Applications and Extensions of The Bin Packing Problem

by

Pooya Nikbakht

A Thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfilment of the requirements of the degree of

MASTER OF SCIENCE

Department of Computer Science
University of Manitoba
Winnipeg

Acknowledgements

I wish to express my deepest gratitude to my supervisor, Dr. Shahin Kamali, who made my master's program an experience far beyond what I expected - beyond just studying. Even though without his generous and persistent help and support, I would never have been able to accomplish my thesis to the level that I did, he generously allowed me to experience a feeling of confidence and fulfillment from every single step we took throughout the thesis process. He is a wonderful person both professionally and personally, and I feel very fortunate to have had the chance to learn from him. My master's was a period of my life that I will cherish forever.

Furthermore, I thank Dr. Stephane Durocher and Dr. Helen Cameron, my committee members; your encouraging words and detailed feedback have been very valuable to me.

And my sincerest thanks to my wonderful wife, Elahe, who has always been a source of selfless support and love for me. Elahe, without you and all your support, I would not even be here to write these words; thanks for everything you have done for me...

Abstract

Bin packing is a classic optimization problem with many applications and variants. In its basic form, the goal is to pack items of different sizes in the range $(0, 1]$ into the minimum number of bins of unit capacity. In doing so, an "offline" algorithm has access to all items before packing any of them, while an "online" algorithm receives items one by one and places each item into a bin without any prior knowledge about the forthcoming items. In this thesis, we study two variants of the bin packing problem: square packing, and fault-tolerant bin packing. Our goal is to design the algorithms that work well under the worst-case scenarios, where algorithms are evaluated based on their approximation factor (in the offline setting), and competitive ratio (in the online setting).

In the "square packing problem", items are squares of various side lengths, and bins are unit squares. We study this problem in both offline and online settings. Unlike previous work, we allow an algorithm to rotate items by any degree. This modification is consistent with the applications of square packing in stock cutting. We prove that the offline problem is NP-Hard and provide an asymptotic polynomial-time approximation scheme (APTAS) under an augmented resource setting. For the online setting, we present an online algorithm with a competitive ratio of at most 2.306. We also introduce an online algorithm with a competitive ratio of at most 1.897 for a related problem in which the goal is to pack "tans" (half-square triangles) into unit squares.

In the "fault-tolerant bin packing problem", items are replicas of tenants (such as applications or databases), and bins are servers of uniform capacity. As servers fail on a regular basis, a valid packing should tolerate the failure of a certain number of bins. Applications of this problem are mostly online as requests for hosting tenants are made sequentially and over time. The best existing online algorithm is known to have a competitive ratio of at most 2, and works under a setting in which bin failures take place after the packing process is concluded. We study a more practical setting where the bin failures might take place during the packing process, and introduce an algorithm with an improved competitive ratio of at most 1.75 for this general setting.

Contents

List of Figures

List of Tables

For Elahe, my wife,

the most kind-hearted and humane person I know...

In honor of my father, who I lost unbelievably before my thesis defense, who taught me the way of critical thinking, my most valuable asset...

For my mother, to whom I owe my existence...

Chapter 1

Introduction

Bin packing is a well-known problem in optimization theory with many variants and applications in practice. Each variant of the problem can be studied in offline or online settings. In the most basic form of the problem, an input to an *offline algorithm* is a multi-set of items with different sizes in the range $(0, 1]$, and the target is to pack them into the minimum number of bins of uniform capacity. An input to an *online algorithm*, on the other hand, is not revealed at once, and items appear one by one. An online algorithm can either place a revealed item in an open bin that has enough space, or open a new bin for the item. The decisions of an online algorithm are *irrevocable* in the sense that, after placing an item into a bin, the algorithm cannot move the item into another bin. Note that the objective is the same under both settings, that is, to minimize the number of opened bins.

The bin packing problem and almost all its variants are NP-hard problems. Therefore, in the offline setting, *approximation algorithms* are devised to achieve a polynomial running time at the expense of the optimality. A (polynomial-time) approximation algorithm A_{off} has asymptotic approximation ratio of r iff for any input σ, we can write $A_{\text{off}}(\sigma) \leq r\ \textsc{Opt}(\sigma) + c$, where $A_{\text{off}}(\sigma)$ and $\textsc{Opt}(\sigma)$ are, respectively, the number of bins opened by A_{off} and Opt for packing σ, and c is a constant independent of the length of σ. When $c = 0$, the ratio r is called the *absolute approximation ratio* of A_{off}. Our focus on this thesis is on the asymptotic approximation ratio. If one can devise an algorithm with an asymptotic approximation ratio of $1 + \epsilon$, for some arbitrary small $\epsilon > 0$, then such an algorithm is called

an asymptotic polynomial-time approximation scheme (APTAS). The running time of an APTAS needs to be polynomial in the number n of items, but it can be a super-polynomial function of ϵ.

The standard worst-case measure for comparing online algorithms is the *competitive analysis*, which compares the cost of an online algorithm with that of an optimal offline algorithm Opt [1]. Note that Opt not only knows the entire input (i.e., is offline), but also has unbounded computational power. An online algorithm A_{on} has *asymptotic competitive ratio* of r iff for any input σ, we can write $A_{on}(\sigma) \leq r\ \text{Opt}(\sigma) + c$, where $A_{on}(\sigma)$ and $\text{Opt}(\sigma)$ are, respectively, the number of bins opened by A_{on} and Opt for packing σ, and c is a constant independent of the length of σ. When $c = 0$, the ratio r is called the *absolute competitive ratio* of A_{on}.

Resource augmentation [2] provides a relaxed framework for the analysis of approximation and online algorithms. Let A be an approximation/online algorithm that packs items into augmented bins of capacity $1 + \alpha$, where $\alpha > 0$ is a parameter. Under the augmented setting [3], the competitive ratio of A is the maximum ratio (over all sequences) between the number of bins opened by A when packing items into augmented bins of capacity $1 + \alpha$, and the number of bins opened by Opt when packing items into unit-sized bins.

In this thesis, we consider the following two variants of the bin packing problem:

(I) We study the offline and online square packing problems, where items are squares of various sizes and bins are unit squares. The problem has applications in cutting stock. Unlike previous work, we allow the algorithm to rotate items when placing them into a bin.

(II) We study the online fault-tolerant bin packing problem, where bins represent servers of uniform capacity, and items are tenants' replicas of different workload. The tenants of the server can be for example some databases or applications hosted on the server. Given that servers might fail on a regular basis, we are interested in algorithms that tolerate the failure of up to f bins, where f is an input parameter of the problem. To be fault-tolerant, it is necessary to pack

at least $f + 1$ replicas of each item in separate bins to ensure the availability of at least one replica at any given time.

For (I) in the offline setting, we show that the problem is NP-Hard, and then provide an APTAS under a resource augmented setting. For (I) in the online setting, we introduce an algorithm that provides an upper bound of 2.306 for the competitive ratio attainable by online algorithms. We also studied and introduced an algorithm for two other online variants of the problem where: (I-1) square items have side sizes of at most 1/2, and (I-2) items are half-square triangles (instead of squares). Our algorithms achieve an asymptotic competitive ratio of at most 1.732 and 1.897 for (I-1) and (I-2), respectively.

Applications of the fault-tolerant bin packing problem are mostly online. This is because requests for hosting tenants are made sequentially and over time. Therefore, we study (II) only under the online setting. We introduce a more practical model than the existing models, which assume the bin failures take place only after the packing process. We assume that the failures might also happen during the packing process, and for this general model, provide an algorithm with a competitive ratio of at most 1.75. Given that our algorithm also works under the weaker model (where bins fail only after packing is concluded), our result provides an improvement over the best existing competitive ratio of 2 for that model.

Roadmap. We will start by reviewing the existing models and results for the bin packing problem in Chapter 2. We do not aim to be exhaustive, but rather present the highlights that are related to other parts of the thesis. In Chapter 3, we provide the formal definition of the problems that we study, present the motivation behind them, and briefly review our contribution for each problem. In each of the three chapters that follow, we present our results in detail for one of the main problems that we studied. Section 4 elaborates our results for the offline square packing with rotation problem, Section 5 is related to the online square packing with rotation problem, and Section 6 is focused on the online fault-tolerant bin packing problem. Finally, we provide a conclusion in Chapter 7.

Chapter 2

Literature Review

In this chapter, we review the models and existing results for the variants of the bin packing problem that are related to our research. We will start with an overview of classic bin packing in Section 2.1. We then review the extensions of this problem into two dimensions in Section 2.2. Finally, in Section 2.3, we will review the fault-tolerant bin packing and its applications in cloud systems.

2.1 One-dimensional bin packing

In the classic bin packing problem, the input is a multi-set of items of various sizes in the range $(0, 1]$, and the goal is to pack these items into the minimum number of unit bins of capacity 1.

In the offline setting, the bin packing problem is known to be NP-hard (see, e.g., [4]). The hardness can be proved using a reduction from the partition problem. Yet, an Asymptotic Polynomial Approximation Scheme (APTAS) has been introduced for the problem [5], such that, for any $0 < \epsilon \leq 1$ and input σ, there is a polynomial-time algorithm that uses at most $(1+\epsilon)Opt(\sigma)+1$ bins to pack any input where $Opt(\sigma)$ is the cost of OPT to pack σ. Karmarkar and Karp [6] proposed an algorithm that opens at most $Opt(\sigma) + \mathcal{O}(\lg^2 Opt(\sigma))$ bins to pack any σ. Rothvoß [7] designed a better solution that opens $Opt(\sigma) + \mathcal{O}(\lg Opt(\sigma) \lg\lg Opt(\sigma))$ bins. It should be mentioned that these algorithms use Linear Programming solvers, which make them undesirable in practice due to their

super-linear complexity. There are more practical offline algorithms such as First-Fit-Decreasing (FFD) and Best-Fit-Decreasing (BFD), which first sort all the items in decreasing order of their sizes and then use a simple greedy algorithm to pack them. For example, FFD and BFD, respectively, apply First-Fit (FF), which places each item into the first bin with enough space, and Best-Fit (BF), which places each item into the fullest bin with enough space (FF and BF open a new bin if no bin with enough empty space exists). Johnson [8] proved that both FFD and BFD algorithms have an approximation ratio of 11/9.

The resource augmented setting for the bin packing and related problems have also been studied in several previous works [3, 9–12]. In particular, the makespan scheduling problem can be considered as a resource-augmented variant of the bin packing problem (see, e.g., [13–15]).

In the online setting, the simplest algorithm, called Next-Fit (NF), maintains only one open bin at each time. When packing an item x, if there is enough space for x in the open bin, Next-Fit places x in the open bin; otherwise, it closes the current open bin and opens a new one. NF has a competitive ratio of 2 [8]. Most other algorithms can be categorized into two general groups. The first group, called Any-Fit algorithms, are greedy algorithms that never open a new bin for an incoming item if there is enough space in at least one of the existing bins. A subgroup of Any-Fit algorithms, called Almost-Any-Fit algorithms, treat items like any Any-Fit algorithm with an extra constraint: they do not pack an item into the emptiest bin (unless there is no other choice). FF and BF are both Almost-Any-Fit algorithms. Johnson [8] and Johnson et al. [16] proved that the competitive ratio of all Almost-Any-Fit algorithms is 1.7.

The second group of online algorithms, known as the Harmonic-Family of algorithms, classify items by their sizes and pack items that belong to each class separately from items of other classes. In the most basic version of these algorithms [17], we have K classes: classes $C_1, C_2, \ldots, C_k$, where class C_i, for $1 \leq i \leq K-1$, contains items with sizes in the range $(\frac{1}{i+1}, \frac{1}{i}]$, and class C_K contains items with sizes in the range $(0, \frac{1}{k}]$. The algorithm uses Next-Fit to pack the items of each class, and has a competitive ratio that converges to 1.691 when K goes to infinity, which is better

than the competitive ratio 1.7 of Any-Fit algorithms. Many other algorithms have been designed based on the same idea. To improve the competitive ratio, the main strategy of these algorithms is to combine some smaller classes with larger ones so that the wasted space in bins of the larger classes, like $C_1 = (\frac{1}{2}, 1]$, is avoided. For example, Refined-Harmonic breaks class $C_1 : (\frac{1}{2}, 1]$ into two sub-classes $C_{1a} : (\frac{1}{2}, \alpha]$ and $C_{1b} : (\alpha, 1]$, and class $C_2 : (\frac{1}{3}, \frac{1}{2}]$ into $C_{2a} : (\frac{1}{3}, 1 - \alpha]$ and $C_{2b} : (1 - \alpha, \frac{1}{2}]$, where $\alpha \in (\frac{1}{2}, \frac{2}{3})$. Then, it considers a specific proportion, δ of the bins of class C_{1a} to pack items of subclass C_{1a} together with items of subclass C_{2a}. By optimizing parameters α and δ, Refined-Harmonic achieves a competitive ratio of 1.636 [17]. Modified-Harmonic [18] is another algorithm that combines items of class $C_1 : (\frac{1}{2}, \alpha]$ with items of the sub-classes from different classes (not limited only to class $C_2 : (\frac{1}{3}, \frac{1}{2}]$). Modified-Harmonic-2 [18] breaks the class $C_1 : (\frac{1}{2}, 1]$ into more than two subclasses (i.e., unlike Refined-Harmonic, it does not use only one parameter α), leading to further combination of items. These two algorithms achieve a competitive ratio of at most 1.61562 and 1.612, respectively [17]. Harmonic++ uses a more complicated combination of items, improving the previous bounds by providing a competitive ratio of 1.588 [19].

All harmonic algorithms introduced so far belong to a general framework called Super-Harmonic for which Seiden [19] proved a lower bound of 1.58333. Heydrich and van Stee [20] introduced a new framework called Extreme-Harmonic and claimed a competitive ratio of 1.5813 for algorithms that belong to it. Unlike previous harmonic algorithms that reserve a space equal to the the largest item in the class, Extreme-Harmonic uses the exact size of the items in the range $(\frac{1}{3}, \frac{1}{2}]$ to combine with items of size larger than 1/2. Later, however, the correctness of the claimed competitive ratio for this algorithm was questioned by Balogh et al. [21]. They also introduced a new algorithm, called Advanced-Harmonic, which, in addition to using the exact size of some items, allows the items of the smallest class to be placed together with the items of all larger classes. The competitive ratio of this algorithm is at most 1.57829 [21], which is the best existing upper bound for the online bin packing problem.

In terms of lower bounds, Balogh et al. [22] showed that no online bin packing

algorithm can have a competitive ratio smaller than 1.54278. Thus, there is still a gap of around 0.04 between the best existing upper bound and lower bound. As a result, the online bin packing problem is still recognized as an open problem in terms of the asymptotic competitive ratio.

2.2 Two-dimensional bin packing

There are many ways to extend bin packing to higher dimensions (see [23] for a survey). Packing square items into square bins, called the *square packing problem*, is perhaps one of the most straightforward extensions. In this problem, the goal is to place a multiset of square items of various side lengths in (0,1] into a minimum number of square bins of uniform side length 1 such that two square items placed in the same bin do not overlap (but they possibly touch each other).

In the offline setting, the square packing problem is known to be NP-hard [24]. Bansal et al. [25] provided an APTAS for this problem (indeed, for the more general d-dimensional cubes). In the online setting, the best existing upper bound [26–29] and lower bounds [26–28, 30, 31] have been improved a few times. The best existing algorithm has a competitive ratio of 2.1187 [29], while the best existing lower bound is 1.7515 [31].

A generalization of square packing assumes that the items are rectangles with width and lengths in $(0, 1]$, while the bins are still unit squares. This problem has also been studied extensively (see [23] for details). In some variants, rectangles are allowed to be rotated by exactly ninety degrees (see, e.g., [32]). Another two-dimensional variant of the bin packing problem concerns packing equilateral triangles into unit square bins in an online manner. For this variant, Kamali et al. [33] provided an algorithm that applies the same idea as the Harmonic family algorithms for the classic bin packing and achieves a competitive ratio of 2.474, and proved that no online equilateral triangle packing algorithm can achieve a competitive ratio better than 1.509.

2.3 Fault-tolerant bin packing

Server consolidation is an application of the online bin packing problem in the Cloud. Service providers such as Amazon EC2 [34] or Microsoft Azure [35] host client applications, called *tenants*, on their servers. Upon the arrival of a tenant, an online algorithm assigns it to a server that has enough resources available for the tenant. For each tenant, the service provider commits to a Service Level Agreement (SLA) that specifies the minimum performance requirement for the tenant [36]. In particular, different tenants have various loads that are indicated in their SLAs. The goal of the service provider is to satisfy the SLA requirements while minimizing operational costs. To achieve this goal, Cloud providers often *consolidate* tenants on shared computing machines to improve utilization [37]. Such consolidation can be modeled via online bin packing, where each item represents a tenant and each bin represents a computing machine (a server).

In Cloud systems, the SLA requires the service to be available and uninterrupted at all times. Servers fail on a regular basis, however, and if a tenant is placed only in a single server, the failure of that server interrupts the service. To avoid this situation, client applications are often replicated in multiple servers. In order to maintain a *fault-tolerant packing* where the services stay uninterrupted against the failure of up to f services, it is necessary to replicate each tenant in at least $f+1$ different servers. In the case of database tenants, each tenant has a *primary* replica, which handles read/write queries, and multiple *standby* replicas, which act as backup replicas in anticipation of server failures. Naturally, the computational resources required for hosting primary replicas are more than that of standby replicas [38].

The following two models have been studied for fault-tolerant bin packing:

1. Uniform-replica model: In this model, the load l of an arriving item is evenly divided among all its replicas. In other words, to tolerate against failure of up to f servers, $f+1$ replicas of the same loads $l/(f+1)$ are maintained. This model assumes that the bin failures occur only after all items have been packed. The first online algorithm proposed for this model is called the Mirroring algorithm [39]. The algorithm is designed to tolerate against failure of only one server. To do so, it considers two replicas for each item (a blue replica and a red replica), each with a

load equal to half of the load of the original job. As a result, the load of each replica is in the range $(0, \frac{1}{2}]$. The algorithm packs the blue and red replicas separately using the Next-Fit algorithm while considering a capacity of 1/2 for each bin. When a server fails, say a bin with red replicas fails, then the load of all its replicas will be transferred into its corresponding mirroring bin with blue replicas. Given that the capacity of the bin is assumed to be 1/2 at the time of packing replicas, the total size of any pair of bins remains at most 1. Despite its simplicity, which makes it desirable in practice, the Mirroring algorithm is not efficient because it wastes half of the capacity of all bins. Daudjee et al. [40] analyzed this algorithm and showed that it has a competitive ratio in the range $[2.6, 3]$. They also proposed a new more efficient algorithm called Horizontal-Harmonic (HH), which works based on the Harmonic algorithm for classic bin packing [40]. More precisely, it considers different classes of items in terms of their sizes, and then packs each class separately, using Next-Fit, such that each bin maintains an empty space for one item of the class (which is used when a load is redirected from failed bins). Besides, items are packed such that no two bins of a class share more than one item. This guarantees a valid packing for every input. Dadujee et al. [40] showed that the competitive ratio of HH converges to 1.59 for an arbitrarily large number of classes. They also showed that no online algorithm can have a competitive ratio better than 10/7 for the fault-tolerant server consolidation problem.

