More on online bin packing with two item sizes

Leah Epstein[*] Asaf Levin [†]

Abstract

We follow the work of [7] and study the online bin packing problem, where every item has one of two possible sizes which are known in advance. We focus on the parametric case, where both item sizes are bounded from above by $\frac{1}{k}$ for some natural number $k \geq 1$. We show that for every possible pair of item sizes, there is an algorithm with competitive ratio of at most $\frac{(k+1)^2}{k^2+k+1}$. We prove that this bound is tight for every k, and moreover, that it cannot be achieved if the two item sizes are not known in advance.

1 Introduction

Bin packing is a natural and well studied problem which has applications in problems of computer storage, bandwidth allocation, stock cutting, transportation and many other important fields. In the standard online problem, items of size in $(0, 1]$ arrive one by one to be assigned to unit size bins. Each bin may contain items of total size at most 1, and the goal is to minimize the number of bins used. It is typically assumed that items of any size may occur. However, in many applications, there is a small constant number of item sizes, which are known in advance, but it is unknown how many items of each size will arrive, and in which order.

The standard quality measure of algorithms for online bin packing is the *(asymptotic) competitive ratio*, which we now define. For a given input sequence σ, let $\mathcal{A}(\sigma)$ (or $\mathcal{A}$, if the sequence σ is clear from the context) be the number of bins used by algorithm $\mathcal{A}$ on σ. Let $OPT(\sigma)$ (or OPT) be the cost of an optimal offline algorithm which knows the complete sequence of items in advance, i.e., the minimum possible number of bins used to pack items in σ. The *competitive ratio* of an algorithm $\mathcal{A}$ is defined to be

$$\mathcal{R}(\mathcal{A}) = \limsup_{n\to\infty} \sup_{\sigma} \left\{ \frac{\mathcal{A}(\sigma)}{OPT(\sigma)} | OPT(\sigma) = n \right\} .$$

In this paper, we study the case where there are two possible item sizes. We focus on the parametric case, where both item sizes are bounded from above by $\frac{1}{k}$ for some integer $k \geq 1$. This problem in the online setting was studied by Gutin, Jensen and Yeo [7]. They showed that the overall best competitive ratio for this problem is $\frac{4}{3}$. This means that there exists a pair of sizes for which [7] have proved a lower bound of $\frac{4}{3}$ on the competitive ratio of any algorithm. Moreover, [7] designed a $\frac{4}{3}$-competitive algorithm for any pair of sizes. Moreover, they found the exact best possible competitive ratio for the case where at least one size is larger than $\frac{1}{2}$ (as a function of the two sizes).

The offline variant of the problem with two item sizes can be defined as follows. In this problem the input is given by the values α and $\beta \leq \alpha$, and two non-negative integers which are the numbers of α sized

[*]Department of Mathematics, University of Haifa, 31905 Haifa, Israel. `lea@math.haifa.ac.il`.

[†]Department of Statistics, The Hebrew University, Jerusalem, Israel. `levinas@mscc.huji.ac.il`.

items in the input and β sized items in the input. Interestingly, even with this compact description of the input, the problem can be solved in polynomial time using the algorithm of McCormick et al. [12].

Note that if the input for the online bin packing problem consists of items of two sizes only, but these two sizes are not known in advance, it was shown by Gutin, Jensen and Yeo [6] that the competitive ratio of any algorithm is at least 1.3871, therefore the advance knowledge of the sizes is crucial for $k = 1$. Note also that the lower bound of $\frac{4}{3}$ for two known in advance sizes was already shown in [5], where the case of two items sizes $\frac{2}{3} > \alpha > \frac{1}{2}$ and $\beta = 1 - \alpha$, was solved.

A natural question is how fast the competitive ratio decreases in the online problem, if the items are relatively small. An analysis of a similar flavor was applied to specific algorithms for standard bin packing as well as to other variants of bin packing (see e.g. [8, 9, 1, 4]).

Our main result is finding the best overall competitive ratio as a function of k, where $\frac{1}{k}$ is an upper bound on both item sizes. This problem is called "the parametric problem", since k is a parameter on which the competitive ratio may depend. We show that this ratio is $\frac{(k+1)^2}{k^2+k+1}$. Our result extends and simplifies the result of [7] for $k = 1$. The main difference is in the analysis of the algorithms for which we use weighting functions (see [15, 10, 13, 14]).

