrt{2}^{k+1})$ and let $S(Q)$ denote the smallest brick B_i that may contain a given square Q.

There are two crucial modifications: (1) The first square Q is packed into a brick of size $S(Q)$ with its lower left corner in the origin and (2) instead of always subdividing the existing bricks (starting with three fixed ones), we may repeatedly double the current maximum existing brick B_{max} to make room for large incoming squares. Apart from that, we keep the same packing scheme: Place each square Q into (a subbrick of) the smallest free brick that can contain Q; see Fig. 7 for an illustration.

Theorem 11. *For any input sequence of squares, the Dynamic Brick Algorithm maintains a packing density of at least* $1/8$.

Proof. By construction, every occupied brick has a density of at least $1/(2\sqrt{2})$. It is easy to see that in every step of the algorithm at most half the area of B_{max} consists of free bricks; compare [6]. Because B_{max} always contains all occupied bricks (and thus all packed squares), the ratio of $\|B_{max}\|$ to the area of the smallest enclosing square is at least $1/\sqrt{2}$. Therefore, the algorithm maintains an overall density of at least $(1/(2\sqrt{2})) \cdot (1/2) \cdot (1/\sqrt{2}) = 1/8$. □

3.3 Minimizing Container Size

The above results consider the worst-case ratio for the packing density. A closely related question is the online optimization problem of maintaining a square container with minimum edge length. The following is an easy consequence of Theorem 11, as a square of edge length $2\sqrt{2}$ can accommodate a unit area when packed with density $1/8$. By considering optimal offline packings for the class of examples constructed in Theorem 10, it is straightforward to get a lower bound of $4/3$ for any deterministic online algorithm.

Corollary 12. *Dynamic Brick Packing provides a competitive factor of* $2\sqrt{2} = 2.82\ldots$ *for packing an online sequence of squares into a square container with small edge length. The same problem has a lower bound of 4/3 for the competitive factor.*

4 Conclusion

We have presented progress on two natural variants of packing squares into a square in an online fashion. The most immediate open question remains the critical packing density for a fixed container, where the correct value may actually be less than $1/2$. Online packing into a dynamic container remains wide open. There is still slack in both bounds, but probably more in the lower bound.

There are many interesting related questions. What is the critical density (offline and online) for packing circles into a unit square? This was raised by Demaine et al. [1]. In an offline setting, there is a lower bound of $\pi/8 = 0.392\ldots$, and an upper bound of $\frac{2\pi}{(2+\sqrt{2})^2} = 0.539\ldots$, which is conjectured to be tight. Another question is to consider the critical density as a function of the size of the largest object. In an offline context, the proof by Moon and Moser provides an answer, but little is known in an online setting.

References

1. Demaine, E.D., Fekete, S.P., Lang, R.J.: Circle packing for origami design is hard. In: Origami5, pp. 609–626. AK Peters/CRC Press (2011)
2. Fekete, S.P., Kamphans, T., Schweer, N.: Online square packing. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 302–314. Springer, Heidelberg (2009)
3. Fekete, S.P., Kamphans, T., Schweer, N.: Online square packing with gravity. Algorithmica (to appear, 2013)
4. Han, X., Iwama, K., Zhang, G.: Online removable square packing. Theory of Computing Systems 43(1), 38–55 (2008)
5. Hougardy, S.: On packing squares into a rectangle. Computational Geometry: Theory and Applications 44(8), 456–463 (2011)
6. Januszewski, J., Lassak, M.: On-line packing sequences of cubes in the unit cube. Geometriae Dedicata 67(3), 285–293 (1997)
7. Kleitman, D., Krieger, M.: Packing squares in rectangles I. Annals of the New York Academy of Sciences 175, 253–262 (1970)
8. Kleitman, D.J., Krieger, M.M.: An optimal bound for two dimensional bin packing. In: 16th Annual Symposium on Foundations of Computer Science (FOCS), pp. 163–168 (1975)
9. Leung, J.Y.-T., Tam, T.W., Wong, C.S., Young, G.H., Chin, F.Y.L.: Packing squares into a square. J. Parallel and Dist. Comp. 10(3), 271–275 (1990)
10. Meir, A., Moser, L.: On packing of squares and cubes. Journal of Combinatorial Theory 5(2), 126–134 (1968)
11. Moon, J., Moser, L.: Some packing and covering theorems. Colloquium Mathematicum 17, 103–110 (1967)
12. Moser, L.: Poorly formulated unsolved problems of combinatorial geometry. Mimeographed (1966)
13. Novotný, P.: A note on a packing of squares. Stud. Univ. Transp. Commun. Žilina Math.-Phys. Ser. 10, 35–39 (1995)
14. Novotný, P.: On packing of squares into a rectangle. Arch. Math. (Brno) 32(2), 75–83 (1996)