2. Primary-standby model: This model was introduced in 2017 [41] and extends the previous model (with replicas of equal loads) to a more practical model that considers a real-world design assumption. In this model, each tenant is replicated into multiple servers, that is, $f + 1$ replicas of each tenant are maintained for handling up to f possible server failures. As before, it is assumed that all items are first packed, and then the bin failures happen. Moreover, one replica of each item is designated as the *primary replica* and the remaining replicas as *standby replicas*. In the database tenant application, the read queries can be answered by any replica, while the write queries should be handled by the primary replica before being propagated to other replicas. As such, the load of the primary replica is more than that of the standby replicas. More precisely, a tenant of workload x has a primary

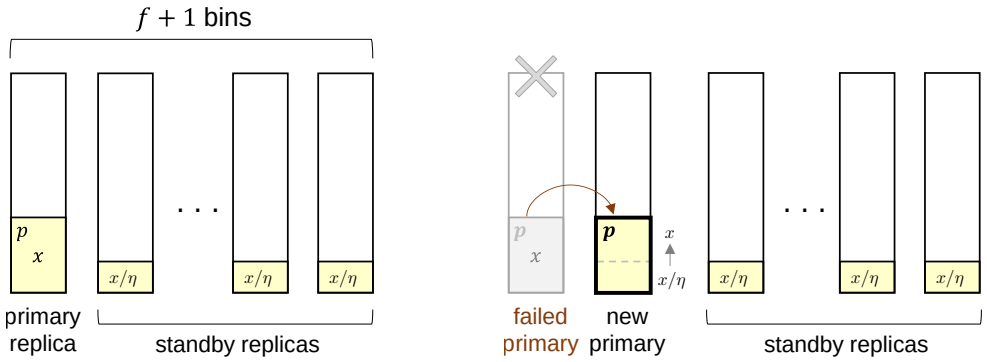


Figure 2.1: An illustration of the Primary-standby model. (a) A tenant of workload x has one primary replica of size x and f standby replicas of size x/η where $\eta > 1$. (b) When a server hosting some primary replicas fails, for each failed primary replica in the server, a standby replica is selected as the new primary replica, and its size increases from x/η to x.

replica of size x and f standby replicas of size x/η where $\eta > 1$ (see Figure 2.1a). When a server with primary replicas fails, for each failed primary replica in the server, a standby replica should be selected as the new primary replica, increasing the load of the selected replica from x/η to x (see Figure 2.1b). Consequently, the placement of replicas should guarantee that no server overflow takes place after transferring the load of the failed primary replicas into the new primary replicas. Li and Tang [42] used the ideas from the two algorithms introduced for the previous model, namely Mirroring and HH algorithms, and proposed two online algorithms called Mirroring and Harmonic-Shifting. They proved a competitive ratio of at most $(2\eta(1+f))/(\eta+f)$ for the Mirroring algorithm, and 2 for Harmonic-Shifting. Here, η denotes the ratio between the loads of primary and standby replicas. Later, in [43], an algorithm with improved *average-case* performance was presented. Our focus in this thesis, however, is on the worst-case performance, that is, algorithms with improved competitive ratios.

Chapter 3

Problem Statement and Contribution

In this chapter, we provide a formal definition of the three main problems that we study and provide the motivation behind them. For each problem, we will briefly review our contribution as well. We first consider the square packing problem with rotation, a variant of the square packing problem (as discussed in Section 2.2), under the offline setting (Section 3.1) and under the online setting (Section 3.2). We then visit the primary-standby model of the fault-tolerant bin packing problem in Section 3.3. Due to the online nature of its applications, we will consider this problem only under the online setting.

3.1 Offline square packing with rotation

Two-dimensional bin packing problems (where both items and bins are two-dimensional shapes) have many applications in practice. One application is cutting stock, where the bins represent the stock (e.g., wood boards), and items are demanded pieces of stock of different sizes. An algorithm has to cut the stock to provide the pieces that match the requests. This cutting process is equivalent to placing items into bins. Note that the goal of cutting stock is to minimize the amount of stock cut. When the pieces of stock and demanded pieces are square-shaped, the cutting stock is consistent with the objective of the square packing problem. Recall that, in the

square packing problem, the goal is to place a multi-set of square items of various sizes into a minimum number of square bins of equal size, where items have various side lengths of at most 1, and bins have uniform side length 1.

As reviewed in Section 2.2, despite being studied previously, the existing models for the problem do not allow rotation of items, that is, the sides of all items should be parallel to the edges of the bins. While this assumption makes combinatorial analysis of the problem easier, it comes at a price. Consider, for example, an instance of the problem formed by n items of size $\sqrt{2}/(2\sqrt{2}+1) \approx 0.369$. If we do not allow rotation, any bin can include at most 4 items, which gives a total cost of $\lceil n/4 \rceil$. Allowing rotation, however, 5 items fit in each bin and we can reduce the cost to $\lceil n/5 \rceil$ (see Figure 3.1). As a result, by using rotation, the number of required bins decreases by $n/20$, a notable saving in practice (e.g., for cutting stock applications).

Therefore, we consider the square packing problem in the presence of rotation. More formally:

Problem 3.1. *The input to the* ***(offline) square packing with rotation*** *problem is a multi-set of squares (items), defined with their side lengths* $\sigma = \{x_1, ..., x_n\}$*, where* $0 < x_i \leq 1$*. The goal is to pack these squares into the minimum number of squares of unit size (bins) such that no two items overlap (while they can touch each other). When placing a square into a bin, an algorithm can translate it to any position and rotate it by any degree, as long as the item remains fully inside the bin.*

We note that, in the offline setting, an algorithm has all the input items of σ in hand, and then pack them into some bins.

3.1.1 Contribution

We will show that the problem is NP-hard. We then study the problem under a resource augmented setting where an approximation algorithm can use bins of size $1+\alpha$, for some $\alpha > 0$, while the algorithm's packing is compared to an optimal packing into bins of size 1. Under this setting, we show that the problem admits an asymptotic polynomial time scheme (APTAS) whose solutions can be encoded in a poly-logarithmic number of bits.

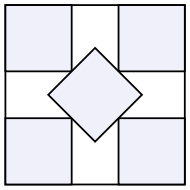


Figure 3.1: If all items in the input have length $\sqrt{2}/(2\sqrt{2}+1) \approx 0.35$, allowing rotation allows packing 5 items per bins instead of 4.

3.2 Online square packing with rotation

As mentioned in Section 3.1, cutting stock is one of the applications of the offline square packing problem. Unlike in the offline setting, in many practical applications, requests appear in an online manner, that is, one by one, and the stock should be cut without any prior knowledge about the future requests. Since the cutting process is irrevocable, such applications of the square packing problem have online nature. As a result, we consider the online square packing problem with rotation, which is defined as follows.

Problem 3.2. *In the **online square packing with rotation** problem, the input is a sequence $\sigma = \langle x_1, x_2, \ldots, x_n \rangle$ of squares that is revealed in an online manner. Each value of $x_i \in (0, 1]$ indicates the side length of a square that needs to be packed into square bins of unit size. When packing an item into a bin, an online algorithm can translate the item into any position in the bin and rotate it by any degree, as long as the item does not overlap other square items in the bin. The goal is to pack all squares in the input sequence into a minimum number of unit bins. The decisions of the algorithm are irrevocable and are made without knowing the values of $x_{t'}$ for $t' > t$.*

3.2.1 Contribution

We present an online algorithm that achieves a competitive ratio of at most 2.306. Our algorithm works by classifying squares based on their sizes and placing the squares of similar sizes tightly (and possibly using rotations) in the same bins. This approach is previously used to introduced different families of *Harmonic algorithms* for bin packing in both one dimension and higher dimensions. The presence of rotations, however, makes our classification and analysis different from the previous work.

In addition, we consider the following two related problems:

(i) a setting where the square item sizes are at most 1/2, and

(ii) a problem where items are right isosceles triangles (half-squares instead of squares), called "tans" [44], while bins remain unit squares.

For (i), we show that our algorithm (for the online square packing with rotation problem) achieves a competitive ratio of at most 1.732. For (ii), with a relatively similar approach, we introduce an algorithm of competitive ratio at most 1.897.

3.3 Online fault-tolerant bin packing

Server consolidation is widely used by the service providers in the cloud systems (see Section 2.3). There is usually a formal agreement between these service providers (e.g., Amazon) and clients (owners of tenants (items), e.g., the companies that own database tenants). Such an agreement requires the service to be available and uninterrupted at all times. In order to avoid interruptions in the case of server failures, each tenant should be replicated in multiple servers. Such a setting can be modeled by online fault-tolerant bin packing. Clearly, a good fault-tolerant bin packing algorithm that minimizes the number of used bins (in addition to ensuring the fault-tolerance) can help in decreasing the operational cost of a data center.

In Chapter 2, we reviewed two existing models for such settings, and observed that the primary-standby model is a more relevant model for certain applications. In this model, each item has one primary copy of workload x, and f standby copies with smaller workloads of x/η (for some $\eta > 1$). Li and Tang [42] presented a Harmonic-based algorithm named Harmonic-Shifting with a competitive ratio of at most 2, which is the best existing algorithm for the problem. The existing algorithms, in particular the Harmonic-Shifting algorithm, however, assume that an online sequence is first packed and *then* a set of up to f bins might fail. In practice, however, the packing is an ongoing process, and the servers might fail (and recover later) while the input is still being revealed.

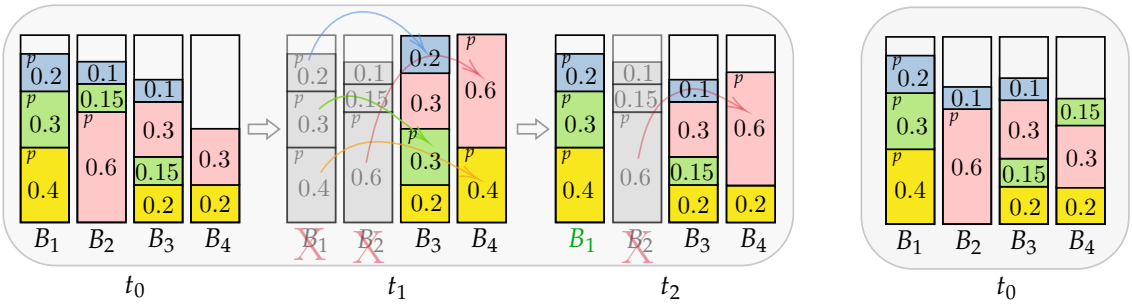
In this thesis, we study the fault-tolerant bin packing problem with the primary-standby scheme under a more relaxed and practical setting where bins can fail and

recover in an online manner during the execution of the algorithm so that at most f bins have failed at each given time. More precisely:

Problem 3.3. *In the* (f,η)***-fault-tolerant bin packing*** *problem, a sequence of* n *items, each having a size in the range* $(0,1]$*, is revealed in an online manner. When an item of size* x *arrives, an algorithm places a* primary replica *of size* x *and* f standby replicas*, each of size* x/η*, into bins of unit capacity, without any prior information about the forthcoming items. Throughout the packing process, some bins might fail and some of the previously failed bins might recover, so that the number of failed bins is at most* f *at any time. In a valid packing, a primary replica of each item should be always available. Therefore, upon failure of a bin with a primary replica of an item* x*, a standby replica of* x *(in a non-failed bin) needs to be selected and promoted to become the new primary replica of* x*. The subsequent increase in the size of the promoted standby replica (from* x/η *to* x*) should not cause an overload in any bin. The objective is to maintain a valid packing with a minimum number of bins.*

We assume that the packing of the $f+1$ replicas of any item takes place concurrently, that is, no bin fails when a group of $f+1$ replicas is being packed. Note that an algorithm can change the status of a replica from primary to standby and vice versa, but it cannot move replicas from one bin to another. In order to achieve a valid packing, the $f+1$ replicas of each item need to be packed in $f+1$ different bins; otherwise, failure of the up to f bins that contain all replicas of an item makes that item inaccessible.

Example 3.1. *Figure 3.2 illustrates an example instance of Definition 3.3. Each item of size* x *has a primary replica of size* x *and* $f = 2$ *standby replicas of size* x/η*, where* $\eta = 2.0$*. The packing (a) is a valid packing. The failure of any single bin or a pair of bins at any time can be addressed by promoting a standby replica into a primary replica without overloading any bin. For example, the arrows in the figure point to the standby replicas that are selected to become primary replicas after the simultaneous failure of bins* B_1 *and* B_2 *at time* t_1*. The packing (b), on the other hand, is not a valid packing of* σ*: if* B_1 *and* B_2 *fail, it is not possible to select standby replicas to replace the failed primary replicas without overloading a bin.*



(a) A valid solution: four items are packed at time t_0. Bins B_1 and B_2 fail at time t_1, and standby replicas in other bins replace primary replicas in the failed bins. B_1 recovers at t_2, and the algorithm gives back the primary status to replicas in B_1.

(b) An invalid solution at time t_0.

Figure 3.2: Packing sequence $\sigma = (0.4, 0.6, 0.3, 0.2)$, where $f = 2$ and $\eta = 2.0$. Replicas of the same items have the same color; primary replicas have the letter p.

We note that all possible inputs to the previous primary-backup model (as studied in [41]) are only special cases for our introduced model, where up to f bins fail *once* and only *after* all items have been packed. This implies that any algorithm for our model is also a valid algorithm for the previous model. Similarly, given that our model is a generalization of the previous model, an upper bound for the competitive ratio of any algorithm under our model extends to the previous model.

3.3.1 Contribution

For the fault-tolerant bin packing problem, as defined in Definition 3.3, we introduce an algorithm named Harmonic-Stretch that maintains fault-tolerant packings for an online sequence of items. We prove that Harmonic-Stretch has a competitive ratio of at most 1.75, which is an improvement over the competitive ratio 2 Harmonic-Shifting [45] for the previous primary-backup model. In summary, Harmonic-Stretch is designed to work in a more general setting, and yet achieves a better competitive ratio when compared to the previous algorithms.

Chapter 4

Offline Square Packing with Rotation*

In this chapter, we will study the offline square packing with rotation problem and present our results. As introduced in Section 3.1, an instance of the *offline square packing problem with rotation* is defined with a multi-set of *squares-items* of side lengths at most 1. The goal is to place these squares into a minimum number of unit *square bins* in a way that two squares-items placed in the same bin do not overlap, but potentially touch each other; unlike previous work on the square packing problem, when placing square items into square bins, the algorithm is allowed to rotate the items by any degrees (see Definition 3.1 for a formal definition). Since the problem is a variant of the classical bin packing problem, we refer to the square items simply as "items" and square bins as "bins". A square item can be recognized by its side length, which we refer to as the *size* of the item. Throughout, we refer to the number of bins used by an algorithm as the *cost* of the solution. As discussed in Section 3.1, allowing item rotations can lead to considerable savings in applications such as cutting stock where bins are identical size square-shaped pieces of stock (i.e., wood stock) and items are demanded square-shaped pieces of various sizes (see Figure 3.1 as an example).

In Section 4.1, we review a decision problem that asks whether a multi-set of squares can be placed into a single bin (where item rotations are allowed). We refer

*A summary of the results in this chapter has been in published in the Proceedings of the 14th International Conference on Combinatorial Optimization and Applications (COCOA'20) [46]

to this problem as the *1-Bin Square Packing* (1BSP) problem and show that it is NP-hard. It turns out that allowing rotation makes the problem much harder. It is not even clear how to answer the 1BSP problem when all square items have uniform size, as there has been little progress in the *congruent square packing problem* [47] (see Section 4.1.2 for details). The source of difficulty is that it does not appear that one can effectively discretize the problem and apply standard algorithmic techniques. A recent study [48] shows that similar packing problems are $\exists\mathbb{R}$-hard, that is, their verification algorithms need exponential bit-precision (see Section 4.1.3 for details). The same is likely true for the 1BSP problem.

In Section 4.2, we present our main result, which is an APTAS for the problem under a relaxed augmented setting. Precisely, given small, constant values for $\epsilon, \alpha > 0$, we present an algorithm that runs in polynomial time and packs any input σ in at most $(1+\epsilon)Opt(\sigma)+3$ augmented bins of size $1+\alpha$. Here, $Opt(\sigma)$ denotes the number of bins in an optimal packing of σ when packed into unit bins. The extra space given in the augmented bins enables free translation and rotation of squares, which ultimately allows packing items with encodable bit-precision.

4.1 A Review of the 1-Bin Square Packing (1BSP) Problem

Given a multi-set S of squares, the 1-Bin Square Packing (1BSP) problem asks whether items in S can be placed into a square of size c, called a *container*, where translation to any position and rotation by any degree is allowed. The 1BSP problem is a decision variant of the square packing problem introduced in Definition 3.1 (scale everything to ensure the container has size 1). In this section, we make some basic observations about the complexity of the 1BSP problem.

4.1.1 NP-Hardness

Many geometric packing problems are known to be NP-hard. Examples include packing squares into squares without rotation [24], packing circles into equilateral triangles [49], packing triangles into rectangles [50], and packing identical simple

polygons into a larger polygon [51]. Despite similarities, none of these results establish the hardness of the 1BSP problem. As such, we provide a proof to show the problem is NP-hard, even if the container and all items have integer sizes. We start with the following lemma:

Lemma 4.1. *Let S be a multi-set of squares with integer side lengths and of total area A, and let P be a rectilinear polygon of area A. It is possible to pack all squares in S into P only if no square is rotated.*

Proof. Assume S is fully packed into P, that is, P is fully covered by the squares in S. We show that no square is rotated in such packing. We start with a simple observation and then use an inductive argument.

Let r be an arbitrary convex vertex of P, that is, a vertex with an interior angle of 90°. Note that there are at least four convex vertices in any rectilinear polygon. Since P is fully covered by S, there should be a square $s \in S$ that includes or touches r. As Figure 4.1 illustrates, the only case where s remains fully inside P is when it has a vertex located at r and has no rotation (Figure 4.1(a)); in all other cases, a portion of s lies outside of P, which is not possible.

Given the above observation, we use an inductive argument on A to prove the lemma. For the base of induction, when $A = 1$, S is formed by one square of size 1 packed into a square P of size 1. Clearly, packing S into P involves no rotation. Next, assume we have a multi-set S of squares of total area k packed into a polygon P of area k. Consider an arbitrary convex vertex r of P. By the above observation, r is covered by a square $s \in S$ that is not rotated. Let an integer $x \geq 1$ denote the area of s. Removing the area covered by s from P results in a packing of the multi-set $S - \{s\}$ of squares into a smaller polygon P', where both $S - \{s\}$ and P' have area $k - x$. By the induction hypothesis, none of the squares in $S - \{s\}$ are rotated when packed into P'. Given that s was not rotated either, no square is rotated in the original packing of S into P. ■

Theorem 4.1. *The 1-Bin Square Packing (1BSP) problem (which allows rotation) is strongly NP-hard, even if the container and all items have integer sizes.*

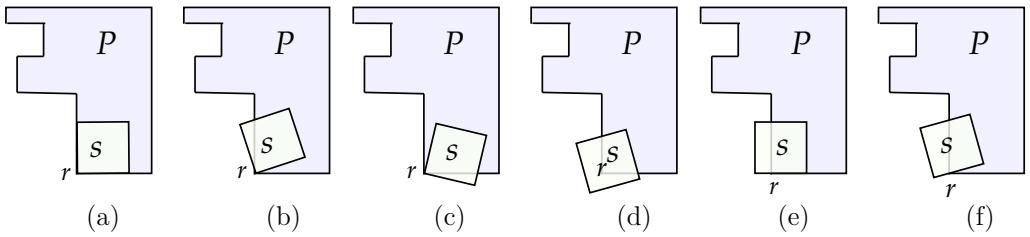


Figure 4.1: To pack and cover a rectilinear polygon with a multi-set of squares, no square should be rotated.

Proof. Given a multi-set S_0 of squares with integer side lengths and an integer c, Leung et al. [24] proved that it is strongly NP-hard to decide whether S_0 can be packed, without rotation, in a square container of side length c. Refer to this problem as the *1-bin square packing without rotation* (1BSW) problem. We provide a reduction from 1BSW to 1BSP.

Assume we are given an instance of 1BSW that asks whether a multi-set S_0 of squares can be packed into a square container B of side length c. Let u be an integer that denotes the total area of all items in S_0. Define an instance of the 1BSP problem through a multi-set S formed as the union of S_0 and $c^2 - u$ squares of unit size 1. We show that S can be packed into B with rotation if and only if S_0 can be packed into B without rotation.