We further show that, if there are two item sizes, but they are not known in advance, then even if we know the parameter k and one out of the two sizes, and the other size is one of two values, which are also given in advance, already the earlier bound of $\frac{(k+1)^2}{k^2+k+1}$ cannot be achieved for any value of k.

To motivate further work on the subject, we show a pair of sizes for which tight bounds for this specific pair of values can be achieved using an algorithm which we describe, which is different from our general algorithm. Moreover, our general algorithm achieves a higher bound for this pair of sizes.

The classic online bin packing problem was first investigated by Ullman [15]. He showed that algorithm FIRST FIT (FF) has competitive ratio $\frac{17}{10}$. This result was then published in [9]. Johnson [8] showed that algorithm NEXT FIT (NF) has competitive ratio 2. The goal at this time was to find an algorithm which performs better than FF. Yao [17] designed an algorithm REVISED FIRST FIT which is based on FF with some changes and showed that it has competitive ratio of at most $\frac{5}{3}$.

The further sequence of improvements became possible after Lee and Lee [10] introduced the HARMONIC algorithm, which is based on harmonic partitioning of the input items, together with a greedy packing of the class of smallest items. This is actually a sequence of algorithms, whose competitive ratio tends to approximately 1.69103. All algorithms which were designed after this are based on adaptations of the HARMONIC algorithms. These algorithms try to combine selected items of different sizes to allow a better packing. Such were the algorithms REFINED HARMONIC [10], MODIFIED HARMONIC [13], and finally, the current champion, HARMONIC++, given by Seiden [14], which has an upper bound of 1.58889 on the competitive ratio.

As for lower bounds, Yao [17] showed that no online algorithm has competitive ratio smaller than $\frac{3}{2}$. Brown and Liang independently improved this lower bound to 1.53635 [2, 11]. This was subsequently improved by van Vliet to 1.54014 [16]. Chandra [3] showed that the preceding lower bounds also apply to randomized algorithms. The results of Gutin et al. [7] imply that the special case where there are only two item sizes known in advance, is significantly easier than the general online bin packing problem.

2 Definitions, notation and methods

Throughout the paper, we denote the two item sizes by α and β, where $\alpha \geq \beta$. We call items of size α, "larger items", and items of size β "smaller items". We use the notation x_i to denote the largest number of smaller items that can fit into a bin with i larger items. That is, x_i is the maximum integer value t such that $i\alpha + t\beta \leq 1$, or $x_i = \lfloor \frac{1-i\alpha}{\beta} \rfloor$. The value x_i is defined for $0 \leq i \leq \lfloor \frac{1}{\alpha} \rfloor$. Note that $x_0 = \lfloor \frac{1}{\beta} \rfloor$.

A pattern is a pair of non-negative integers (a, b). It describes a packing of a single bin, which has a larger items and b smaller items. It is valid if $a\alpha + b\beta \leq 1$.

To analyze bin packing algorithms, we use the well known weighting method. The weighting technique was originally introduced by Ullman [15]. The main idea of this method is to give an item a weight which is roughly the amount of space it occupies in a bin. The weight of an item is typically related to its size and for our purposes, the weight of an item is defined to be a function of its size. Since not all bins are fully packed, the weight of an item is typically (but not always) larger than its size. Given a weight measure w, then $w(s)$ is the *weight* of an item of size s according to w.

If there exist several possible scenarios in a given algorithm, we can define a weight function for each of them. Note that, if identical (or similar) items are packed in several different ways, we can assign equal weights to all of them, which is the average amount of space that such an item occupies.

The weights for a given scenario are assigned so that the number of bins used is no larger than the total weight assigned to all items (up to an additive constant). Given weights that satisfy this property, we use the following theorem, see Seiden [14]. Note that this type of analysis was done also earlier for algorithms which are adaptations of Harmonic, see [10, 13].