Consider a $c \times c$ grid formed on B, where the grid vertices have integer coordinates. Assume S_0 can be packed into B without rotation; the total area of items in S_0 is u, and the empty area in B is $c^2 - u$. Since S_0 is packed without rotation, we can translate squares downward and towards the left such that their corners lie on the vertices of the grid (see the proof of Lemma 3.4 in [25] for details of such translation). The unused area in the resulting packing is formed by $c^2 - u$ grid cells of size 1×1. We can use the same packing and place the small squares in $S - S_0$ in the empty grid cells. This gives a valid packing of S. Next, assume S can be packed into B. Since the total area of items in S is exactly c^2, there is no empty (wasted) area in B. By Lemma 4.1, the packing does not involve rotation of any item. Since no rotation is involved, removing unit squares in $S - S_0$ from this packing gives a valid packing of S_0 (without rotation). ■

4.1.2 Congruent Square Packing

The congruent square packing problem, first studied by Erdős and Graham [47], asks for the minimum size $s(i)$ of a square that can contain i unit-sized squares. The problem is equivalent to finding the largest value of x such that i congruent squares of size x fit into a container of unit size. Clearly, an algorithm for the 1BSP problem can be used to find $s(i)$, using a binary search approach. Without rotation, it is easy to answer if a set of congruent squares fit in a square container. Allowing rotation, however, makes the problem harder. Despite being extensively studied (see, e.g., [52–55]), the value of $s(i)$ is not known for small values like $i = 11$. Figure 4.2 shows the packings that give the smallest known upper bounds for $s(11)$ and (18). We refer to the survey by Friedman [55] for more details on congruent square packing.

4.1.3 Existential Theory of the Reals

An Existential Theory of the Reals (ETR) formula can be stated as

$$\psi = \exists x_1, \ldots, x_n \;\; \Phi(x_1, \ldots, x_n),$$

where Φ is a well-formed sentence over alphabet $\{0, 1, x_1, \ldots, x_n, ., =, \leq, <, \wedge, \vee, \neg\}$ [12]. A decision problem belongs to the complexity class $\exists\mathbb{R}$-complete iff it is equivalent to deciding (in polynomial time) whether an ETR formula is true or not. In particular, these problems are $\exists\mathbb{R}$-hard in that a witness, which certifies the problem is in NP, might need an exponential number of bits in any numerical representation [12]. Abrahamsen et al. [48] have recently proved that a set of problems similar to the 1BSP problem, such as deciding whether a set of simple polygons fit into a square container, are $\exists\mathbb{R}$-hard. It is not clear, however, whether packing convex objects (in particular squares) into a square container is $\exists\mathbb{R}$-hard. Nevertheless, Erickson et al. [12] proposed to augment a square container in order to avoid (a potential) exponential bit precision for encoding the packings. They showed that if the container's size is augmented, where a slight perturbation is applied to the augmentation parameter (as in the smoothed analysis [56]), the bit precision for encoding a solution is expected to be logarithmic in the input size. Our APTAS in the next section

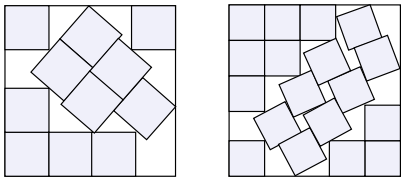


Figure 4.2: The smallest known container for placing 11 (left) and 18 (right) unit squares into a larger square [55].

uses the augmentation of bin sizes, along with some of the ideas from [12], to ensure logarithmic precision when packing squares into augmented bins.

4.2 An APTAS for Square Packing with Rotation

In this section, we describe an APTAS for placing square items, with rotation, into augmented square bins. The algorithm has two constant parameters α and ϵ, which are both small, positive values. The algorithm places any input σ into at most $(1+\epsilon)Opt(\sigma)+3$ augmented bins of size $1+\alpha$, where $Opt(\sigma)$ is the number of bins in an optimal packing of σ into unit squares. Throughout, we assume the total area of σ is arbitrarily large. Furthermore, we let $\xi = \epsilon/1081$.

4.2.1 Overview

The algorithm classifies items into *small*, *medium*, and *large* items. The classification is similar to the one in [25] for square packing without rotation. Medium and small items are smaller than ξ (large items can also be smaller than ξ). Medium items have a negligible total area. As such, we place them separately from others into a set of at most $\xi \cdot Opt(\sigma)+1$ *medium bins*.

We place large items into augmented *large bins*. For that, we round up item sizes, as suggested by de la Vega and Lueker [57], to get a constant number of possible item sizes. The resulting instance is then packed, using an exhaustive approach, into a minimum number of augmented bins. The extra space in the augmented bins enables us to discretize the problem, using the ideas from [12]. We show that the number of large bins will be no more than $(1+\xi)Opt(\sigma)$.

After placing large items, we partition the empty area in each large bin into a set of trapezoids and show that small items can be tightly packed into these trapezoids.

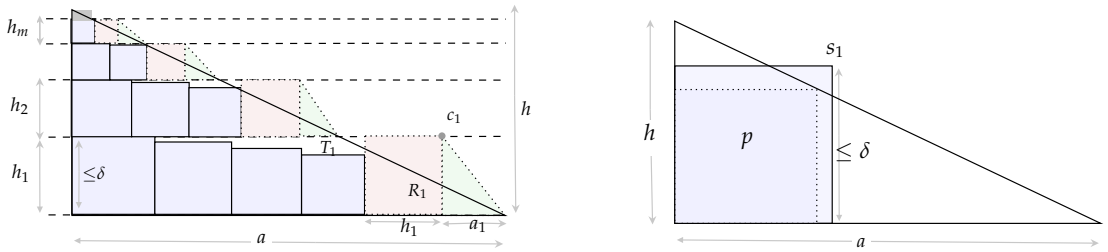
For that, we partition trapezoids further into right-angled triangles and use the Next-Fit-Decreasing-Height (Nfdh) algorithm of Coffman et al. [58] to pack these triangles. If all small items are placed in the trapezoids of the large bins, the resulting packing is almost-optimal, as the packing of large items is almost optimal (and no new bin is opened for small items). If some small items do not fit in the large bins, we place them, using the Nfdh strategy, into *small bins*. In this case, we prove that all bins are "almost full", which eventually gives the claimed guarantee.

In what follows, we first describe how small items can be packed into trapezoids, then we explain the placement of large items into augmented bins, and finally describe the packing of arbitrary inputs.

4.2.2 Triangle & trapezoid packing

Let T be a given right-angled triangle with two legs of size a and h (assume $h \leq a$). In what follows, we show how to pack a set of items of size at most δ into T without wasting too much area in T. In our solution, items are packed without rotation so that their sides are parallel to the legs of T. Later, we use this packing to show items of size at most δ can be packed into a trapezoid container, with rotation, so that not too much area is wasted.

In order to pack items into T, we use the Next-Fit-Decreasing-Height (Nfdh) strategy [58]. We sort items in non-increasing order of their sizes and place them one by one in the following manner. Without loss of generality, assume the legs of T extend along the x- and y-axes; we refer to the two legs as 'left leg' (of length h) and 'lower leg' (of length a). We place the first item in a way that it is tangent to the two legs of T. Let h_1 denote the size of the first item. The area within a distance h_1 from the lower leg of T forms a *shelf* of height h_1. We place subsequent items on this shelf such that the left side of each item touches the right side of the previous item, while its lower side touches the lower leg of T. At some point, the next item might not fit in the shelf; at this point, we "close" the shelf and recursively pack the remaining items in the right-angled triangle formed by removing the shelf from T (see Figure 4.3a). The algorithm stops when it cannot open a new shelf. Note that



(a) (b)

Figure 4.3: (a) Applying the Nfdh algorithm for packing a right-angled triangle T with squares of size at most δ. The algorithm places items in the non-increasing order of their sizes in shelves that are packed from left to right. The pink squares and the green triangles are respectively the covering squares and triangles. (b) When no item can be packed into T, the area of T is less than $(3h/2 + a/2)\delta$.

it is possible that no item is placed in T, which happens when the largest square in the input (the first square in the sorted order) does not fit in T (see Figure 4.3b).

Lemma 4.2. *Assume we apply Nfdh to pack a multi-set of items, each having a size at most δ, into a right-angled triangle T with legs of sizes a and h, where $h \leq a$. If the algorithm stops before packing all items (if not all items can be packed into T), then the wasted area in T is less than $(3.5a + h)\delta$.*

Proof. First, assume no item can be packed into T. Then placing the first square s_1 of the multi-set in T so that its two sides are tangent with the legs of T results in a part of s_1 lying outside of T (see Figure 4.3b). In this case, a square p that is tangent to the legs of T and touches its third side is fully contained in s_1 and, hence, has side length less than that of s_1 (consequently less than δ) and also less than h. So, the area of T can be partitioned into p (of area less than $h\delta$), a triangle above p (of area less than $h\delta/2$), and a triangle on the right of p (of area less than $a\delta/2$). Consequently, the total (wasted) area of T is less than $(3h/2 + a/2)\delta < (3.5a + h)\delta$.

Next, assume there are $m \geq 1$ shelves in the final packing (see Figure 4.3a). The wasted area in the ith shelf can be partitioned into two areas: the area on the right of the last item placed on the shelf (call it R_i), and the area on top of the squares placed on the shelf (call it T_i). R_i can be covered by two components: a *covering square* of area h_i^2 and (if required) a right-angled *covering triangle* of area $h_i a_i/2$, where a_i is

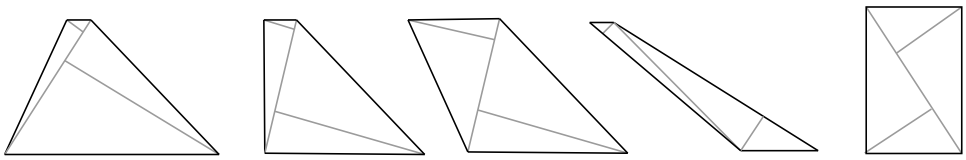


Figure 4.4: Any trapezoid can be partitioned into four right-angled triangles.

the base of the covering triangle (see Figure 4.3a). The bases of covering triangles of different shelves do not intersect when projected into the base of T. To see that, consider the top-right corner c_i of the covering square of shelf i. Since c_i appears outside T and just below shelf $i+1$, the covering triangle of shelf $i+1$ appears on the left of c_i, while the covering triangle of shelf i appears on its right. Consequently, given that $R_i < h_i^2 + h_i a_i/2 < \delta(h_i + a_i/2)$, and summing over all values of i, we can write $\sum_{i=1}^{m} R_i < \delta \sum_{i}^{m}(h_i + a_i/2) < \delta(h + a/2)$. For the wasted area on top of the ith shelf, we can write $T_i \leq a(h_i - h_{i+1})$; this is because all items placed on the shelf have a height at least h_{i+1}. Summing over all but the very last shelf, we get $\sum_{i=1}^{m-1} T_i \leq a \sum_{i=1}^{m-1}(h_i - h_{i+1}) = a(h_1 - h_m) < a\delta$. The wasted area on top of the last shelf is no more than $a\delta$, which is an upper bound for the size of the shelf. So, we have $\sum_{i=1}^{m} T_i < 2a\delta$. Finally, the unused area U on top of the whole packing (the dark area in Figure 4.3a) has a size of no more than $a\delta$. The total wasted area is thus $\sum_{i=1}^{m}(R_i + T_i) + U < (a/2 + h)\delta + 2a\delta + a\delta = (3.5a + h)\delta$. ∎

Lemma 4.3. *Let Z be a trapezoid in which every side has length at most x, and S be a multi-set of squares of size at most δ. There is an algorithm that packs items from S into Z such that either all items are packed into Z or a subset of S is packed while the wasted area in Z is at most $54x\delta$.*

Proof. First, we partition Z into four right-angled triangles. This can be done by partitioning Z into two triangles by drawing a diagonal. By the triangle inequality, any side of these two triangles has side lengths less than $2x$. Then, we partition each triangle into two right-angled triangles by drawing an altitude that lies inside the triangle (see Figure 4.4). In each of the four resulting triangles, one edge is within a side of the trapezoid (of the size at most x), another edge is within an edge of the two previous triangles (of the size of at most $2x$), and hence the last side is shorter than the sum of the other two, i.e., $3x$. Overall, each of the four triangles has sides

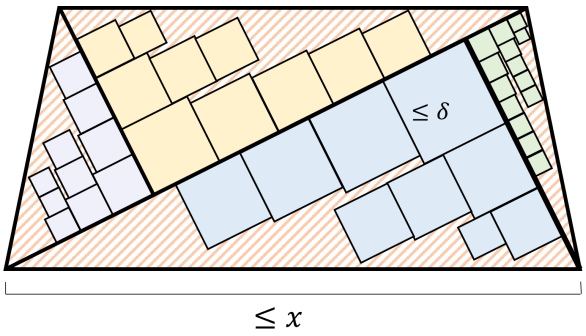


Figure 4.5: An illustration of packing squares of side sizes at most x into the resulted four right-angled triangles of a trapezoid with side sizes of at most δ by applying the introduced triangle NFDH algorithm. The striped red lines show the wasted area of the trapezoid.

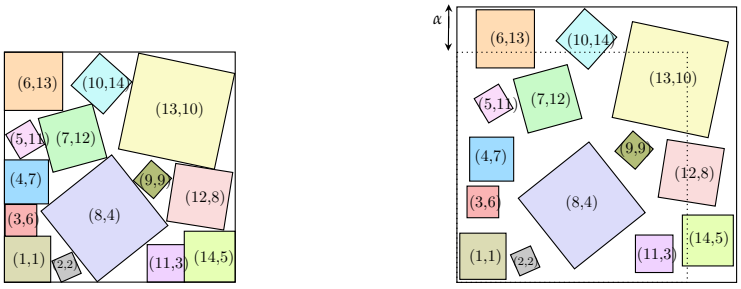
of side length no more than $3x$. So, we can apply Lemma 4.2 to pack any of the resulting right-angled triangles with items in S. The wasted area in each of these triangles is less than $(3.5 \times 3x + 3x)\delta = 13.5x\delta$. Consequently, the total wasted area in the trapezoid is no more than $4 \times 13.5x\delta = 54x\delta$. Figure 4.5 illustrates a final packing of a trapezoid after applying the NFDH algorithm on it to pack its four resulted right-angled triangles. ■

4.2.3 Packing large items

We explain how to pack large square items into augmented bins. The following lemma implies that augmenting bins enables us to encode the translation (position) and the rotation of items in a given packing, using logarithmic bit-precision. Erickson et al. [12] proved a more general statement about packing convex polygons. For completeness, we include their proof for square packing.

Lemma 4.4. *[12] Let S be a multi-set of m squares that can be packed into a unit bin. For any $\alpha > 0$, it is possible to pack S into an augmented bin of size $1 + \alpha$ such that the translation and rotation of each item can be encoded in $O(\log(m/\alpha))$.*

Proof. *[Sketch]* The extra space given by an augmented bin can be used to ensure all items can be placed at a distance of at least $\frac{\alpha}{m+2}$ from each other and the boundary of the bin, which enables free translation and rotation of squares to a certain encodable degree. Consider a packing of S into a unit bin. Form a partial



(a) packing into a unit bin (b) packing into an augmented bin

Figure 4.6: (a) A set of 14 square items that are tightly packed into a unit square. The numbers (i, j) for a square indicate its respective indices in π_x and π_y. (b) The packing of squares into an augmented bin of size $1 + \alpha$, where items are separated from each other by at least $\alpha/14$. A square with indices (i, j) is shifted by $(\frac{i\alpha}{16}, \frac{j\alpha}{16})$.

ordering of items along the x-coordinate in which, for items $a, b \in S$, we have $a < b$ iff there is a horizontal line that passes through both a and b and crosses a before b. Let π_x be a total ordering of items that respects the above partial ordering, and π_y be another ordering defined symmetrically based on a partial ordering through crossings of vertical lines. Let $a \in S$ be the ith element in π_x and jth element in π_y (we have $i, j \in \{1, \ldots, m\}$). In the augmented bin, we shift a towards the right by $\frac{i\alpha}{m+2}$ and upwards by $\frac{j\alpha}{m+2}$ (see Figure 4.6). At this point, any two squares are separated by at least $\frac{\alpha}{m+2}$. So, it is possible to shift items towards left or right by $O(\alpha/m)$ and/or rotate them by $O(\alpha/m)$ degrees such that squares still do not intersect. Consequently, there is a positioning of squares in the augmented bin in which the position (translation) and rotation of each square can be presented with $O(\log(m/\alpha))$ bit-precision. For details, see the proof of Corollary 26 in [12]. ■

Lemma 4.4 enables us to use an exhaustive approach for packing large items, provided that there is a constant number of item sizes.

Lemma 4.5. *Assume a multi-set S of n squares can be optimally packed into $Opt(S)$ unit bins. Assume all items in S have size at least $\delta > 0$, and there are only K possible sizes for them, where δ and K are both constants independent of n. There is an algorithm that packs S into $Opt(S)$ augmented bins of size $1 + \alpha$ in time $O(polylog(n))$, where $\alpha > 0$ is an arbitrary constant parameter.*

Proof. Since all items are of side length at least δ, the number of items that fit in a bin is bounded by $m = \lceil 1/\delta^2 \rceil$. Consider multi-sets of items described by vectors of form $(x_1, x_2, \ldots, x_K)$, where $1 \leq x_i \leq m$ denotes the number of items of type (size) i in the multi-set. We say a vector is *valid* if the multi-set that it represents can be packed into a unit bin. According to Lemma 4.4, if a vector V is valid, the multi-set of squares associated with it can be packed into an augmented bin of size $1+\alpha$ in a way that the exact translation and rotation of each square can be encoded in $O(\log(m/\alpha))$ bits. Since there are up to m squares in the bin and each has one of the K possible sizes, at most $C = O(m((\log K) + \log(m/\alpha)))$ bits is sufficient to encode how a multi-set associated with a valid vector should be packed into an augmented bin. Since K, α, and δ (and hence m) are constant values, C is also a constant. Therefore, if we check all 2^C possible codes of length C, we can retrieve all valid vectors and their packing into augmented bins in a (huge) constant time. In summary, we can create a set $\{T_1, \ldots, T_Q\}$, where each T_i is associated with a unique, valid vector together with its translation and rotation and is referred to as a *bin type*. Here, Q is the number of bin types and is a constant (since $Q \leq 2^C$). From the discussion above, each bin type has an explicit description of how a multi-set of items is placed into an augmented bin of size $1+\alpha$.

The remainder of the proof is identical to a similar proof from [25]. Let T_{ij} denote the number of squares of type j in a bin type T_i, and let n_j denote the number of squares of type j in the input. Furthermore, let y_i denote the number of bins of type T_i in a potential solution. The following integer programs indicate the values of y_i's in an optimal solution:

$$min \sum_{i=1}^{Q} y_i, \quad s.t. \qquad \sum_{i=1}^{Q} T_{ij} y_i \geq n_j \qquad \text{for } j = 1, \ldots, K,$$
$$y_i \geq 0; \quad y_i \in \mathbb{Z} \qquad \text{for } i = 1, \ldots, Q,$$

This integer program has size $O(\log n)$ and a constant number of variables (recall that Q is a constant). So, we can use Lenstra's algorithm [59] to find its optimal solution in $O(polylog(n))$. Such a solution indicates how many bins of each given type should be opened, and as mentioned earlier, each bin type has an explicit description of placements of items into an augmented bin. ■

Provided with Lemma 4.5, we can use the standard approach of de la Vega and Lueker [57] to achieve an APTAS for large items.

Lemma 4.6. *Assume a multi-set S of n squares, all of the size at least $\delta > 0$, where δ is a constant independent of n, can be optimally packed in $Opt(S)$ unit bins. For any constant parameters $\alpha, \xi > 0$, it is possible to pack S into at most $(1+\xi)Opt(S)+1$ augmented bins of size $1+\alpha$ in time $O(polylog(n))$.*

Proof. We use the same notation as in [25]. Consider an arrangement of squares in S in non-increasing order of sizes. We partition S into $K = \lceil l/(\xi\delta^2) \rceil$ groups, each containing at most $g = \lceil n/K \rceil$ squares. Let J be a multi-set formed from S by rounding up the size of each square $s \in S$ to the largest member of the group that it belongs to. Let J' be a multi-set defined similarly except that each square is rounded down to the smallest member of its group.

In order to pack S, we use the described algorithm in Lemma 4.5 to pack J into $Opt(J)$ augmented bins of size $1+\alpha$, where $Opt(J)$ is the optimal number of unit bins that J can be packed to. Note that there are K different item sizes in J, and all have a size larger than δ, which means we can use Lemma 4.5 to pack J. Since squares in S are rounded up to form J, the same packing can be used to pack S. Note that it takes $O(polylog(n))$ to achieve this packing.

To analyze the packing, let $Opt(J')$ be the optimal number of unit bins that J' can be packed to. We note that $Opt(J') \leq Opt(S) \leq Opt(J)$. This is because items of S are rounded down in J' and rounded up in J. Let x denote the size of squares in group g of J', and y denote the sizes in group $g+1$ of J. Since y appears after x in the non-increasing ordering of squares in S, we can assert any square in group g of J' has a size no smaller than a square in group $g+1$ of J. Therefore, if we exclude the first group, any square in J can be mapped to a square of the same or larger size in J'. Because S is formed by n squares, each of area at least δ^2, we have $n\delta^2 < Opt(S)$. As a result, since there are $q = \lceil n/K \rceil \leq \lceil n\xi\delta^2 \rceil < \lceil \xi Opt(S) \rceil$ squares in the first group, we can conclude $Opt(J) \leq Opt(J') + q \leq Opt(S) + q \leq \lceil (1+\xi)Opt(S) \rceil \leq (1+\xi)Opt(S) + 1$. ∎

4.3 Packing arbitrary input

We use the results presented in Sections 4.2.2, 4.2.3 to describe an algorithm for packing arbitrary inputs. We start with the following lemma:

Lemma 4.7. *Assume a multi-set of m squares is packed into a bin. The unused area in the bin can be partitioned into at most $5m$ trapezoids.*

Proof. Create an arbitrary labelling of squares as $s_1, s_2, \ldots, s_m$. Let t, b, l and r respectively denote the topmost, bottom-most, leftmost, and rightmost points of a square s_i (break ties arbitrarily). Draw the following five horizontal line segments for s_i (see Figure 4.7a):

1,2) two line segments starting at l and b and extending towards the left until they touch another square or the left side of the bin.

3,4) two line segments starting at r and b and extending towards the right until they touch another square or the right side of the bin.

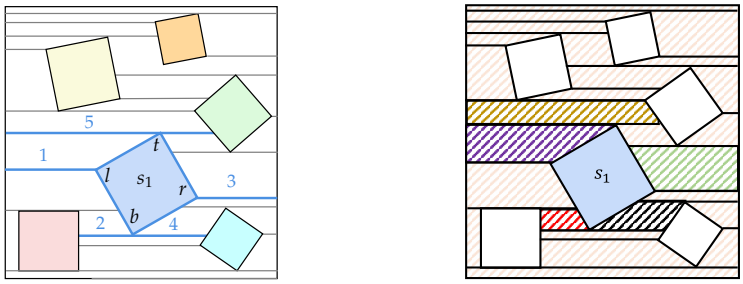
5) a line segment that passes through t and extends towards the left and right until it touches other squares or boundaries of the bin

We label the line segments of s_i with the label i. When we draw the line segments for all squares, the unused area in the bin will be partitioned into trapezoids. We label each trapezoid in the partition with the label of its lower base. So, for each square s_i, there will be at most five trapezoids with the label i (see Figure 4.7b). Consequently, the number of trapezoids will not exceed $5m$, which completes the proof. ■

In the above proof, we treated the topmost point t and bottom-most point b of squares differently. This is because the line passing b can be the lower base of two trapezoids on the two sides of the square, while the line passing t can be the lower base of one trapezoid on top of the square.

4.3.1 Item classification

Assume we are given an arbitrary input σ and small, constant parameters $\alpha, \epsilon > 0$. Recall that $\xi = \epsilon/1081$. Let $r = \lceil 1/\xi \rceil$, and define the following $r + 1$ classes



(a) (b)

Figure 4.7: (a) The five horizontal line segments drawn for each square to partition the unused area of a bin with m items into at most $5m$ trapezoids. (b) The five trapezoids associated with square s_1 after drawing the 5 lines corresponding to it.

for items smaller than ξ. Class 1 contains items with sizes in $[\xi^3, \xi)$, class 2 contains items in $[\xi^7, \xi^3)$, and generally class i $(1 \leq i \leq r+1)$ contains items in $[\xi^{2^{i+1}-1}, \xi^{2^i-1})$. Since there are $r+1$ classes, the total area of squares in at least one class, say class j, is bounded from above by $area(\sigma)/(r+1) \leq \xi \cdot area(\sigma)$, where $area(\sigma)$ is the total area of the squares in σ. We partition square-items in the input σ into *large*, *medium*, and *small* items as follows. Medium items are the members of class j, that is, items with size in $[\xi^{2^{j+1}-1}, \xi^{2^j-1})$. Large items are items of size at least ξ^{2^j-1}, and small items are items of size less than $\xi^{2^{j+1}-1}$.

4.3.2 Packing algorithm

We are now ready to explain how to pack items in σ. Medium items are placed separately from other items into unit bins. For that, we apply the Nfdh algorithm of [58] to place medium items into *medium bins* without rotation (see Figure 4.8a). We use m_m to denote the number of resulting medium bins. We apply Lemma 4.6 to pack the multi-set L of large items into augmented bins of size $1+\alpha$. We refer to these bins as *large bins* and use m_l to denote the number of large bins. It remains to pack small items. We use Lemma 4.7 to partition the empty area of any large bins into a set of trapezoids. We pack small items into these trapezoids. For that, we consider an arbitrary ordering of trapezoids and use the Nfdh strategy (as described in Section 4.2.2) to place items into the first trapezoid (after partitioning it into four right-angled triangles). If an item does not fit, we close the trapezoids and consider the next one. The closed trapezoids are not referred to again. This process continues

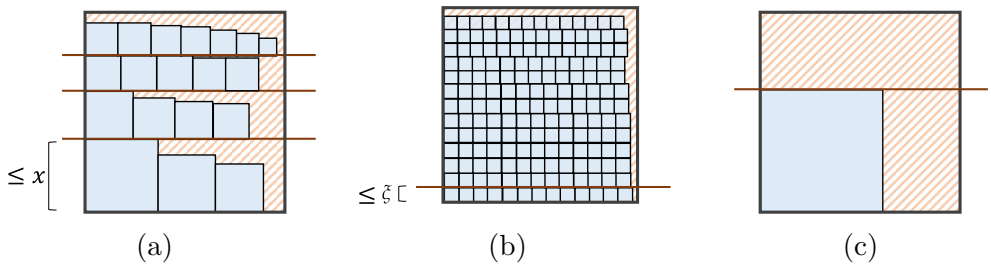


Figure 4.8: (a) An illustration of NFDH algorithm [58]: it places items in the non-increasing order of their sizes in shelves that are packed from left to right. The horizontal red lines show the shelves, and the diagonal red lines illustrate the wasted area of the bin, which is at most $2x$ where x is the largest side size of the squares inside the bin [58]. (b) When we apply NFDH on relatively small items (for example, the small and medium items in our classification, which have sizes smaller than ξ), the bins are almost full, and the wasted area of each bin is small. (c) Instead, if we apply NFDH on relatively large items, the wasted area of the bin is large.

until either all small items are placed into trapezoids or all trapezoids are closed. In the latter case, the remaining small items are placed using the NFDH strategy of [58] into new bins that we call *small bins*.

4.3.3 Analysis

When we apply NFDH to place square items of size at most ξ into square bins (without rotation), the wasted area in each bin is at most 2ξ [58] (see Figure 4.8b). Since all medium items have size at most $\xi^3 < \xi$ (for $\xi < 1$), the wasted area in each medium bin will be at most 2ξ. On the other hand, the total area of medium items is at most $\xi \cdot area(\sigma)$. So, the total number m_m of medium bins is at most $\frac{\xi \cdot area(\sigma)}{1-2\xi} + 1$, which is at most $\xi \cdot area(\sigma) + 1$ for $\xi < 1/4$. Note that $area(\sigma)$ is a lower bound for the optimal number of unit bins for packing σ. So, $m_m \leq \xi Opt(\sigma) + 1$. We consider two cases for the remainder of the analysis.

- Case I: Assume the algorithm does not open a small bin. By Lemma 4.6, the number of large bins m_l is no more than $(1+\xi)Opt(L) + 1$. Clearly, $Opt(L) \leq Opt(\sigma)$ and we can write $m_l \leq (1+\xi)Opt(\sigma) + 1$. For the total number of bins in the packing, we can write $m_m + m_l \leq (1+2\xi)Opt(\sigma) + 2$.

- Case II: Assume the algorithm opens at least one small bin. We show that the area of all bins (except possibly the last small bin) is almost entirely used. Large items

are of size at least ξ^{2^j-1} and area at least $\xi^{2^{j+1}-2}$. So, the number of large items in each large bin is at most $\frac{1}{\xi^{2^{j+1}-2}}$. By Lemma 4.7, the number of trapezoids in each bin is at most $\frac{5}{\xi^{2^{j+1}-2}}$. Given that bins are augmented (with a size of $1+\alpha$), any trapezoid has side length at most $\sqrt{2(1+\alpha)^2} < 2(1+\alpha)$. Since we pack small items of size at most $\delta = \xi^{2^{j+1}-1}$ inside these trapezoids, by Lemma 4.3, the wasted area in each trapezoid is less than $54\xi^{2^{j+1}-1} \times 2(1+\alpha)$. Summing up over all trapezoids, the wasted area in each large bin is at most $\frac{5}{\xi^{2^{j+1}-2}} \times \xi^{2^{j+1}-1} \cdot 108(1+\alpha) = 540\xi(1+\alpha)$.

So, any large bin includes squares of total area at least $(1+\alpha)^2 - 540\xi(1+\alpha) > 1 - 540\xi$, assuming $\xi < 1/270$. Moreover, since packed by NFDH, all small bins (except potentially the last one), have a filled area of at least $1-2\xi > 1-540\xi$. In summary, with the exception of at most one bin (the last bin), any large or small bin includes items of total area at least $1-540\xi$. As such, for $\xi < 1/1080$, we can write $m_l + m_s \leq \lceil area(\sigma)/(1-540\xi) \rceil + 1 \leq area(\sigma)(1+540\xi/(1-540\xi)) + 2 \leq (1+1080\xi) \cdot area(\sigma) + 2 \leq (1+1080\xi) \cdot Opt(\sigma) + 2$. Adding the number of medium bins, the total number of bins will be at most $(1+1081\xi) \cdot Opt(\sigma) + 3$.

Recall that we have $\xi = \epsilon/1081$. So, given any $\epsilon < 1$, the number of bins in the resulting packing will be at $(1+\epsilon) \cdot Opt(\sigma) + 3$. Our algorithm's time complexity is dominated by the sorting process used for classifying items and packing small items. As such, the algorithm runs in $O(n \log n)$. We can conclude the following:

Theorem 4.2. *Assume a multi-set σ of n squares can be optimally packed in $Opt(S)$ unit bins. There is a polynomial-time algorithm that, for any constant $\alpha > 0$ and $\epsilon \in (0,1)$, packs S into at most $(1+\epsilon)Opt(\sigma) + 3$ augmented bins of size $1+\alpha$.*

Chapter 5

Online Square Packing with Rotation

Recall that in the square packing problem, the goal is to place a multiset of square items of different side lengths $\in (0, 1]$ into a minimum number of square bins of uniform side length 1. As opposed to the offline setting, where all square items are given in advance, in the online setting, the multi-set of items forms a sequence which is revealed in an online and sequential manner. When an item is revealed, an online algorithm has to place it into a square bin without any priory knowledge of the forthcoming items (see Section 3.2 for a formal definition). As we discussed earlier, all existing results in both offline and online settings are restricted to the case when square items are placed orthogonally to the square bins. We presented our results for the offline setting in presence of item rotations in Chapter 4. In this chapter, we study the problem in the online setting. We provide an algorithm that achieves an asymptotic competitive ratio of 2.306 when square items have sizes in the range $(0, 1]$, and a better asymptotic competitive ratio of 1.732 when item sizes are in the range $(0, 1/2]$. We also study another problem where items, instead of being squares, are isosceles right triangles (half-square triangles) called "tans", and present an online algorithm with an asymptotic competitive ratio of at most 1.897.

j	$u(j)$	Optimal?	j	$u(j)$	Optimal?
1	$= 1.0$	optimal	19	≈ 0.2047	best-known
2-4	$= 0.5$	optimal	20-22	$= 0.2$	best-known
5	≈ 0.3694	optimal	23-25	$= 0.2$	optimal
6-9	≈ 0.3333	optimal	26	≈ 0.1779	best-known
10	≈ 0.2697	optimal	27	≈ 0.1752	best-known
11	≈ 0.2579	best-known	28	≈ 0.1716	best-known
12-13	$= 0.25$	best-known	29	≈ 0.1685	best-known
14-16	$= 0.25$	optimal	30-33	≈ 0.1667	best-known
17	≈ 0.2139	best-known	34-36	≈ 0.1667	optimal
18	≈ 0.2073	best-known			

Table 5.1: Optimal or best-known $u(j)$ values for $1 \leq j \leq 36$ when the goal is to pack j identical squares of the largest size $u(j)$ into a unit square. Values of u(j) are the scaled values of the known results on congruent square packing [60].

5.1 Square-Rotate algorithm

In this section, we introduce our square packing algorithm called Square-Rotate.

5.1.1 Item classification

Similar to the Harmonic family of algorithms, we classify squares by the size of their side lengths (which we simply refer to as the "size" of the items). Square-Rotate packs squares of each class separately from other classes.

In total, there are 13 classes of squares. Square items with sizes in the range $(0, 1.1752]$ are in class 13. We refer to class 13 as the "tiny class", and items that belong to this class are referred to as tiny items. We refer to items that belong to class $i \in [1, 12]$ as "regular items". For each class $i \in [1, 12]$, the range of items in the class is specified as $(x_i, x_{i-1}]$ (for convenience, we define $x_0 = 1$). The values of x_i's are defined in a way that a certain number of items, denoted by S_i, of class i can fit in the same bin. The specific range of item sizes for each class $i \in [1, 12]$ and values of S_i is derived from the best-known or optimal results [60] on the congruent square packing problem [47], which asks for the minimum size $c(j)$ of a square that can contain j unit-sized squares (see Section 4.1.2). A scaling argument, where the container size is fixed to be 1, gives $u(j)$ values when the goal is to pack j identical squares of maximum size $u(j)$ into a unit square. Table 5.1 provides the scaled best-known/optimal $u(j)$ values for $1 \leq j \leq 36$. These scaled numbers gives the specific

Class	Side length x	S_i	Occupied Area	Weight	Density
1	$(0.5000, 1.0000]$	1	$> 1(0.250)$=0.250	1	< 4.000
2	$(0.3694, 0.5000]$	4	$> 4(0.136)$=0.544	1/4	< 1.838
3	$(0.3333, 0.3694]$	5	$> 5(0.111)$=0.555	1/5	< 1.801
4	$(0.2697, 0.3333]$	9	$> 9(0.072)$=0.648	1/9	< 1.543
5	$(0.2579, 0.2697]$	10	$> 10(0.066)$=0.660	1/10	< 1.515
6	$(0.2500, 0.2579]$	11	$> 11(0.062)$=0.682	1/11	< 1.466
7	$(0.2139, 0.2500]$	16	$> 16(0.045)$=0.720	1/16	< 1.388
8	$(0.2073, 0.2139]$	17	$> 17(0.042)$=0.714	1/17	< 1.400
9	$(0.2047, 0.2073]$	18	$> 18(0.041)$=0.738	1/18	< 1.355
10	$(0.2000, 0.2047]$	19	$> 19(0.040)$=0.760	1/19	< 1.315
11	$(0.1779, 0.2000]$	25	$> 25(0.031)$=0.775	1/20	< 1.290
12	$(0.1752, 0.1779]$	26	$> 26(0.030)$=0.780	1/26	< 1.282
13	$(0, 0.1752]$		> 0.702	$1.425x^2$	≈ 1.425

Table 5.2: A summary of item classification and details on item weights and densities, as used in the definition and analysis of Square-Rotate.

ranges that we used for classifying items as follows: Items of class 1 have sizes in the range $(1/2, 1]$, and we have $x_1 = 1/2$. Note that exactly $S_1 = 1$ item of class 1 can fit in the same bin. For $i \in [2, 12]$, S_i is the number of items of size x_{i-1} that fit in the same bin. For example, for $i = 2$, we have $S_2 = 4$ because $x_1 = 1/2$, and 4 items of size $1/2$ fit in the same bin. Moreover, x_i is defined as the largest value so that $S_i + 1$ items of size x_i cannot fit in the same bin. For example, we have $x_2 = 0.3694$ because, according to Table 5.1, $S_2 + 1 = 5$ squares of size 0.3694 cannot fit in the same bin.

The respective range of items for each class, as well as the values of S_i, is presented in Table 5.2. For example, a square is in class 1, 2, or 12 if its side size is in the interval $(0.5, 1]$, $(0.3694, .5]$, or $(0.1752, 0.1779]$, respectively. In Figure 5.1, it is specified how S_i items of the largest size in class i can fit into a square bin. We refer to [60] for details on the unit square packing problem.

5.1.2 Packing regular items

For each class i $(1 \leq i \leq 12)$, the algorithm has at most one active bin of type i. When a bin of type i is opened, it is declared as the active bin of the class, and S_i square "spots", each of which having a size equal to the largest square of class i, are reserved in the bin. Upon the arrival of an item of class i, it is placed in one of the

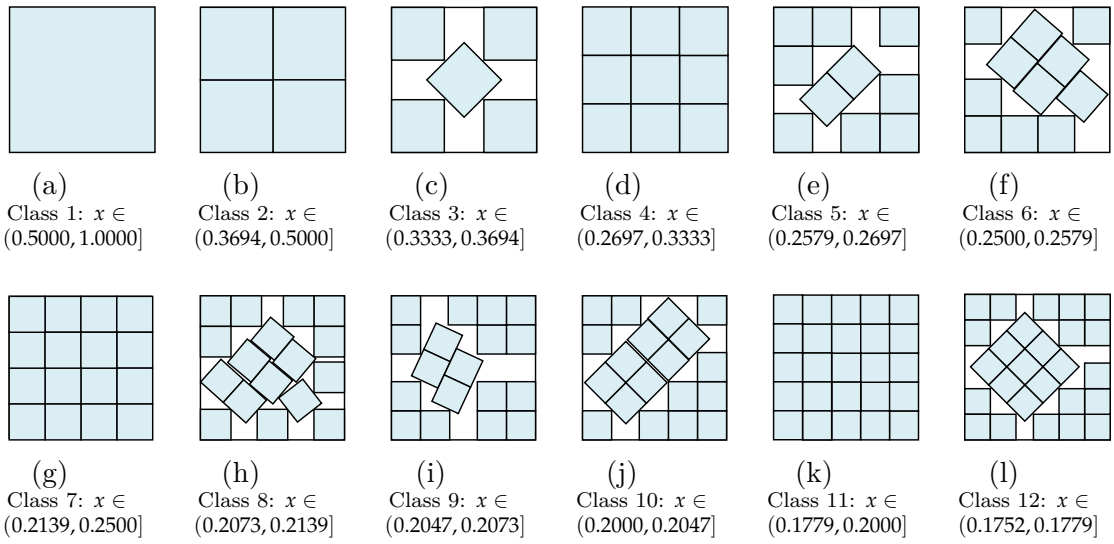


Figure 5.1: Placement of regular square items of class $i \in [1, 12]$ in their respective bin. It is possible to pack i square items of class i into a single square bin [60].