Lemma 1 *Consider a bin packing algorithm denoted by ALG. Let $w_1, \ldots, w_s$ be s weight measures. Assume that for every input sequence σ, there exists i ($1 \leq i \leq s$) such that the number of bins used by the algorithm ALG is at most $X_i(\sigma) + c$ for some constant c, where $X_i(\sigma)$ is the sum of weights of all items in the sequence according to w_i. Denote by W_i the supremum amount of weight that can be packed into a single bin, according to measure w_i ($1 \leq i \leq s$). Then, the asymptotic competitive ratio of the algorithm is at most $\max\limits_{1 \leq i \leq s} W_i$.*

Proof. Given an input, let i be the value that satisfies the assumptions of the theorem for this input. Clearly $OPT(\sigma) \geq \frac{X_i(\sigma)}{W_i}$. We get $ALG \leq X_i(\sigma) + c \leq W_i \cdot OPT + c$. ∎

3 Algorithms

For any pair of α, β, we use one of the following two algorithms. A similar approach of choosing one of two algorithms was used in [7]. We generalize it, and our analysis is different. The basic idea, which is to combine some fraction of relatively small items with larger items in the same bins, using specific patterns, is based on algorithms in [10, 13, 14].

Let $k = \lfloor \frac{1}{\alpha} \rfloor$ be the maximum number of larger items that can fit into a common bin, and let $s = x_0$ be the maximum number of smaller items that can fit into a common bin. We clearly have $\alpha \in (\frac{1}{k+1}, \frac{1}{k}]$ and $\beta \in (\frac{1}{s+1}, \frac{1}{s}]$. We also let $t = x_k$.

The following algorithm is used if $\frac{t}{s} \leq \frac{k}{k^2+k+1}$.

Algorithm Greedy. If $k = s$, use NF, that is, keep a single active bin. The active bin is closed and a new open bin is opened once the active bin contains exactly k items. If $s > k$, we keep two active bins, one for each size. Each bin for smaller items, except for possibly the last bin, will contain s items. Each bin for larger items, except for possibly the last bin, will contain k items.

The following algorithm is used if $\frac{t}{s} > \frac{k}{k^2+k+1}$. Note that this means that $t > 0$ and therefore $s > k$.

Algorithm Combine. The algorithm packs the larger items greedily, such that each bin, except for possibly the last one, contains k items. Let $\delta = \frac{t^2}{s^2-st+t^2}$. Clearly, since $0 < t < s$, we have $0 < \delta < 1$. A δ fraction of the smaller items is colored red and the remaining ones are colored blue. This can be implemented by partitioning the smaller items into blocks of s^2-st+t^2 consecutive items (ignoring the larger items) in each block (except perhaps the last block). Then, we color red the last t^2 items out of every s^2-st+t^2 items in the block, and all other items (the first s^2-st in the block) are colored blue. The blue items are packed s to a bin, whereas the red items are packed t to a bin. The goal is to combine the red items with the larger items. Thus, when a larger item arrives, if some bin already has larger items, but less than k such items, the new item is packed there. Otherwise, if there exists a bin with red items only, the new item is packed there. If none of these two conditions is satisfied, then a new bin is opened for the new item. Similarly, when a smaller item is colored red, the algorithm first tries to pack it into a bin which contains between 1 and $t-1$ smaller red items (this option exists only for $t \geq 2$), or if such a bin does not exist, the algorithm tries to pack the new item into a bin with at least one larger item and no red items. Finally, if none of these two conditions hold, i.e., such bins do not exist, a new bin is opened for red items.

The algorithm does not have more than one bin for larger items with less than k larger items. Similarly, there is at most one bin for blue smaller items with less than s items, and at most one bin for red smaller items with less than t red items. Note that by the definition of the algorithm, it is impossible to have at the same time both a bin with k larger items and no red items, and a bin with t red items and no larger items.

Weight functions

To analyze the algorithms we define three weight functions. One weight function, w_1, is defined for Greedy. Two functions are defined for Combine, specifically, one weight function is suitable for the case where there are no bins containing k larger items, but no red items. The last weight function is designed for the case where there are no bins containing t red items, but no blue items. We call these functions w_2 and w_3.

The functions are defined in Table 1.

x	$w_1(x)$	$w_2(x)$	$w_3(x)$
α	$\frac{1}{k}$	0	$\frac{1}{k}$
β	$\frac{1}{s}$	$\frac{1-\delta}{x_0}+\frac{\delta}{x_k}=\frac{s^2-st}{s(s^2-st+t^2)}+\frac{t^2}{t(s^2-st+t^2)}=\frac{s}{s^2-st+t^2}$	$\frac{1-\delta}{x_0}=\frac{s-t}{s^2-st+t^2}$