S_i spots of the active bin. If all these spots are occupied by previous items, a new bin of type i is opened. This ensures that all bins of type i, except potentially the current active bin, include S_i items.

5.1.3 Packing tiny items

For the last class, i.e., tiny items, the algorithm uses a different approach, proposed by Epstein and van Stee [61]. Briefly, it maintains at most one active bin for placing tiny items. When a bin is opened for these items, the algorithm reserves four square spots of size 1/2, i.e., the four squares of class 2 in Figure 5.1b. These square spots are used as bins for placing tiny items. Then, the algorithm chooses one of the innermost sub-bin squares that has enough space for the arrived item, and repeats the procedure for the selected sub-bin until it cannot split any of the innermost sub-squares into four new ones with enough space for the item. At this step, the item is placed in one of those smallest sub-bins. When the next item arrives, if there is a sub-bin of the smallest possible size in which the item can fit, the algorithm places the item in that spot. Otherwise, the algorithm finds the smallest sub-bin that can fit the item and repeats the previous procedure to split it into the smaller sub-bins to reach an appropriate spot for the item. If no sub-bin with enough empty space

is available in the bin, the algorithm closes the current bin and opens a new empty active bin for the item and applies the whole process from the beginning (see [61], for details). Note that the algorithm does not rotate any of the tiny items to pack them. Epstein and van Stee proved the following result, which we will use in our analysis later.

Lemma 5.1. *[61] Consider the square packing problem (without rotation) in which all items are of size at most* $1/M$ *for some integer* $M \geq 2$*. There is an online algorithm (as described above) that creates a packing in which all bins, except possibly one, have an occupied area of size at least* $(M^2-1)/(M+1)^2$*.*

5.1.4 Algorithm's analysis

In this section, we prove a competitive ratio of at most 2.306 for our algorithm. We use a *weighting function argument*. For each item of size x, we define a weight $w(x) \geq x$ for the item and prove that: (1) the total weight of square items in each bin of the algorithm, except potentially a constant number of them, is at least 1, and (2) the total weight of items in each bin of an optimal packing is at most 2.306. If $w(\sigma)$ denote the total weight of items in an input sequence σ, then (1) implies that the number of bins opened by the algorithm is at most $w(\sigma)+c$, for some constant value of c, and (2) implies that the number of bins in an optimal packing is at least $w(\sigma)/2.306$. Therefore, the (asymptotic) competitive ratio of the algorithm would be at most 2.306.

Recall that all bins opened for squares of class i $(1 \leq i \leq 12)$, except possibly the last active bin, include S_i squares. We define the weight of items of class i to be $1/S_i$. This way, the total weight of items in bins opened for all squares of classes 1 to 12, except possibly 12 of them (the last bin from each class), is exactly 1. Therefore, (1) holds for bins opened for regular items.

We define the weight of a tiny square of size x as $x^2/0.701 (= 1.425x^2)$. All tiny items are of size at most 0.1752. Therefore, by Lemma 5.1, the occupied area of all bins opened for tiny items (except possibly one of them) will be at least 0.701. This implies their total weight is at least $0.701/0.701 = 1$.

Table 5.2 gives a summary of the weights of items in different classes. From the above argument, we conclude the following lemma.

Lemma 5.2. *The total weight of squares in each bin opened by Square-Rotate, except possibly a constant number of them, is at least 1.*

Next, we provide an upper bound for the total weight of items in a bin of the optimal offline algorithm (Opt).

Lemma 5.3. *The total weight of items in a bin of Opt is less than* 2.306.

Proof. We first define the *density* of an item of size x as the ratio between its weight and area, i.e., $w(x)/x^2$. Given the lower bound for the size of each square belonging to class i $(1 \leq i \leq 12)$, we can calculate a lower bound for the density of each item in the class. For tiny items, the density is simply $1.425x^2/x^2 = 1.425$. Densities for all classes have been reported in Table 5.2.

Defining densities comes handy in the following case analysis to prove that the total weight of items in any bin B of an optimal packing is at most 2.306.

Case 1: First, assume there is no item of class 1 in B. Since the density of items of other classes are less than 1.838, even if B is fully packed with items of the largest density, the total weight of items cannot be more than 1.838 which is less than 2.306.

Case 2: In the second case, we assume there is one item x of class 1 (note that no two items of class 1 fit in the bin). Without loss of generality, we assume the size of x is $1/2 + \epsilon$, where ϵ is a small real value greater than zero. Clearly, a larger size for x does not increase the total weight of other items in B because it would leave less space to occupy more items in the bin (while the weight of x stays 1). Next, we consider all possible cases in which we have some items of class 2 and 3 together with x in B. As presented in Table 5.3, there will be 14 sub-cases to analyze. To see how we reach these 14 sub-cases, first note that it is not possible to accommodate 4 or more items of class 2 in addition to x in B (i.e., a total number of 5 or more items from these classes 1 and 2). This is because no five items with size larger than 0.3694 can fit in B [60]. A similar argument shows that we cannot have 6 or more items from classes 1, 2, and 3 together in a bin, otherwise we could accommodate

	C1	C2	C3	Sum of Weights (W)	Sum of Areas (A)	Remaining Area ($A_r = 1 - A$)	Weight of Items in the Remaining Area ($W_r = A_r \times 1.543$)	Total Weight of Items in the Bin ($W_{max} = W + W_r$)
Number of items of each class $c \leq 3$ in B	1	0	0	1.00	> 0.250	< 0.750	< 1.157	< 2.157
	1	0	1	1.20	> 0.361	< 0.639	< 0.986	< 2.186
	1	0	2	1.40	> 0.472	< 0.528	< 0.815	< 2.215
	1	0	3	1.60	> 0.583	< 0.417	< 0.644	< 2.244
	1	0	4	1.80	> 0.694	< 0.306	< 0.472	< 2.272
	1	1	0	1.25	> 0.386	< 0.614	< 0.948	< 2.198
	1	1	1	1.45	> 0.497	< 0.503	< 0.776	< 2.226
	1	1	2	1.65	> 0.608	< 0.392	< 0.605	< 2.255
	1	1	3	1.85	> 0.719	< 0.281	< 0.434	< 2.284
	1	2	0	1.50	> 0.522	< 0.478	< 0.738	< 2.238
	1	2	1	1.70	> 0.633	< 0.367	< 0.566	< 2.266
	1	2	2	1.90	> 0.744	< 0.256	< 0.395	< 2.295
	1	3	0	1.75	> 0.658	< 0.342	< 0.528	< 2.278
	1	3	1	1.95	> 0.769	< 0.231	< 0.356	< 2.306

Table 5.3: All fourteen possible cases in which we have a combination of items of class 2 (C2) and 3 (C3) together with one item x of class 1 (C1) in a single bin B. Here, "sum of weights (W)" and "sum of areas (A)" indicate, respectively, the total weight and area of items of the first three classes in B. "Remaining area" is the area left in the bin that is used for packing items of class 4 or higher. "Weight of items in the remaining area" is an upper bound for the total weight of items of class 4 or higher in B (these items have density no more than 1.543). Finally, "the total weight of items in the bin" indicate the sum of weights of all items (from all classes) in B.

6 identical squares of size strictly larger than 0.3333 which is a contradiction to the fact that no six items of size larger than 0.3333 can fit in the same bin [60]. We can conclude that the 14 sub-cases summarized in Table 5.3 cover all possibilities for items of the first three classes in Case 2.

According to Table 5.2, the density of items belonging to class i ($4 \leq i \leq 12$) as well as tiny items is at most 1.543 (which is the density of class-4 items). Using a similar argument made for Case 1, we suppose that, after placing a certain number of items of class 2 and 3 beside x in B, in each sub-case, we are able to completely fill the remaining empty space of B with the items of the maximum density 1.543. This makes us able to calculate an upper bound for the maximum total weight of items in B for each of the sub-cases. The resulting bounds for each sub-case can be found in the last column of Table 5.3, where the maximum upper bound among all sub-cases is 2.306, which happens when we have one item of class 1 in B together with 3 items of class 2 and one item of class 3.

	C2	C3	Total Weight of Items in the Bin	C2	C3	Total Weight of Items in the Bin
Number of items of each class $c \in \{2,3\}$ in B.	0	0	< 1.543	1	4	< 1.698
	0	1	< 1.572	2	0	< 1.623
	0	2	< 1.601	2	1	< 1.652
	0	3	< 1.629	2	2	< 1.681
	0	4	< 1.658	3	0	< 1.664
	0	5	< 1.687	3	1	< 1.692
	1	0	< 1.583	3	2	< 1.721
	1	1	< 1.612	4	0	< 1.704
	1	2	< 1.641	4	1	< 1.732
	1	3	< 1.669			

Table 5.4: Maximum total weight of items of size $\in (0, 1/2)$ in a bin B of an optimal packing in all nineteen possible cases in which there is no item of class 1 but a combination of items of class 2 (C2) and 3 (C3) in B.

As a result, in both Case 1 and Case 2, the total weight of items in B cannot be more than 2.306. ■

Provided with the above two lemmas, we can derive the main result of this section.

Theorem 5.1. *There is an online algorithm for the square packing problem with rotation problem which achieves a competitive ratio of at most* 2.306*.*

Proof. For an input σ, let $SR(\sigma)$ and $OPT(\sigma)$ denotes the cost of Square-Rotate and Opt, respectively. Let $w(\sigma)$ denote the total weight of items of σ. Lemmas 5.2 implies that $SR(\sigma) \leq w(\sigma) + c$, where c is a constant independent of the length of σ. Meanwhile, Lemma 5.3 implies that $Opt(\sigma) \geq w(\sigma)/2.306$. From these inequalities, we conclude $SR(\sigma) \leq 2.306\ OPT(\sigma) + c$, which proves an upper bound 2.306 for the competitive ratio of Square-Rotate. ■

In the analysis of the bin packing problem and its variants, it is common to study algorithms in restricted settings in which there is an upper bound for the maximum size of items. In what follows, we study Square-Rotate when all items are of size at most 1/2. Similar approaches can be used to study the competitive ratio when all items are smaller than x for any $x \leq 1/2$.

Theorem 5.2. *When all items have a size of less than 1/2, Square-Rotate achieves a competitive ratio of* 1.732*.*

Proof. We employ the same approach and weighting function as the one we used to analyze the upper bound of Square-Rotate for the general setting, except that

we exclude items of class 1, that is, items of size more than 1/2. By lemma 5.2, the total weight of items in each bin of Square-Rotate, except for a possibly constant number of them, is at least 1. Therefore, to prove the theorem, it suffices to show the total weight of items in any bin B of an optimal packing is at most 1.732. To study the maximum weight of items in B, we consider all possible combinations of items from classes 2 and 3 in B. For that, we consider the following two limitations: (i) we cannot have more than 4 items of class 2 in B; otherwise, we could accommodate 5 squares of size more than 0.3694, which is not possible [60]. (ii) we cannot have more than 6 items of class 2 or 3 in B. Otherwise, one could place 6 squares of size more than 0.3333, which is is known to be impossible [60]. Altogether, we will have 19 cases to consider as presented in Table 5.4. We have used the same method as in Lemma 5.3 to calculate the upper bound for the total weight of items in each case. Table 5.4 summarizes the final results for all cases. The maximum weight, 1.732, happens when we have four items of class 2 together with one item of class 3 packed in B. ■

5.2 Online tan packing

In this section we study a problem similar to the online square packing problem, called the online tan packing problem, defined as follows:

Definition 5.1. *In the **online tan packing with rotation** problem, the input is an online sequence $\sigma = \langle x_1, x_2, \ldots, x_n \rangle$, where $x_i \in (0, 1]$ denote the leg sizes of right isosceles triangles (half-square triangles), referred to as "tans", that need to be packed into unit bins. The goal is to pack the input into a minimum number of bins. The decisions of the algorithm at any time t are irrevocable and are made without knowing the values of $x_{t'}$ for $t' > t$.*

5.2.1 Half-square-Rotate algorithm

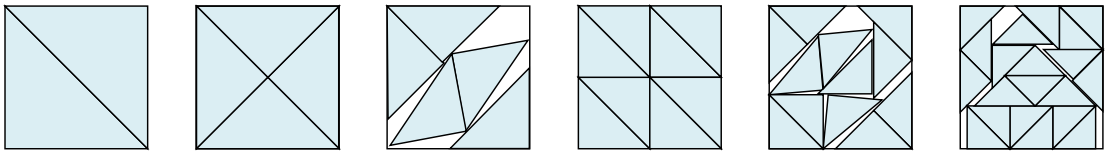
We will introduce an online algorithm, called Half-square-Rotate, to pack tans into unit-square bins. In a similar way to Square-Rotate, we classify tans by their

Class	Side length x	T_i	Occupied Area	Weight	Density
1	$(0.7072, 1.0000]$	2	$> 2(0.250)=0.500$	$1/2$	< 2.000
2	$(0.5593, 0.7072]$	4	$> 4(0.156)=0.626$	$1/4$	< 1.599
3	$(0.5000, 0.5593]$	5	$> 5(0.125)=0.625$	$1/5$	< 1.600
4	$(0.3828, 0.5000]$	8	$> 8(0.073)=0.586$	$1/8$	< 1.706
5	$(0.3056, 0.3828]$	12	$> 12(0.047)=0.560$	$1/12$	< 1.785
6	$(0.2843, 0.3056]$	20	$> 20(0.040)=0.808$	$1/20$	< 1.237
Tiny	$(0, 0.2843]$		> 0.557	$1.795(x^2/2)$	1.795

Table 5.5: A summary of item classification and details on item weights and densities, as used in the definition and analysis of Half-square-Rotate.

leg sizes, and the algorithm packs tans of each class separately from other classes. There are 7 classes, as presented in Table 5.5. We refer to items that belong to classes $i \in [1, 6]$ as regular items and those in class 7 as tiny items. Tiny items have sizes in the range $(0, 1.1752]$. For each class $i \in [1, 6]$, the range of items in class i is specified as $(y_i, y_{i-1}]$ (for convenience, we define $y_0 = 1$). The values of y_i are defined so that a certain number T_i of tans of class i can fit in the same bin.

The specific range of item sizes for each class $i \in [1, 6]$ and values of T_i is derived from the best-known or optimal results on the congruent tan packing problem [60], which asks for the minimum size $s(j)$ of a square that can contain j tans of unit leg size. A scaling argument, where the container size is fixed to be 1, gives $t(j)$ values when the goal is to pack j identical tans of maximum leg size $t(j)$ into a unit square. Table 5.6 provides the scaled best-known/optimal $t(j)$ values for $1 \leq j \leq 20$. These scaled numbers gives the specific ranges that we used for classifying items as follows: Tans of class 1 have sizes in the range $(0.7072, 1]$, and we have $y_1 = 0.7072$. Note that exactly $T_1 = 1$ item of class 1 can fit in the same bin (for tans of size ≤ 0.7072, it is possible to pack at least two tans in the bin). For $i \in [2, 6]$, T_i is the number of



(a) Class 1: $x \in (0.7072, 1.0000]$ $t = 1.000$
(b) Class 2: $x \in (0.5593, 0.7072]$ $t = 0.7072$
(c) Class 3: $x \in (0.5000, 0.5593]$ $t = 0.5593$
(d) Class 4: $x \in (0.3828, 0.5000]$ $t = 0.5000$
(e) Class 5: $x \in (0.3056, 0.3828]$ $t = 0.3828$
(f) Class 6: $x \in (0.2843, 0.3056]$ $t = 0.3056$

Figure 5.2: Placement of regular tan items of class $i \in [1, 6]$ in their respective bin. It is possible to pack i tans of class i into a single square bin [60].

j	$t(j)$	Optimal?	j	$t(j)$	Optimal?
1	$= 1.0$	optimal	11	≈ 0.4143	best-known
2	$= 1.0$	optimal	12	≈ 0.3828	best-known
3	≈ 0.7072	optimal	13	≈ 0.3720	best-known
4	≈ 0.7072	optimal	14	≈ 0.3614	best-known
5	≈ 0.5593	best-known	15	≈ 0.3536	best-known
6	≈ 0.5163	best-known	16	≈ 0.3536	optimal
7	≈ 0.5003	best-known	17	≈ 0.3367	best-known
8	$= 0.5$	optimal	18	≈ 0.3333	optimal
9	≈ 0.4531	best-known	19	≈ 0.3124	best-known
10	≈ 0.4179	best-known	20	≈ 0.3056	best-known

Table 5.6: Optimal or best-known $t(j)$ values for $1 \leq j \leq 20$ when the goal is to pack j identical tan of the largest leg size $t(j)$ into a unit square. Values of $t(j)$ are the scaled values of the known results on congruent tan packing [60].

items of size y_{i-1} that fit in the same bin. For example, for $i = 2$, we have $T_2 = 4$ because $y_1 = 0.7072$, and 4 items of size 0.7072 fit in the same bin. Moreover, y_i is defined as the largest value so that $T_i + 1$ items of size y_i cannot fit in the same bin. For example, we have $y_2 = 0.5593$ because, according to Table 5.6, $T_2 + 1 = 5$ tans of size 0.5593 cannot fit in the same bin. The respective range of items for each class as well as the values of T_i are presented in Table 5.5. Figure 5.2 shows how T_i items of the largest size in class i can fit into a square bin.

5.2.1.1 Packing regular items

For each class i $(1 \leq i \leq 6)$, the algorithm has at most one active bin of type i. When a bin of type i is opened, it is declared as the active bin of the class, and T_i tan "spots", each of which having a size equal to the largest tan of class i, are reserved in the bin. Upon the arrival of an item of class i, it is placed in one of the T_i spots of the active bin. If all these spots are occupied by previous items, a new bin of type i is opened. This ensures that all bins of type i, except potentially the current active bin, include S_i items.

5.2.1.2 Packing tiny items

In order to pack tiny items, Half-square-Rotate uses the same approach as Square-Rotate for packing tiny tans. Namely, the algorithm maintains a partitioning of any tiny bin into sub-bins formed by tans of various sizes. The only difference, compared to the algorithm of Epstein and van Stee [61], is that the square bin is originally divided into 2 tans (of side length 1) instead of four sub-bins, and subsequently, instead of partitioning larger square sub-bins into four sub-squares, we partition large tan sub-bins into *two* smaller tan sub-bins. One important observation is that it is possible to partition a tan into two sub-tans, which allows using the same approach as in [61]. Using an identical proof to the one in [61], we can show this adapted algorithm almost fully packs each bin:

Lemma 5.4. *[61] Consider the tan packing problem in which all items are of size at most $1/M$ for some integer $M \geq 2$. There is an online algorithm that creates a packing in which all bins, except possibly one, have an occupied area of size at least $(M^2-1)/(M+1)^2$.*

5.2.1.3 Algorithm's analysis

In this section, we use a weighting argument to analyze the competitive ratio of our algorithm and prove a competitive ratio of at most 1.897 for it. Corresponding to each tan of size x, we define a weight $w(x)$ as follows. Recall that all bins opened for tans of class i $(1 \leq i \leq 6)$, except possibly the last active bin of each class, include T_i tans. We define the weight of items of class i to be $1/T_i$. For a tiny tan of leg size x, we define its weight as $1.795(x^2/2)$ where $x^2/2$ is the area of the tan. Table 5.5 gives a summary of the weights of the items in different classes.

Lemma 5.5. *The total weight of tans in each bin opened by Half-square-Rotate, except possibly a constant number of them, is at least 1.*

Proof. In every bin of class i $(1 \leq i \leq 6)$, except possibly the last bin, there are T_i items, each having a weight of $1/T_i$. As a result, the total weight of the items in each bin of such classes, except possibly 6 of them, is exactly $T_i \times 1/T_i = 1$. For bins of tiny items, we know, by Lemma 5.4 and considering $1/M = 0.2843$, that the

occupied area of all bins (except a constant number of them) will be at least 0.557. The total weight of items in such a bin is therefore at least $W = 1.795 \times 0.557 = 1$.

■

Next, we provide an upper bound for the total weight of items in a bin of the optimal offline algorithm (Opt).

Lemma 5.6. *The total weight of items in a bin of Opt is less than* 1.897.

Proof. We first define the *density* of a tan item of leg size x as the ratio between its weight and its area, i.e., $w(x)/(x^2/2)$. Given the lower bound for the leg size of items in each class i $(1 \leq i \leq 6)$, and, hence, their area, we can calculate an upper bound for the density of each item in the class. For tiny items, the density is simply equal to $1.795(x^2/2)/(x^2/2) = 1.795$. The densities of items from different classes are reported in Table 5.5. In what follows, we use a case analysis approach to prove that the total weight of items in a bin B of an optimal packing is at most 1.897. There are three cases to consider: either 0, 1, or 2 tans of class 1 exist in the bin. Note that it is not possible to accommodate 3 or more items of class 1 in a bin because their total area would exceed 1.

Case 1: No class-1 item in B: Since the density of items of class 2 or larger is at most 1.795, even if all area of B is filled with items of the largest density, the total weight of items cannot exceed 1×1.795 which is less than 1.897.

Case 2: Exactly one class-1 item in B: If there is exactly one item of class 1 (with weight 1/2 and the area of at least $(0.7072 \times 0.7072)/2 = 0.25$) in B, the remaining area in the bin will be at most 0.75. Items of classes other than class 1 have a density of at most 1.795. Even if we fill the remaining area with such items of the highest density, the total weight of items in B will not exceed 1.346. As a result, the total weight of items in B cannot be more than $1/2 + 1.346 = 1.846$ which is again less than 1.897.

Case 3: Exactly two class-1 items in B: If there exists exactly two items of class 1 with weight 1/2 (and total weight of 1) and total area of at least $2 \times (0.7072 \times 0.7072)/2 = 0.50$, the remaining area in B will be at most 0.50. If this remaining area is filled by items of the highest density, that are items of the last class with

a density of at most 1.795, the total weight of items in B will not be more than $1 + 0.50 \times 1.795 = 1.897$.

In conclusion, the total weight of items in a bin B of an optimal packing cannot be more than 1.897 in all cases. ■

Now, we can give the main result of this section.

Theorem 5.3. *There is an online algorithm for the tan packing with rotation that achieves a competitive ratio of at most* 1.897.

Proof. For an input σ, let $HR(\sigma)$ and $OPT(\sigma)$, respectively, denote the cost of Half-square-Rotate and Opt. Let $w(\sigma)$ denote the total weight of items of σ. Lemmas 5.5 implies that $HR(\sigma) \leq w(\sigma) + c$, where c is a constant independent of the length of σ. Meanwhile, Lemma 5.6 implies that $Opt(\sigma) \geq w(\sigma)/1.897$. From these inequalities, we conclude $HR(\sigma) \leq 1.897\ OPT(\sigma) + c$, which proves an upper bound 2.306 for the competitive ratio of Square-Rotate. ■

Chapter 6

Online Fault-tolerant Bin Packing*

In this chapter, we study the primary-standby scheme for the fault-tolerant bin packing problem (see Sections 2.3 and 3.3 for more details). In this problem, the goal is to pack an online sequence of items (tenants) into a minimum number of bins (servers) such that each item of size x has a primary replica of size x and f standby replicas, each of size x/η, for some parameter $\eta > 1$. Over time, some servers might fail, and some of the previously failed servers might recover. An algorithm has no knowledge about how servers fail or recover, but it is guaranteed that the number of failed servers at each given time is at most f. To ensure the service is fault-tolerant, the primary replica of each tenant should be available at any given time. Therefore, when a server that hosts the primary replica of an item x fails, a standby replica of x should be selected to become its new primary replica. The subsequent increase in the load of such replica (from x/η to x) should not cause an overflow in the bin (see Definition 3.3 for a formal definition).

As discussed in Section 3.3, we make two assumptions that make our model more practical compared to the existing primary-standby models for the bin packing problem. First, we assume that bins might fail during the execution of the algorithm (before all items are packed). Second, we assume that the failed bins might later recover again after a failure. Our main contribution is an algorithm, named Harmonic-Stretch, which maintains fault-tolerant packings under these two assumptions. We prove that Harmonic-Stretch has an asymptotic competitive

*A summary of the results in this chapter will be published in the Proceedings of the 6th International Workshop on Algorithmic Aspects of Cloud Computing (ALGOCLOUD'21) [62]

ratio of at most 1.75. Given that our model is a generalization of the existing model for the problem (as discussed in Section 3.3), our result is also an improvement over the best existing asymptotic competitive ratio 2 of an algorithm by Li and Tang [45] (see Section 2.3), which works under a model that assumes bins fail only after all items are packed.

6.1 Harmonic-Stretch algorithm

In this section, we introduce our algorithm, Harmonic-Stretch, which maintains fault-tolerant packings for an online sequence of items. As the prefix "Harmonic" suggests, Harmonic-Stretch classifies items by their sizes. The classification and treatment of items in each class is, however, different from the existing Harmonic-based algorithms. In particular, unlike the algorithms in [41, 45] (see Section 2.3), which classify items based on the size of their standby replicas, Harmonic-Stretch classifies items based on the size of *both* their primary and standby replicas.

As mentioned before, in our model (see Definition 3.3), we assume that bins can fail and recover in an online manner so that at most f bins are failed at any given time. As such, an algorithm in this model requires a *packing strategy*, which allocates items to bins, and an *adjustment strategy*, which makes necessary adjustments (i.e., promoting a standby replica to a primary replica and vice versa) when bins fail or recover. The packing and adjustment strategies in Harmonic-Stretch are designed in a flexible way that maintains valid packings, that is, primary replicas are available for all items, and no bin is overloaded throughout the packing and adjustment processes.

First, we provide an overview of the main components of the algorithm.

Item classification. Items are partitioned into *classes*, based on the size of their primary and standby replicas. There are 7 possible classes for primary replicas and $\lfloor 6\eta \rfloor + 1$ classes for standby replicas. An item that has a primary replica of class i ($1 \leq i \leq 7$) and a standby replica of class j ($1 \leq j \leq \lfloor 6\eta \rfloor + 1$) is called an (i, j)-item. When $i = 7$ and $j = \lfloor 6\eta \rfloor + 1$, the (i, j)-items are called *small items*, while other items are called *regular items* (see Section 6.1.1 for details).

Items that have the same primary and standby classes are packed separately from other items, that is, a replica of an (i, j)-item is never placed with a replica of an (i', j')-item together in the same bin if $i \neq i'$ or $j \neq j'$.

Maintaining bin groups. To place (i, j)-items, the algorithm maintains *groups* of bins. Each group is formed by a constant number of (initially empty) bins. At any given time, there is one "active" group for (i, j)-items, into which the incoming (i, j)-items are packed. When no more items can fit into the active group, the group becomes "complete", and another group becomes the active group. If a bin of the active group fails, the algorithm declares that group as "unavailable", leaves that group "incomplete" and selects a new active group. When all failed bins of an incomplete group recover, that group becomes "available" again. When a new active group is required (i.e., when the currently active group becomes complete or one of its bins fails), an (incomplete) available group is selected as the new active group. If no available group exists, the algorithm opens a fresh group with empty bins as the active group. Given that at most f bins can fail at the same time, the algorithm maintains up to $f + 1$ incomplete groups for (i, j)-items, out of which one group is the active group, and f groups are either unavailable or include bins that have recovered from a failure (see Section 6.1.2).

Packing strategy. When placing (i, j)-items inside their active group, primary and standby replicas are packed in separate bins, which are, respectively, called *primary* and *standby bins*. Items in primary bins are packed as tightly as possible, while items in the standby bins are packed so that there is enough space for the promotion of exactly one replica. The packing strategy ensures that a primary bin shares replicas of at most one item with any standby bin. For small items, a consecutive number of them are merged to form "super-replicas"; each super-replica is treated in the same way that regular items are packed (see Section 6.1.3).

Adjustment strategy. When a bin B fails, for each primary replica x in B, a standby replica of x in a non-failed bin should be promoted to become the new primary replica. This is done through maintaining an injective mapping h. The domain of h is the set of primary replicas like x residing in the primary bins that

are failed, and the range of h is a set of non-failed standby bins in the same group such that $h(x)$ contains a standby replica of x. The injective nature of the mapping, and the fact that there is enough space for expansion of one standby replica in $h(x)$, implies that one can promote the standby replica in $h(x)$ to replace x as the primary replica, without causing an overflow in $h(x)$. The assignment of standby replicas as primary replicas is temporary, that is, upon the recovery of B, its primary replicas like x will retain their primary status and are removed from the domain of h, while the promoted replica in $h(x)$ is demoted to become a standby replica again (see Section 6.1.4).

In what follows, we explain the above components in more detail.

6.1.1 Item classification

There are seven classes for primary replicas and $\lfloor 6\eta \rfloor + 1$ classes for standby replicas (see Table 6.1 for details). An item has *primary class* $i \in \{1,2,3,4,5\}$ if its primary replica is of size in the range $(\frac{1}{i+1}, \frac{1}{i}]$, primary class 6 if its primary replica is of size in the range $(\frac{1}{7-1/\eta}, \frac{1}{6}]$, and primary class 7 if its primary replica is of size at most $\frac{1}{7-1/\eta}$. We refer to items of primary class $i \leq 6$ as *regular* items, and items of primary class 7 as *small* items. An item has *standby class* $j \in \{1,2,\ldots,\lfloor 6\eta \rfloor - 1\}$ if its standby replica has size in the range $(\frac{1}{j+\eta}, \frac{1}{j+\eta-1}]$, standby class $j = \lfloor 6\eta \rfloor$ if its standby class has size in the range $(\frac{1}{7\eta-1}, \frac{1}{\lfloor 6\eta \rfloor+\eta-1}]$, and standby class $\lfloor 6\eta \rfloor + 1$ if its standby replica is of size at most $\frac{1}{7\eta-1}$.

In what follows, we refer to an item of primary class i and standby class j as an (i,j)-item. Primary replicas of small items (with $i = 7$) are in the range $(0, \frac{1}{7-1/\eta}]$, and their standby replicas are in the range $(0, \frac{1}{7\eta-1}]$. Therefore, an (i,j)-item is a small item if $i = 7$ and $j = \lfloor 6\eta \rfloor + 1$, and a regular item otherwise.

6.1.2 Maintaining bin groups

For each pair of i,j, the algorithm maintains one group of bins, which are all non-failed, as the *active* group. As (i,j)-items are revealed, they are packed into bins of the active group, as will be explained in Section 6.1.3. In the beginning, a group of all-empty bins is opened and declared as the active group for (i,j)-items. A new

primary replicas			
class	**size**	**weight**	**density**
$i = 1$	$(\frac{1}{2}, 1]$	1	< 2
$i = 2$	$(\frac{1}{3}, \frac{1}{2}]$	$\frac{1}{2}$	$< \frac{3}{2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$i \in [1, 5]$	$(\frac{1}{i+1}, \frac{1}{i}]$	$\frac{1}{i}$	$< \frac{i+1}{i}$
$i = 6$	$(\frac{1}{7-1/\eta}, \frac{1}{6}]$	$\frac{1}{6}$	$< \frac{7}{6}$
$i = 7$	$p \in (0, \frac{1}{7-1/\eta}]$	$\frac{3}{2}p$	$\frac{3}{2}$

standby replicas			
class	**size**	**weight**	**density**
$j = 1$	$(\frac{1}{\eta+1}, \frac{1}{\eta}]$	1	$< \eta + 1$
$j = 2$	$(\frac{1}{\eta+2}, \frac{1}{\eta+1}]$	$\frac{1}{2}$	$< \frac{\eta+2}{2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$j \in [1, \lfloor 6\eta \rfloor - 1]$	$(\frac{1}{\eta+j}, \frac{1}{\eta+j-1}]$	$\frac{1}{j}$	$< \frac{\eta+j}{j}$
$j = \lfloor 6\eta \rfloor$	$(\frac{1}{7\eta-1}, \frac{1}{\eta+\lfloor 6\eta \rfloor-1}]$	$\frac{1}{\lfloor 6\eta \rfloor}$	$< \frac{\eta+\lfloor 6\eta \rfloor}{\lfloor 6\eta \rfloor}$
$j = \lfloor 6\eta \rfloor + 1$	$s \in (0, \frac{1}{7\eta-1}]$	$\frac{3}{2}s$	$\frac{3}{2}$

Table 6.1: A summary of the replica classes used in the definition and analysis of the Harmonic-Stretch algorithm. The weight and density of classes is used in the analysis of the algorithm.

active group is needed when either i) one of the bins in the active group fails or ii) enough items are placed inside the active group, and the group becomes *complete*; before that, the group is *incomplete*. A group is said to be *available* if none of its bins are failed. An active group is always available. When a new active group is required, the algorithm checks whether an incomplete and available group exists. Such a group, if it exists, is a former active group that, at some point lost its active status due to a bin failure. Since the group is now available, its failed bins should be recovered. If such a group exists, it is selected as the new active group (if multiple such groups exist, one is chosen arbitrarily). On the other hand, if no incomplete, available group exists, the algorithm opens a fresh group of all-empty bins and declares it as the active group.

Lemma 6.1. *There are at most $f + 1$ incomplete groups at any given time during the execution of the algorithm.*

Proof. Consider otherwise, that is, at some point, there are at least $f + 2$ incomplete groups. Let t denote the time at which the $(f + 2)$th group G is initiated. There are at most f groups that contain at least one failed bin at any given time, in particular, at time t. So, out of the $f + 1$ incomplete groups at time t (before G is initiated), at least one group G' has been incomplete and available. Therefore, G' had to be selected as the new active group instead of G, a contradiction. ■

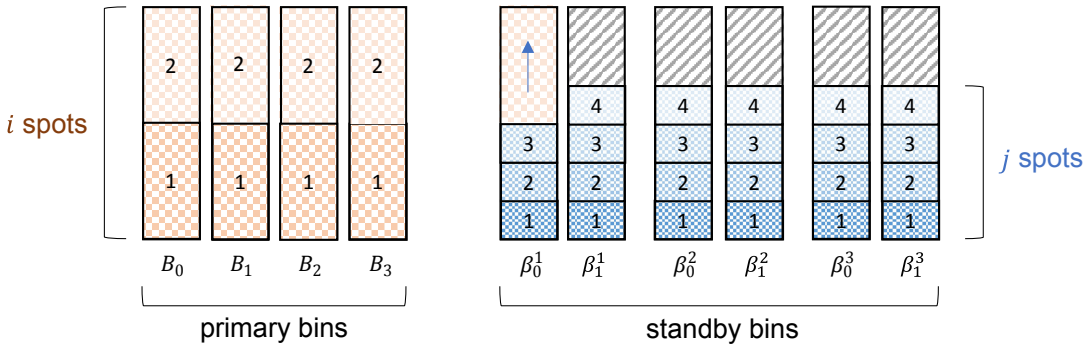


Figure 6.1: An illustration of the group structure for (i, j)-items where $i = 2$, $j = 4$, and $f = 3$: There are j primary bins each partitioned into i *spots* of capacity $1/i$. There are fi standby bins formed by f sets of bins, each containing i standby bins. Each standby bin is partitioned into j spots of size $\frac{1}{j+\eta-1}$, which leaves a reserved space for the expansion of one standby replica of class j into the primary replica of class i.

6.1.3 Packing strategy

We explain how the algorithm packs (i, j)-items inside their active group. The placement is slightly different for regular and small items:

Regular items. We describe how (i, j)-items are packed, where $i \leq 6$ and $j \leq \lfloor 6\eta \rfloor$. Each bin group for (i, j)-items, in particular the active group, is formed by $j + fi$ bins and has enough space for ij items. The group becomes complete when ij items are placed in it. There are j primary bins $B_0, B_1, \ldots, B_{j-1}$ that are each partitioned into i *spots* of capacity $1/i$. There are fi standby bins formed by f sets of bins, each containing i standby bins. We use $\beta_0^k, \beta_1^k, \ldots \beta_{i-1}^k$ to denote the standby bins in the kth set ($k \leq f$). Each standby bin is partitioned into j spots of size $\frac{1}{j+\eta-1}$. This leaves a *reserved space* of size $\frac{\eta-1}{j+\eta-1}$ in the bin. The spots in the standby bins are labeled from 0 to $j-1$. Figure 6.1 illustrates the structure of a group for $(2, 4)$-items where $f = 3$.

Let a_t be the tth item that is to be packed into the group ($0 \leq t \leq ij - 1$). Let $w = (t \mod j)$ and $z = \lfloor t/j \rfloor$. Note that w and z are in the ranges $[0, j-1]$ and $[0, i-1]$, respectively. The algorithm places the primary replica of a_t in the spot z of the primary bin B_w in the active group. Standby replicas of a_t are placed in the spot w of bins $\beta_z^1, \beta_z^2, \ldots, \beta_z^f$ of the active group.

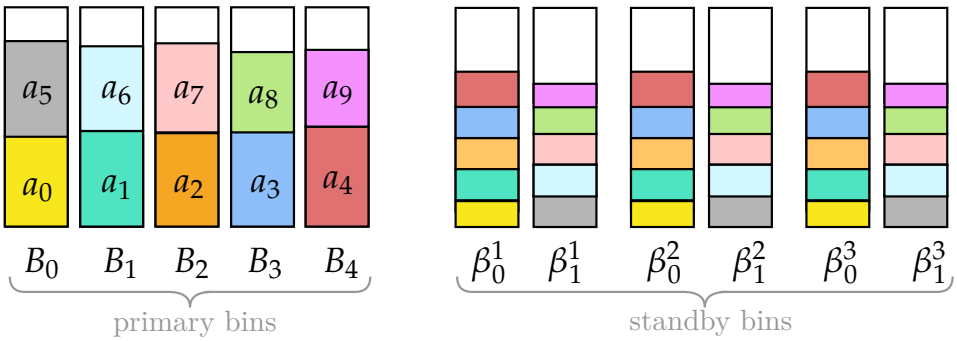


Figure 6.2: An illustration of the Harmonic-Stretch packing for regular (i,j)-items in a complete group, where $i = 2$, $j = 5$, and $f = 3$.

Example 6.1. *Figure 6.2 illustrates packing items of class (i,j), where $i = 2$, $j = 5$, and $f = 3$ in a complete group. There are $j = 5$ primary bins $B_0, \ldots, B_4$, each partitioned into $i = 2$ spots. There are $f = 3$ groups of standby bins, each containing $i = 2$ bins that are partitioned into $j = 5$ spots. For an item like a_4 (the red item), we have $w = 4$ and $z = 0$. The primary replica of a_4 is thus placed in the 0th spot of the 4th bin, while standby replicas of a_4 are placed in the 4th spot of the bin β_0^k for $k \in [1,3]$.*

Example 6.2. *Figure 6.3 shows an incomplete group of bins opened for (i,j)-items where $i = 2$, $j = 5$, and $f = 3$. The group includes $j + fi = 11$ bins. Each of the primary bins and standby bins has $i = 2$ and $j = 6$ spots, respectively, out of which the black spots have not been filled yet. The algorithm has placed items $a_0, \ldots a_6$ in their respective bins when the group was active (and available). At some time t, bins B_0 and B_4 failed. At this point, the group becomes unavailable, and the adjustment strategy processes a_0, a_5, and a_4 to assign new primary replicas for them. After time t, the group will not be active anymore (because it is not available), and the algorithm does not place upcoming items in this group until it is selected as the active group later again; this requires the group to become available again, that is, B_0 and B_4 must be recovered.*

Small items. In order to pack replicas of small items, Harmonic-Stretch merges sets of consecutive small items into *super-replicas* (SRs). Given that the size of small primary and standby replicas are, respectively, at most $\frac{1}{7-1/\eta}$ and $\frac{1}{7\eta-1}$, it is possible to group consecutive primary and standby replicas into SRs with sizes in the range

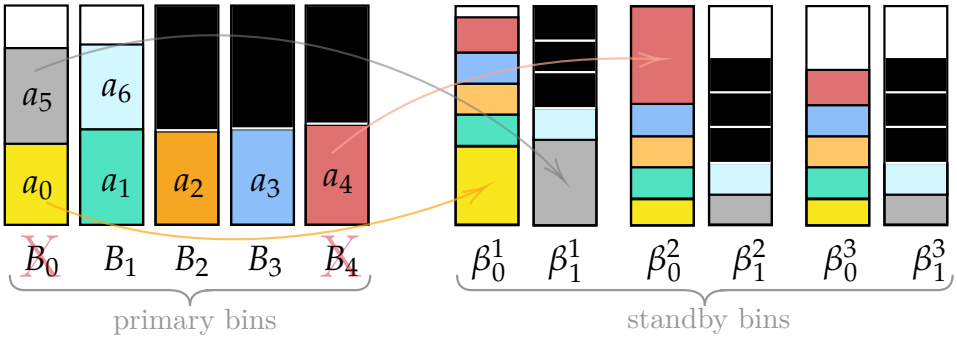


Figure 6.3: An unavailable and incomplete group of bins for (i, j)-items, where $i = 2, j = 5$, and $f = 3$.