Table 1: The weight functions

Analysis

Theorem 2 *Given two item sizes $\beta \leq \alpha \leq \frac{1}{k}$, let $\rho(k)=\frac{(k+1)^2}{k^2+k+1}$. If $\frac{t}{s} \leq \frac{k}{k^2+k+1}$, then the competitive ratio of Greedy is at most $\rho(k)$. Otherwise, Combine has a competitive ratio of at most $\rho(k)$.*

Proof. We need to show that the sum of weights according to each weight function is at least the number of bins used by the algorithm, up to a constant additive factor. This constant factor is caused by bins that do not receive the full number of items, and thus is equal to at most 3; As we saw, for Greedy there are at most two such bins. For Combine, there are at most three such bins. We will compute the weight of every block of s^2-st+t^2 smaller items together. Thus, if the total number of such items is not divisible by s^2-st+t^2, we assume that it is divisible and neglect the last at most s^2-st+t^2-1 items, which may contribute to the additive constant as well. That is, in this case the additive constant is a function of $\frac{1}{\alpha}$ and $\frac{1}{\beta}$. (This situation can be avoided by keeping the ratio between the number of bins with blue smaller items, and the number of bins with red smaller items as close to $s-t:t$ as possible at all times. In this case, the additive constant becomes independent of the sizes.)

If $s>k$, then each full bin of Greedy has either k larger items or s smaller items, and thus has weight 1, according to w_1. If $s=k$, then every bin of Greedy (except for possibly the last one) has $k=s$ items, and therefore has weight 1.

For Combine, in the first scenario we need to check that the total weight of smaller items in all bins is large enough. We do not need to consider bins that contain only larger items, since such bins do not exist in this scenario. Every block of s^2-st+t^2 smaller items creates exactly $s-t$ bins of blue items and t bins of red items. The total weight of these items according to w_2 is s. In the second scenario, we need to only consider bins of smaller blue items and bins of larger items. The weight of larger items in a bin, according to w_3, is exactly 1. In order to prove that the total weight of all items is large enough, it suffices to show that the weight of s^2-st+t^2 smaller items in a block covers the cost of the bins of blue items (of the block), since red items are always combined with larger items. The total weight of these smaller items is $s-t$ and this is the number of bins with blue items (from the current block) that the algorithm creates.

Next, we need to find an upper bound on the total weight that can be packed into a single bin. There are $k+1$ valid patterns which we call dominant, according to the number of larger items in the bin, together with a maximum number of smaller items. The weight of a non-dominant pattern is at most the weight of some dominant pattern, thus it is enough to consider dominant patterns.

If $s=k$, then we use the Greedy algorithm and all items have weight $\frac{1}{k}=\frac{1}{s}$. A bin can contain at most k items, and we are done. In the remainder of the proof we assume $s>k$.

For the pattern $(0,s)$, the weight for w_1 is simply 1. Since $w_3(\beta)<w_2(\beta)$, the largest weight if Combine is used, is caused by w_2. The weight of this bin with respect to all three functions is therefore at most $\frac{s^2}{s^2-st+t^2}$. We are therefore interested in the function $f(z)=\frac{1}{z^2-z+1}$ where $z=\frac{t}{s}$. The derivative of f is given by $f'(z)=\frac{-(2z-1)}{(z^2-z+1)^2}$. Therefore, f is monotonically increasing in z if $z\le\frac{1}{2}$ (and monotonically decreasing otherwise). We next find bounds on $\frac{t}{s}$. Since $1-k\alpha<\alpha$, we conclude that $t\le\lfloor\frac{\alpha}{\beta}\rfloor$. On the other hand $s\ge t+k\lfloor\frac{\alpha}{\beta}\rfloor$, and therefore $s\ge(k+1)t$. Hence

$$\frac{1}{2}\ge\frac{1}{k+1}\ge\frac{t}{s}.$$

An upper bound on the weight is therefore given by using $z=\frac{1}{k+1}$. Therefore, the total weight is at most $\frac{(k+1)^2}{(k+1)^2-(k+1)+1}=1+\frac{k}{k^2+k+1}$.

Note that for any other pattern, the weight according to w_2 is lower than the weight of w_2 for $(0,s)$. That is, for all feasible pattern (a,b), we have $aw_2(\alpha)+bw_2(\beta)\le sw_2(\beta)$, and therefore we do not need to consider w_2 for other patterns.

For the pattern (k,t) we get that in Greedy, $kw_1(\alpha)+tw_1(\beta)=1+\frac{t}{s}\le1+\frac{k}{k^2+k+1}$, where the last inequality holds because Greedy is chosen. Otherwise, i.e., if the pattern is (k,t) and we chose Combine, then the total weight is at most $kw_3(\alpha)+tw_3(\beta)=1+\frac{t(s-t)}{s^2-st+t^2}=\frac{s^2}{s^2-st+t^2}\le1+\frac{k}{k^2+k+1}$ where the last inequality follows by the analysis of $f(z)$ above.

From now on, since $w_1(x)\ge w_3(x)$ for $x=\alpha,\beta$, it suffices to consider all remaining patterns with respect to w_1 only, if we do not use the value of $\frac{t}{s}$ (which determines which algorithm was used). We need to consider patterns (i,x_i) for $1\le i\le k-1$. Thus, we need to find an upper bound on $\frac{i}{k}+\frac{x_i}{s}$.

We make a case analysis depending on the value of s. First suppose that $s\ge2k+2$. Note that $x_i\le\frac{1-i\alpha}{\beta}<\frac{1-\frac{i}{k+1}}{\frac{1}{s+1}}=\frac{(s+1)(k+1-i)}{k+1}$ and we need to find a bound for $\frac{i}{k}+\frac{x_i}{s}\le\frac{i}{k}+\frac{(s+1)(k+1-i)}{s(k+1)}=1+\frac{1}{s}+$

$i\frac{s(k+1)-k(s+1)}{sk(k+1)} = 1 + \frac{1}{s} + i\frac{s-k}{sk(k+1)}$. Since $s > k$, the worst case (i.e., the case which causes the expression to be maximized) occurs for $i = k-1$ (i.e., the maximum value of i). Therefore, $1 + \frac{(k-1)(s-k)+k(k+1)}{sk(k+1)} = 1 + \frac{ks-s+2k}{sk(k+1)} = 1 + \frac{k-1}{k(k+1)} + \frac{2}{s(k+1)}$ is an upper bound on the total weight in a bin.

Since the last expression is a monotonically decreasing function of s (for positive values of s), and since $s \geq 2k+2$, we substitute and get an upper bound on the total weight of $1 + \frac{k-1}{k(k+1)} + \frac{2}{(2k+2)(k+1)} = 1 + \frac{k^2-1+k}{k(k+1)^2}$. Since we have $(k^2+k-1)(k^2+k+1) = k^2(k+1)^2 - 1 < k^2(k+1)^2$, we get $\frac{k^2-1+k}{k(k+1)^2} < \frac{k}{k^2+k+1}$, which completes the proof for the case $s \geq 2k+2$.

Next, assume $s \leq 2k+1$, and let $c = i + x_i - k$. A bin packed by a dominant pattern contains at least k items, thus $x_i \geq k - i$, and therefore $c \geq 0$. We can determine the possible values of c that may violate the competitive ratio as follows. Since (i, x_i) is a valid pattern and $\alpha > \frac{1}{k+1}$, $\beta > \frac{1}{s+1}$, we conclude that $\frac{i}{k+1} + \frac{k+c-i}{s+1} < 1$. Therefore, $i\left(\frac{1}{k+1} - \frac{1}{s+1}\right) < \frac{s+1-k-c}{s+1}$. From this we get $i < \frac{(k+1)(s+1-k-c)}{s-k}$. If the competitive ratio is violated, we get $\frac{(k+1)^2}{k^2+k+1} < \frac{i}{k} + \frac{k+c-i}{s} = \frac{k+c}{s} + \frac{i(s-k)}{ks} < \frac{k+c}{s} + \frac{(k+1)(s+1-k-c)}{ks} = \frac{k^2+kc+ks+s+k+1-k^2-k-kc-c}{ks} = \frac{ks+s+1-c}{ks}$. Therefore,

$$c < ks+s+1-\frac{(k+1)^2ks}{k^2+k+1} = 1+s(k+1)\cdot\left(1 - \frac{(k+1)k}{k^2+k+1}\right) = 1+\frac{s(k+1)}{k^2+k+1} \leq 1+\frac{(2k+1)(k+1)}{k^2+k+1} < 4.$$

We therefore need to consider the cases $c = 0, 1, 2, 3$. We let $1 \leq j = k - i \leq k-1$ and $1 \leq d = s - k \leq k+1$. Then, $x_i = k + c - i = j + c$.