$(\frac{1}{7-1/\eta}, \frac{2}{7-1/\eta})$ and $(\frac{1}{7\eta-1}, \frac{2}{7\eta-1})$, respectively. The algorithm maintains an (initially empty) *open* primary SR of capacity $\frac{2}{7-1/\eta}$ and places consecutive primary replicas in the open SR until placing the next replica causes the total size of replicas in the SR to exceed its capacity. At this point, the SR is closed and a new SR is opened. Similarly, the algorithm maintains f (initially empty) open standby SRs, each of capacity $\frac{2}{7\eta-1}$, and places the f standby replicas in these bins until a replica does not fit in the open SRs, at which point the f open SRs are closed and a set of f new SRs are opened. Since the primary and standby SRs are opened and closed at the same time, we can think of a set of small replicas that are placed in an SR as a single regular replica. In what follows, we describe how the newly opened SRs are placed into bins.

Each group G of bins opened for small items contains f standby bins that mirror each other* and one primary replica which is "committed" to G. There are also "free" primary bins that are not committed to any group. Upon the arrival of a small item, its standby replicas are placed into the open SRs located on (mirroring bins) of the active group, and its primary replica is placed into the open SR located on the committed bin. As before, if any bin of the active group fails, the group becomes unavailable, and the algorithm declares another incomplete, available group as the active group (or creates a new one if no such group exists). There is a reserved space of size $\frac{2(\eta-1)}{7\eta-1} + \frac{2}{7\eta-1} = \frac{2(\eta-1)}{7\eta-1}$ inside each of the f mirroring bins of a group, which is used for the promotion of the standby SRs when required. When it is needed to open

*Two bins "mirror" each other iff they contain the same set of replicas.

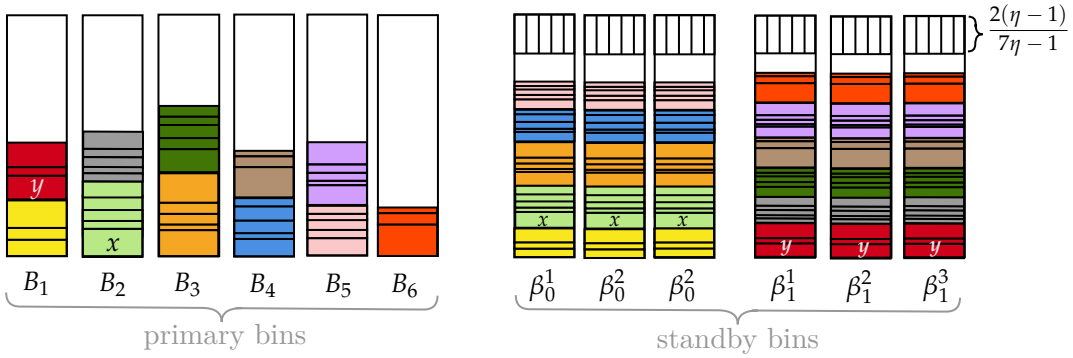


Figure 6.4: An illustration of the Harmonic-Stretch packing for placing small items, where $f = 3$.

a new SR (when the new replicas do not fit in the open SR), Harmonic-Stretch first places f standby SRs and then a single primary SR as follows. For the f standby SRs, if the available space in the mirroring standby bins of the active group is at least $\frac{2\eta}{7\eta-1}$, then the new SRs will be placed into the existing open standby bins. Otherwise, the active group gets complete, and either another incomplete, available group is selected as the active group, or (if there is no such an available group) a new group is opened. For placing a new primary SR, the algorithm first frees the bin committed to the active group, and then selects any free primary bin B' that i) has an empty space of size at least $\frac{2}{7-1/\eta}$ and ii) is not sharing an SR with any of the f standby bins in the active group. If no such bin B' exists, the algorithm opens a new primary bin B'. The bin B' is then declared as the new primary bin committed to the active group, where the new primary SR is placed.

Example 6.3. *Figure 6.4 illustrates how* Harmonic-Stretch *packs small items, where* $\eta = 2$ *and* $f = 3$. *Replicas of items that form the same SR have the same color. SRs of primary replicas have sizes in the range* $(\frac{1}{7-1/\eta}, \frac{2}{7-1/\eta})$, *that is,* $(2/13, 4/13)$. *Standby SRs have sizes in the range* $(\frac{1}{7\eta-1}, \frac{2}{7\eta-1})$, *that is,* $(1/13, 2/13)$. *The algorithm maintains a reserved space of* $\frac{2(\eta-1)}{7\eta-1} = 2/13$ *inside each standby bin. The initial small items are placed into the yellow SRs. Note that the standby SRs are placed in* $f = 3$ *open mirroring bins* (β_0^1, β_0^2, *and* β_0^3) *and the primary SR is placed in bin* B_1, *which is the initially committed bin to the active group. At some point, replicas of an item* x *do not fit in the standby SRs of the active group. This is because*

the total size of primary and standby yellow replicas exceeds $\frac{4}{13}$ *and* $\frac{2}{13}$*, respectively, if replicas of* x *are included in the yellow SRs. As such, a new SR (the light green SR) is opened for* x*. Given that the empty and non-reserved space in the mirroring bins of the current group is enough to fit another standby SR of size at most 2/13, the standby replicas of the new SR are placed in the open mirroring bins. Meanwhile,* B_1 *is freed, and a new bin* B_2 *is selected as the new committed primary bin to the active group, where the new primary SR is placed. Note that* B_1 *is freed because it already shares yellow SR with the standby bins of the active group. Similarly, the standby replicas of the subsequent SRs (of colors orange, blue, and pink) are placed in the mirroring bins of the active group while their primary replicas are placed in separate primary bins (each becoming the new committed bins upon freeing the previous one). At some point, replicas of some item* y *cannot fit in the current (pink) SRs. So, a new SR (of color red) needs to be opened. The current mirroring bins do not have an available space of* $\frac{2}{7\eta-1} = 2/13$*; as such, the active group gets complete, and a new group with* $f = 3$ *standby bins* (β_1^1, β_1^2*, and* β_1^3*) is opened, where the new standby SRs are placed. At this point, the primary replica of the new SR (the red SR) can be placed in any of the primary bins* B_1 *to* B_6*, that is, any of* B_1 *to* B_6 *can be selected as the committed bin to the new active group. This is because none of these primary bins are related to this new set of open standby bins. In the figure,* B_1 *is initially selected as the primary bin committed to the new group.*

6.1.4 Adjustment strategy

We describe an adjustment strategy that ensures a primary replica is available at any given time during the execution of Harmonic-Stretch.

Two bins in the packing of Harmonic-Stretch are said to be *related through item* x if they both contain (primary or standby) replicas of item x. Clearly, any two related bins should belong to the same group of bins.

Lemma 6.2. *In a packing maintained by Harmonic-Stretch, if a primary bin* B_p *is related to a standby bin* β_1 *through an item* x *and to a standby bin* β_2 *through an item* $y \neq x$*, then* β_1 *and* β_2 *are not related through any item.*

Proof. Let $\mathcal{P}$ be the packing maintained by Harmonic-Stretch. First, we show that (i) any pair of standby bins that are related mirror each other, and (ii) any primary bin in $\mathcal{P}$ shares replicas of at most one item with any standby bin. For (i), note that the algorithm places the standby replicas of each regular item in bins that mirror each other. The same holds for the small replicas because the standby SRs are placed into mirroring bins (see Figures 6.2 and 6.4). For small items, (ii) follows directly from the definition of the Harmonic-Stretch. This is because the algorithm places each primary SR into a primary bin that does not share an SR with any (standby) bin of the active group. We use proof by contradiction to prove (ii) for the regular items. Assume a primary bin B includes primary replicas of items x and y of class (i, j) while a standby bin β' also includes replicas of x and y. Given that x and y are regular items, they belong to the same set of the ij items that are placed in the same group (with $j + fi$ bins) in $\mathcal{P}$. Let t_x and t_y, respectively, denote the indices of x and y in the group ($t_x \neq t_y$). Since x and y have their primary replicas in B, we should have $(t_x \mod j) = (t_y \mod j)$. Similarly, since their standby replicas are placed in β', we should have $\lfloor t_x/j \rfloor = \lfloor t_y/j \rfloor$. This contradicts $t_x \neq t_y$.

Provided with (i) and (ii), we are ready to prove the lemma. Suppose the lemma does not hold, that is, a primary bin B_p in $\mathcal{P}$ is related to a standby bin β_1 through an item x and to standby bin β_2 through an item y ($y \neq x$), while β_1 and β_2 are also related. Since β_1 and β_2 are related, by (i), they should mirror each other, that is, β_1 includes replicas of both x and y, and so does β_2. Thus, B_p shares replicas of both x and y with β_1 (and β_2), contradicting (ii). ■

We use Lemma 6.2 to develop the adjustment strategy of Harmonic-Stretch:

Theorem 6.1. *There is an adjustment strategy that ensures the packing of Harmonic-Stretch stays valid, that is, a primary replica of each item is always present in a non-failed bin.*

Proof. We describe an adjustment strategy that maintains an injective mapping h that maps each primary replica x placed originally in a failed primary bin into a non-failed standby bin $h(x)$ that hosts a standby replica of the same item. For each primary replica x in a failed primary bin, the standby replica in $h(x)$ replaces x as the new primary replica. Since the mapping is injective, at most one replica in each

standby bin will be promoted to a primary replica. Given that each standby bin with a replica of size s has an empty space of at least $(\eta - 1)s$, no bin is overloaded. In what follows, we describe an adjustment strategy that maintains the desired injective mapping as bins fail and recover. In this process, the standby bins that are in the range of h are referred to as "marked" bins.

We describe how to maintain the mapping h at time t. Suppose that such mapping is maintained in the previous $t-1$ steps. Let f_p denote the number of primary bins that are failed (and not recovered) before time t (we have $0 \leq f_p \leq f$). Suppose that out of these f_p failed primary bins, r_p bins are recovered at time t ($0 \leq r_p \leq f_p$). A bin B is said to be *critical* iff it fails at time t while containing a primary replica. Let k denote the number of critical bins. All primary bins that fail at time t are critical (non-failed primary bins contain primary replicas). Marked standby bins that fail at time t are also critical (they are in the range of h and, hence, a replica in them has replaced the primary replica of a failed primary bin). For the packing to stay valid, the primary replicas in the critical bins should be mapped to some non-failed bins.

The adjustment algorithm first ensures that the primary replicas in the r_p recovered primary bins are removed from the domain of h, and retain their primary status; this means the standby replicas in the range of h that were previously upgraded to primary replicas become standby again and their bins become unmarked. At this point, the number of failed, unmarked standby bins is at most $f - (f_p - r_p + k)$; this is because, out of at most f failed bins, $f_p - r_p$ of them are primary bins that are failed before t, and k of them are critical and, hence, are either primary bins or marked standby bins.

Consider an arbitrary ordering $(B_1, B_2, \ldots, B_k)$ of the critical bins. We process the critical bins, one by one, in this order. When processing a bin B_q, we process primary replicas in B_q in an arbitrary order ($q \leq k$). Let a be a replica in B_q that is being processed, and let A be the set of f standby bins that include replicas of a. We need to map a to an unmarked bin $\beta \in A$ and then mark β. We show that it is always possible to find such an unmarked bin β. Consider a previously processed bin B' (whose primary replicas are mapped), that is, either B' failed previously at $t' < t$ or B' is $B_{q'}$ for $q' < q$. We claim that during the processing of B', at most one bin

from A has become marked. At the time B' is processed, it has been critical and, hence, either a primary bin or a marked standby bin. If B' was a marked standby bin, then it contained at most one primary replica (since the mapping is injective), and the claim holds. To prove the claim when B' is a primary bin, consider otherwise, that is, assume two standby bins $\beta_x, \beta_y \in A$ have been marked during the process of B' . This means B' is related to β_x and β_y through two different items. On the other hand, β_x and β_y are also related to each other (because they are both in A and, hence, contain a replica of a). This is not possible, however, given the result in Lemma 6.2.

There are $f_p - r_p + q - 1$ failed bins that are processed before B_q. By the above argument, processing any of these bins results in marking at most one standby bin from A. Therefore, at the time of processing a, at most $f_p - r_p + q - 1$ standby bins from A are previously marked. Note that a is replicated on f standby bins. As a result, there are at least $f - f_p + r_p - q + 1$ unmarked standby bins that host standby replicas of a. Among these bins, at most $f - (f_p + r_p - k)$ bins are failed. So, there are at least $f - f_p + r_p - q + 1 - (f - f_b + r_p - k) = k - q + 1$ non-failed and unmarked standby bins in A. Given that $q \leq k$, there is at least one unmarked bin that a can be mapped to. ■

6.2 Competitiveness of Harmonic-Stretch

In this section, we use a weighting argument to provide an upper bound for the competitive ratio of Harmonic-Stretch. We assign a *weight* to each replica in the final packing of the algorithm. The weights are defined in a way that the total weight of replicas placed in each bin of the algorithm, except possibly a constant number of them, is at least 1. Therefore, if $w(\sigma)$ denotes the total weight of all replicas in the input sequence, the number of bins in the packing of Harmonic-Stretch is no more than $w(\sigma) + c$, for some constant c independent of the input length (but possibly a function of parameters f and η). At the same time, we show that any bin in an optimal packing has a weight at most 1.75, which means the number of

bins in an optimal packing is at least $w(\sigma)/1.75$. As such, the competitive ratio of Harmonic-Stretch will be at most 1.75.

Weighting: For regular items, define the weight of a primary replica of class i (≤ 6) as $1/i$, and the weight of standby replicas of class j ($\leq \lfloor 6\eta \rfloor$) as $1/j$. For small items, a primary or standby replica of size x has weight $3x/2$ (see Table 6.1).

Lemma 6.3. *The total weight of replicas in any bin of Harmonic-Stretch, except for at most a constant number of bins, is at least 1.*

Proof. First, we investigate regular bins. Consider the bins in the complete groups (see Figure 6.2). In any such group, a primary bin of class $i \leq 6$ includes i replicas, each of weight $1/i$. Similarly, a standby bin of class $j \leq \lfloor 6\eta \rfloor$ includes j replicas, each of weight $1/j$. Therefore, all regular bins, except for those in the incomplete groups, have weight 1. We show that the total number of bins inside incomplete groups is a constant independent of the input length. By Lemma 6.1, there are at most $f+1$ incomplete groups for (i,j)-items. There are $j+fi \leq 6(\eta+f)$ bins inside each group. So, there are at most $6(f+1)(\eta+f)$ partially-filled bins for (i,j)-items. Given that $i \leq 6$ and $j \leq 6\eta$, there are at most 36η possible pairs of (i,j). In total, the number of partially filled bins for regular items is at most $(36\eta)6(f+1)(\eta+f) = O(1)$. In summary, the total weight of items in any regular bin, except for at most $O(1)$ of them, is at least 1.

Next, we look into small items. Let x be the primary SR that causes opening the last primary small bin. Also, let m be the number of standby SRs packed in the standby bins of the active group at the time x was placed. There are m primary bins that are related to the f standby bins, and thus cannot host x. By Lemma 6.1, there are up to f incomplete groups other than the active group. The primary bins committed to these groups also cannot host x. The remaining primary bins could not host x only because they did not have enough space. So, all primary small bins, except at most $m+f$ of them, are filled to a level of at least $1 - \frac{2}{7-1/\eta} = \frac{5\eta-1}{7\eta-1} \geq 2/3$. Given that any small item of size s has weight $1.5s$, the total weight of replicas in any of these bins is at least $1.5\ (2/3) = 1$. Next, we show that m is a constant with respect to the input length. Each standby bin has a non-reserved space of size

$1 - \frac{2(\eta-1)}{7\eta-1} = \frac{5\eta+1}{7\eta-1}$, which is used to pack SRs of size at least $\frac{1}{7\eta-1}$. As such, we have $m \leq 5\eta + 1$. So, all primary small bins, except for at most $5\eta + 1 + f \in O(1)$ of them, have weight at least 1. Standby small bins have a reserved space of $\frac{2(\eta-1)}{7\eta-1}$. Except for the bins inside the incomplete groups, other bins have an additional empty space of at most $\frac{2}{7\eta-1}$, giving them a total empty space of at most $\frac{2\eta}{7\eta-1}$. By Lemma 6.1, there are up to $f + 1$ incomplete groups, each containing f mirroring bins. Therefore, the filled space in each standby bin, except for at most $f(f + 1) = O(1)$ of them, is at least $\frac{5\eta-1}{7\eta-1} \geq 2/3$. Given that any standby small replica of size s has weight $1.5s$, the weight of any of these standby bins is then at least $1.5\ (3/2) = 1$. ■

Lemma 6.4. *The total weight of items in any bin of an optimal packing is at most 1.75.*

Proof. Define the *density* of each item as the ratio between the weight and the size of the item. A primary replica of class $i \leq 6$ has a size in the range $(\frac{1}{i+1}, \frac{1}{i}]$ and weight $1/i$, which gives a density of at most $\frac{i+1}{i}$. Similarly, standby replicas of class $j \leq \lfloor 6\eta \rfloor$ have size in the range $(\frac{1}{j+\eta}, \frac{1}{j+\eta-1}]$ and weight $1/j$, giving them a density of at most $(j + \eta)/j$. Small replicas (both primary and standby) have a density of 3/2. We consider three possible cases and show that the total weight of items in any bin B^* of an optimal packing is at most 1.75 in each case. To follow the proof, it helps to consult Table 6.1.

Case 1: No standby replica in B^*: Suppose B^* does not include any standby replica. In this case, B^* includes 0 or 1 primary replica of class 1 (it cannot include more than 1 such replica since all replicas of class 1 have sizes larger than 1/2). If it includes no replica of class 1, the density of each of the items (of other classes) is at most 1.5, giving a total weight of at most 1.5 for items in B^*. If B^* includes one item of class 1 (with weight 1), the total size of other items will be less than 1/2, and since their density is at most 3/2, their total weight will be no more than $3/2 \cdot 1/2 = 3/4$, giving a total weight of at most $1 + 3/4 = 1.75$ for items in B^*.