If $c \leq 1$, since $s \geq k+1$ and $i \leq k-1$, we conclude that $\frac{i}{k} + \frac{x_i}{s} \leq \frac{i}{k} + \frac{k+1-i}{k+1} \leq \frac{i}{k^2+k} + 1 \leq \frac{k^2+k+k-1}{k^2+k} < \frac{(k+1)^2}{k^2+k+1}$, where the last inequality holds because $(k^2+k+k-1)(k^2+k+1) = k^4+3k^3+2k^2+k-1 < k^4+3k^3+3k^2+k = (k^2+k)(k+1)^2$.

If $c = 2$, then $\frac{i}{k} + \frac{x_i}{s} = \frac{k-j}{k} + \frac{j+2}{k+d} = 1 + \frac{2k-jd}{k(k+d)}$. We first argue that $k \leq jd+d-2$. To see this note that otherwise if $k \geq jd+d-1$, we get that $k-j$ items larger than $\frac{1}{k+1}$ and $j+2$ items larger than $\frac{1}{s+1}$ do not fit into one bin, so the pattern $(i, x_i) = (k-j, j+2)$ is impossible. This is true since in order to have $\frac{k-j}{k+1} + \frac{j+2}{d+k+1} < 1$ we need $\frac{j+1}{k+1} - \frac{j+2}{d+k+1} > 0$ or $0 < jd+jk+j+d+k+1-jk-2k-j-2 = jd+d-1-k$, and this is a contradiction to the assumption on k. Therefore, we can assume that $k \leq jd+d-2$.

Recall that $d \leq k+1$. First assume that $d \leq k$. We conclude that $1 + \frac{2k-jd}{k(k+d)} \leq 1 + \frac{k+d-2}{k^2+kd} = 1 + \frac{1}{k} - \frac{2}{k(k+d)} \leq 1 + \frac{k}{k^2} - \frac{1}{k^2} = 1 + \frac{k-1}{k^2} \leq 1 + \frac{k}{k^2+k+1}$, where the first inequality holds because $k \leq jd+d-2$, the second inequality holds because $d \leq k$, and the third inequality holds because $(k-1)(k^2+k+1) = k^3 - 1 < k \cdot k^2$. Next, assume that $d = k+1$. Then, for $j \geq 2$, $1 + \frac{2k-jd}{k(k+d)} < 1$. Therefore, we need to consider the case where $j = 1$. In this case the competitive ratio is at most $1 + \frac{k-1}{k(2k+1)} < 1 + \frac{k}{k^2+k+1}$, where the inequality holds because $(k-1)(k^2+k+1) = k^3 - 1 < k^2(2k+1)$.

If $c = 3$, then we get $\frac{i}{k} + \frac{x_i}{s} = \frac{k-j}{k} + \frac{j+3}{k+d} = 1 + \frac{3k-jd}{k(k+d)}$.

If $k \geq \frac{jd+d-2}{2}$, we get that $k-j$ items larger than $\frac{1}{k+1}$ and $j+3$ items larger than $\frac{1}{s+1}$ do not fit into one bin so the pattern $(i, x_i) = (k-j, j+3)$ is impossible. In fact, in order to have $\frac{k-j}{k+1} + \frac{j+3}{d+k+1} < 1$ we need $\frac{j+1}{k+1} - \frac{j+3}{d+k+1} > 0$ or $0 < jd+jk+j+d+k+1-jk-3k-j-3 = jd+d-2-2k$ which gives $2k < jd+d-2$ or $k < \frac{jd+d-2}{2}$, and this is a contradiction. Therefore, we can assume that $k < \frac{jd+d-2}{2}$, i.e., $k \leq \frac{jd+d-3}{2}$.

Using $2k \leq jd+d-3$, we have $1 + \frac{3k-jd}{k(k+d)} \leq 1 + \frac{k+d-3}{k(k+d)} = 1 + \frac{1}{k} - \frac{3}{k(k+d)}$. Recall that $d \leq k+1$, which gives $1 + \frac{3k-jd}{k(k+d)} \leq 1 + \frac{1}{k} - \frac{3}{k(k+d)} \leq 1 + \frac{1}{k} - \frac{3}{k(2k+1)} = 1 + \frac{2k-2}{2k^2+k} \leq 1 + \frac{k}{k^2+k+1}$, where the last inequality follows from $2(k-1)(k^2+k+1) = 2k^3 - 2 < 2k^3 < 2k^3 + k^2$. ■

4 Lower bound

In this section we show that the upper bound as a function of k cannot be improved and prove the following theorem.