Case 2: Some standby replicas in B^* are regular: Suppose B^* includes at least one regular standby replica. Let x be the largest standby replica in B^* and j

denote the class of x; we have $1 \leq j \leq \lfloor 6\eta \rfloor$. There should be enough empty space in B^* so that if all f bins containing other replicas of x are failed, x can be declared as a primary replica. Increasing the size of x by a factor η should not cause an overflow, that is, there should be empty space of at least $(\eta-1)x > (\eta-1)/(j+\eta)$ in B^* (recall that replicas of class j are of sizes at least $1/(j+\eta)$). So, the total size of items in B^* is less than $1 - \frac{\eta-1}{j+\eta} = \frac{j+1}{j+\eta}$. There are two cases to consider: either $j = 1$ or $j \geq 2$:

i) Suppose $j = 1$, that is, there is a standby replica x of size more than $1/(1+\eta)$ in B^*. The total size of items in the bin is less than $\frac{j+1}{j+\eta} = \frac{2}{\eta+1}$, and items other than x in B^* have a total size less than $\frac{1}{\eta+1}$. As a result, there is no primary replica of class 1 (of size at least $1/2 > \frac{1}{\eta+1}$) or standby replica of class 1 (of size more than $\frac{1}{\eta+1}$) in B^*. So, primary replicas in B^* have class 2 or more and hence density at most 1.5. Similarly, standby replicas other than x have class 2 or more and hence density no more than $\frac{\eta+2}{2}$. So, all replicas other than x in B^* have a density at most $\max\{1.5, \frac{\eta+2}{2}\} = \frac{\eta+2}{2}$. Since the total size of these replicas is at most $\frac{1}{\eta+1}$, their total weight is at most $\frac{1}{\eta+1} \cdot \frac{\eta+2}{2} = \frac{\eta+2}{2\eta+2}$. Adding the weight 1 of x, the total weight of replicas in B^* is at most $\frac{3\eta+4}{2\eta+2}$, which is at most 1.75, given that $\eta > 1$.

ii) Suppose $j \geq 2$. Recall that the total size of items in B^* is less than $\frac{j+1}{j+\eta}$. First, assume there is also a primary replica y of class 1 in B^*. This is possible only if $\frac{1}{j+\eta} + \frac{1}{2} < \frac{j+1}{j+\eta}$, that is, $\eta < j$. The size of replicas other than y in B^* is less than $\frac{j+1}{j+\eta} - 1/2 = \frac{j+2-\eta}{2j+2\eta}$, and their density is at most $\max\{1.5, (j+\eta)/j\}$ (primary replicas of class ≥ 2 have density at most 3/2, and standby replicas have density at most $(j+\eta)/j$). The total weight of replicas other than y is hence less than $\max\{\frac{3j+6-3\eta}{4j+4\eta}, \frac{j+2-\eta}{2j}\} \leq \max\{3/4, \frac{j+2-\eta}{2j}\}$, which is at most 0.75, given that $\eta > 1$ and $j \geq 2$. Adding the weight 1 of y, the total weight of replicas in B^* will not be more than 1.75. Next, assume there is no primary replica of class 1 in B^*. In this case, the total size of replicas in B^* is at most $\frac{j+1}{j+\eta}$, and their density is at most $\max\{1.5, (j+\eta)/j\}$, giving them a total weight of at most $\max\{\frac{3j+3}{2j+2\eta}, (j+1)/j\}$, which is at most $\max\{1.5, (j+\eta)/j\}$ = 1.5, given that $\eta > 1$ and $j \geq 2$.

Case 3: All standby replicas in B^* are small: Assume there is no regular standby replica in B^*, but there is at least one small standby replica in B^*. We consider two cases: either there is a primary replica of class 1 in B^* or not:

i) If there is a primary replica y of class 1 in B^*, the remaining space of B^* is less than $1/2$ (as y is of size more than $1/2$). No other primary replica y' of class 1 can be in B^* because each of y and y' would have a size more than $1/2$. Therefore, the remaining space in B^* (of size less than $1/2$) can be filled with primary replicas of class $i \geq 2$ (of the density of at most $3/2$) and with other standby small replicas (of density $3/2$). So, the total weight of items other than y in B^* is at most $1.5(1/2) = 3/4$. Given that the weight of y is 1, the total weight of items in B^* will be at most 1.75.

ii) If there is no primary replica of class 1 in B^*, then B^* is filled with primary replicas of class $i \geq 2$ (of density at most $3/2$) and standby replicas of class $j = \lfloor 6\eta \rfloor + 1$ (of the density of $3/2$). As a result, the total weight of B^* will be no more than $3/2$.

■

Theorem 6.2. *Harmonic-Stretch has a competitive ratio of at most 1.75.*

Proof. Let σ be any input sequence, and $w(\sigma)$ be the total weight of items in σ. Let $HStr(\sigma)$ be the number of bins that Harmonic-Stretch opens for σ. By Lemma 6.3, we have $HStr(\sigma) \leq w(\sigma) + c$ for some constant c independent of $|\sigma|$. On the other hand, by Lemma 6.4, we have $\textsc{Opt}(\sigma) \geq w(\sigma)/1.75$. We can write

$$\frac{HStr(\sigma)}{Opt(\sigma)} \leq \frac{w(\sigma) + c}{w(\sigma)/1.75'},$$

which converges to 1.75, given that c is a constant. ■

6.3 Concluding remarks

We proved that the competitive ratio of Harmonic-Stretch is at most 1.75, which is an improvement over the competitive ratio 2 of the best existing algorithm. We

note that this upper bound holds for all values of f and η. When η is close to 1, the existing lower bounds for the classic online bin packing extend to the fault-tolerant setting. In particular, no fault-tolerant bin packing algorithm can achieve a competitive ratio better than 1.54 [63, 64]. As a topic for future work, one may consider tightening the gap between the lower bound of 1.54 and the upper bound of 1.75.

Chapter 7

Conclusion

In this thesis, we studied two variants of the bin packing problem, that is, square packing, which is a variant of the two-dimensional bin packing, and fault-tolerant bin packing, which has wide applications in cloud systems. Packing squares into squares of unit size is a well-studied variant of the bin packing problem. It has been, however, studied only in a setting where square-items are not allowed to be rotated. We considered, for the first time, square packing in the presence of item rotations, which is a more realistic assumption for applications such as stock cutting. We first proved that the problem is NP-Hard, and then provided an APTAS for a relaxed augmented setting where bins have a capacity of $(1 + \alpha)$ for some small $\alpha > 0$. We also studied the online setting of the problem where square items are revealed one by one (which is another realistic assumption), and introduced an online algorithm with a competitive ratio of at most 2.306. We also considered a similar problem in which the goal is to pack tans (half-square triangles) of various sizes into unit square bins. With a similar approach, we presented an online algorithm with a competitive ratio of at most 1.897 for this problem.

We also studied the fault-tolerant bin packing problem, which has applications in server consolidation in the Cloud. In this problem, bins represent servers of unit capacity, and items are databases (tenants) of various workloads hosted in the Cloud servers. The problem is online by nature, as requests for hosting tenants appear in an online manner. We introduced a practical and general model in which servers might fail and recover before all jobs/tenants (items) are assigned (packed) into servers.

In the existing models for the problem, server failures happen only after all items are packed. We designed an algorithm with a competitive ratio at most 1.75, and showed that it is also valid for the (less practical) existing models, where the best existing algorithm had a competitive ratio of at most 2.

7.1 Future work

For the offline square packing with rotation problem, we showed that the problem admits an APTAS under a resource augmented setting. As a topic for future work, one might investigate if such a result also holds when bin sizes are not augmented.

For online square packing with rotation, we presented an upper bound of at most 2.306. This competitive ratio might be improved by allowing square-items of similar classes placed in the same bins as in, e.g., the Modified-Harmonic algorithm [18] for the 1-dimensional bin packing problem. Another potential topic for future work concerns providing lower bounds for the competitive ratio attainable by online algorithms for this problem.

As for the online fault-tolerant bin packing problem, we introduced an algorithm with a competitive ratio of at most 1.75. It is easy to derive a lower bound of 1.54 (from 1-dimensional bin packing). Therefore, there is a gap between the two bounds. Closing/tightening this gap can be a topic for future work.

Bibliography

[1] J. L. Bentley and C. C. McGeoch, "Amortized analyses of self-organizing sequential search heuristics," *Communications of the ACM*, vol. 28, pp. 404–411, 1985. [Cited on page 2.]

[2] D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, pp. 202–208, 1985. [Cited on page 2.]

[3] L. Epstein and R. van Stee, "Online bin packing with resource augmentation," *Discrete Optimization*, vol. 4, no. 3-4, pp. 322–333, 2007. [Cited on pages 2 and 5.]

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the theory of of NP-Completeness*. Freeman and Company, 1979. [Cited on page 4.]

[5] W. F. de la Vega and G. Lueker, "Bin packing can be solved within $1 + \epsilon$ in linear time," *Combinatorica*, vol. 1, pp. 349–355, 1981. [Cited on page 4.]

[6] N. Karmarkar and R. M. Karp, "An efficient approximation scheme for the one-dimensional bin-packing problem," in *Annual Symposium on Foundations of Computer Science (FOCS)*, 1982, pp. 312–320. [Cited on page 4.]

[7] T. Rothvoß, "Approximating bin packing within o(log opt * log log opt) bins," in *Annual Symposium on Foundations of Computer Science*, 2013, pp. 20–29. [Cited on page 4.]

[8] D. S. Johnson, "Near-optimal bin packing algorithms," Ph.D. dissertation, MIT, Cambridge, MA, 1973. [Cited on page 5.]

[9] L. Epstein and A. Ganot, "Optimal on-line algorithms to minimize makespan on two machines with resource augmentation," *Theory of Computing Systems*, vol. 42, no. 4, pp. 431–449, 2008. [Cited on page 5.]

[10] J. Boyar, L. Epstein, and A. Levin, "Tight results for Next Fit and Worst Fit with resource augmentation," *Theoretical Computer Science*, vol. 411, no. 26-28, pp. 2572–2580, 2010.

[11] D. R. Kowalski, P. W. H. Wong, and E. Zavou, "Fault tolerant scheduling of tasks of two sizes under resource augmentation," *Journal of Scheduling*, vol. 20, no. 6, pp. 695–711, 2017.

[12] J. Erickson, I. van der Hoog, and T. Miltzow, "A framework for robust realistic geometric computations," *CoRR*, 2019. [Cited on pages 5, 21, 22, 26, and 27.]

[13] S. Albers, "Better bounds for online scheduling," *SIAM Journal on Computing*, vol. 29, no. 2, pp. 459–473, 1999. [Cited on page 5.]

[14] R. Fleischer and M. Wahl, "Online scheduling revisited," in *Proceedings of the 8th Annual European Symposium*, vol. 1879, 2000, pp. 202–210.

[15] S. Albers and M. Hellwig, "Online makespan minimization with parallel schedules," *Algorithmica*, vol. 78, no. 2, pp. 492–520, 2017. [Cited on page 5.]

[16] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on computing*, vol. 3, pp. 256–278, 1974. [Cited on page 5.]

[17] C. C. Lee and D. T. Lee, "A simple online bin packing algorithm," *Journal of the ACM (JACM)*, vol. 32, pp. 562–572, 1985. [Cited on pages 5 and 6.]

[18] P. V. Ramanan., D. J. Brown, C.-C. Lee, and D.-T. Lee, "On-line bin packing in linear time," *Journal of Algorithms*, vol. 10, pp. 305–326, 1989. [Cited on pages 6 and 67.]

[19] S. S. Seiden, "On the online bin packing problem," *Journal of the ACM (JACM)*, vol. 49, pp. 640–671, 2002. [Cited on page 6.]

[20] S. Heydrich and R. van Stee, "Beating the harmonic lower bound for online bin packing," in *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016. [Cited on page 6.]

[21] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin, "A new and improved algorithm for online bin packing," in *Annual European Symposium on Algorithms (ESA)*, 2018, pp. 5:1–5:14. [Cited on page 6.]

[22] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin, "A new lower bound for classic online bin packing," in *International Workshop on Approximation and Online Algorithms*. Springer, 2019, pp. 18–28. [Cited on page 6.]

[23] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali, "Approximation and online algorithms for multidimensional bin packing: A survey," *Computer Science Review*, vol. 24, pp. 63–79, 2017. [Cited on page 7.]

[24] 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*, vol. 10, no. 3, pp. 271–275, 1990. [Cited on pages 7, 18, and 20.]

[25] N. Bansal, J. R. Correa, C. Kenyon, and M. Sviridenko, "Bin packing in multiple dimensions: Inapproximability results and approximation schemes," *Mathematics of Operations Research*, vol. 31, no. 1, pp. 31–49, 2006. [Cited on pages 7, 20, 22, 28, and 29.]

[26] D. Coppersmith and P. Raghavan, "Multidimensional on-line bin packing: algorithms and worst-case analysis," *Operations Research Letters*, vol. 8, no. 1, pp. 17–20, 1989. [Cited on page 7.]

[27] S. S. Seiden and R. van Stee, "New bounds for multidimensional packing," *Algorithmica*, vol. 36, no. 3, pp. 261–293, 2003.

[28] L. Epstein and R. van Stee, "Online square and cube packing," *Acta Informatica*, vol. 41, no. 9, pp. 595–606, 2005. [Cited on page 7.]

[29] X. Han, D. Ye, and Y. Zhou, "A note on online hypercube packing," *Central European Journal of Operations Research (CEJOR)*, vol. 18, no. 2, pp. 221–239, 2010. [Cited on page 7.]

[30] D. Blitz, S. Heydrich, R. van Stee, A. van Vliet, and G. J. Woeginger, "Improved lower bounds for online hypercube and rectangle packing," *arXiv preprint arXiv:1607.01229*, 2017. [Cited on page 7.]

[31] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin, "Lower bounds for several online variants of bin packing," *Theory of Computing Systems*, vol. 63, no. 8, pp. 1757–1780, 2019. [Cited on page 7.]

[32] L. Epstein, "Two-dimensional online bin packing with rotation," *Theoretical Computer Science*, vol. 411, no. 31-33, pp. 2899–2911, 2010. [Cited on page 7.]

[33] S. Kamali, A. López-Ortiz, and Z. Rahmati, "Online packing of equilateral triangles," in *The Canadian Conference on Computational Geometry (CCCG)*, 2015. [Cited on page 7.]

[34] EC2, "Amazon EC2 instance types." [Online]. Available: https://aws.amazon.com/ec2/ [Cited on page 8.]

[35] Azure, "Azure virtual machine series." [Online]. Available: https://azure.microsoft.com/en-ca/ [Cited on page 8.]

[36] L. Wu and R. Buyya, "Service level agreement (sla) in utility computing systems," in *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. IGI Global, 2012, pp. 1–25. [Cited on page 8.]

[37] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation," in *Proc. the 33rd International Computer Measurement Group Conference (CMG)*, 2007, pp. 399–406. [Cited on page 8.]

[38] H. Yanagisawa, T. Osogami, and R. Raymond, "Dependable virtual machine allocation," in *Proc. the 32nd IEEE International Conference on Computer Communications (INFOCOM)*, 2013, pp. 629–637. [Cited on page 8.]

[39] J. Schaffner, T. Januschowski, M. Kercher, T. Kraska, H. Plattner, M. J. Franklin, and D. Jacobs, "RTP: robust tenant placement for elastic in-memory database clusters," in *International Conference on Management of Data (SIGMOD13)*, 2013, pp. 773–784. [Cited on page 8.]

[40] K. Daudjee, S. Kamali, and A. López-Ortiz, "On the online fault-tolerant server consolidation problem," in *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2014, pp. 12–21. [Cited on page 9.]

[41] C. Li and X. Tang, "Brief announcement: Towards fault-tolerant bin packing for online cloud resource allocation," in *Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2017, pp. 231–233. [Cited on pages 9, 16, and 49.]

[42] ——, "On fault-tolerant bin packing for online resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 817–829, 2019. [Cited on pages 10 and 14.]

[43] B. Li, Y. Dong, B. Wu, and M. Feng, "An online fault tolerance server consolidation algorithm," in *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 2021, pp. 458–463. [Cited on page 10.]

[44] E. Friedman, "Tans in squares." [Online]. Available: https://erich-friedman.github.io/packing/taninsqu/ [Cited on page 14.]

[45] C. Li and X. Tang, "On fault-tolerant bin packing for online resource allocation," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 31, no. 4, pp. 817–829, 2020. [Cited on pages 16 and 49.]

[46] S. Kamali and P. Nikbakht, "Cutting stock with rotation: Packing square items into square bins," in *Proceedings of the 14th International Conference on*

Combinatorial Optimization and Applications (COCOA), ser. Lecture Notes in Computer Science, W. Wu and Z. Zhang, Eds., vol. 12577. Springer, 2020, pp. 530–544. [Online]. Available: https://doi.org/10.1007/978-3-030-64843-5_36 [Cited on page 17.]

[47] P. Erdős and R. L. Graham, "On packing squares with equal squares," *Journal of Combinatorial Theory, Series A*, vol. 19, pp. 119–123, 1975. [Cited on pages 18, 21, and 35.]

[48] M. Abrahamsen, T. Miltzow, and N. Seiferth, "Framework for ∃r-completeness of two-dimensional packing problems," in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2020, pp. 1014–1021. [Cited on pages 18 and 21.]

[49] E. D. Demaine, S. P. Fekete, and R. J. Lang, "Circle packing for origami design is hard," *CoRR*, vol. abs/1008.1224, 2010. [Cited on page 18.]

[50] A. Chou, "NP-hard triangle packing problems," *manuscript*, 2016. [Cited on page 18.]

[51] S. R. Allen and J. Iacono, "Packing identical simple polygons is NP-hard," *CoRR*, vol. abs/1209.5307, 2012. [Cited on page 19.]

[52] F. Göbel, "Geometrical packing and covering problems," *Math Centrum Tracts*, vol. 106, pp. 179–199, 1979. [Cited on page 21.]

[53] W. Stromquist, "Packing 10 or 11 unit squares in a square," *The Electronic Journal of Combinatorics*, pp. R8–R8, 2003.

[54] T. Gensane and P. Ryckelynck, "Improved dense packings of congruent squares in a square," *Discret. Comput. Geom.*, vol. 34, no. 1, pp. 97–109, 2005.

[55] E. Friedman, "Packing unit squares in squares: A survey and new results," *The Electronic Journal of Combinatorics*, vol. 1000, pp. DS7–Aug, 2009. [Cited on pages vi, 21, and 22.]

[56] D. A. Spielman and S. Teng, "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time," *The Journal of the ACM (JACM)*. [Cited on page 21.]

[57] W. F. de la Vega and G. S. Lueker, "Bin packing can be solved within 1+ϵ in linear time," *Combinatorica*, vol. 1, no. 4, pp. 349–355, 1981. [Cited on pages 22 and 29.]

[58] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan, "Performance bounds for level-oriented two-dimensional packing algorithms," *SIAM Journal on Computing*, vol. 9, no. 4, pp. 808–826, 1980. [Cited on pages vii, 23, 31, and 32.]

[59] H. W. Lenstra, "Integer programming with a fixed number of variables," *Mathematics of Operations Research*, vol. 8, no. 4, pp. 538–548, 1983. [Cited on page 28.]

[60] E. Friedman, "Packing unit squares in squares: A survey and new results," *The Electronic Journal of Combinatorics*, pp. 1–24, 2000. [Cited on pages vii, ix, 35, 36, 37, 39, 40, 42, 43, and 44.]

[61] L. Epstein and R. van Stee, "Optimal online bounded space multidimensional packing," in *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, 2004, pp. 214–223. [Cited on pages 37, 38, and 45.]

[62] S. Kamali and P. Nikbakht, "On the fault-tolerant online bin packing problem," *CoRR*, vol. abs/2107.02922, 2021. [Online]. Available: https://arxiv.org/abs/2107.02922 [Cited on page 48.]

[63] J. Balogh, J. Békési, and G. Galambos, "New lower bounds for certain classes of bin packing algorithms," *Theoretical Computer Science (TCS)*, vol. 440–441, pp. 1–13, 2012. [Cited on page 65.]

[64] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin, "A new lower bound for classic online bin packing," in *Proc. the 17th International Workshop on*

Approximation and online algorithms (WAOA), vol. 11926, 2019, pp. 18–28. [Cited on page 65.]