Theorem 3 *For every $k \geq 1$, there exists an input where $\beta \leq \alpha \leq \frac{1}{k}$, such that any online algorithm has competitive ratio of at least $\frac{(k+1)^2}{k^2+k+1}$.*

Proof. Let $\varepsilon > 0$ be a small number such that $\varepsilon \leq \frac{1}{(k+1)^2(k+2)}$. We consider an input in which the item sizes are $\alpha = \frac{1}{k+1} + \varepsilon$ and $\beta = \frac{1}{k+2} + \varepsilon$. Clearly, a bin can contain at most k larger items.

Let N be a large enough integer. The input starts with a first step where $N \cdot x_0$ smaller items arrive. Let $x_{k+1} = 0$. For $0 \leq j \leq k$, we denote by y_j the number of bins which contain at least $x_{j+1} + 1$ items and at most x_j items after the first step. Clearly, such a bin can accommodate at most j larger items. In the second step, $kN \cdot x_0$ larger items arrive. We denote by y the number of additional bins opened in this step. Denote by OPT_1 and OPT_2 the optimal costs after the two steps. Denote by ALG_1 and ALG_2 the costs of the online algorithm after the two steps. Let $\mathcal{R}$ be the competitive ratio of the online algorithm. Thus for $i = 1, 2$ we have $\text{ALG}_i \leq \mathcal{R}\text{OPT}_i$. Thus we get the following inequalities.

$$\sum_{i=0}^{k} y_i \leq \mathcal{R}\text{OPT}_1 \quad \text{and} \quad \sum_{i=0}^{k} y_i + y \leq \mathcal{R}\text{OPT}_2. \tag{1}$$

Next, using the quantities of items, we get the following inequalities.

$$\sum_{i=0}^{k} x_i \cdot y_i \geq N x_0 \quad \text{and} \quad \sum_{i=0}^{k} i \cdot y_i + yk \geq Nkx_0. \tag{2}$$

Next, we compute the values x_i. Note that all items are larger than $\frac{1}{k+2}$, so a bin contains at most $k+1$ items. Therefore $x_i \leq k+1-i$. On the other hand, since $\alpha > \beta$, and $\varepsilon \leq \frac{1}{(k+1)^2(k+2)}$, we have $i\alpha + (k+1-i)\beta \leq k\alpha + \beta = \frac{k}{k+1} + \frac{1}{k+2} + (k+1)\varepsilon \leq \frac{k^2+2k+k+1+1}{(k+1)(k+2)} = 1$. Thus $x_i = k+1-i$. Using these values we can deduce, $\text{OPT}_1 = N$ and $\text{OPT}_2 = N \cdot x_0 = N(k+1)$.

Adding the inequalities in (2), we get $(k+1)\sum\limits_{i=0}^{k} y_i + ky \geq (k+1)^2 N$.

Using the sum of the inequalities in (1), where the second one is multiplied by k, we get $(k+1)\sum\limits_{i=0}^{k} y_i + ky \leq \mathcal{R}k(k+1)N + \mathcal{R}N = \mathcal{R}N(k^2+k+1)$.

Combining the last two inequalities implies a lower bound of $\frac{(k+1)^2}{k^2+k+1}$ on $\mathcal{R}$.

Note that this proof holds for randomized algorithms as well as deterministic ones, since the variables denoting numbers of bins can be seen as variables denoting the expected numbers of bins. ■

5 Unknown item sizes

In this section we demonstrate that if the two item sizes are not known in advance, the resulting competitive ratio is higher. We show that already a small lack of knowledge causes an increase of the competitive ratio. Specifically, we show that if the size β is known in advance, and the size α can be one of two given sizes in the interval $(\frac{1}{k+1}, \frac{1}{k}]$, which are both known in advance, then the competitive ratio is at least $Q(k) = \frac{4k^6+8k^5+10k^4+14k^3+11k^2+6k+3}{4k^6+4k^5+10k^4+8k^3+9k^2+4k+2} = 1 + \frac{4k^5+6k^3+2k^2+2k+1}{4k^6+4k^5+10k^4+8k^3+9k^2+4k+2}$. It is easy to verify that $(4k^5 + 6k^3 + 2k^2 + 2k + 1) \cdot (k^2 + k + 1) = 4k^7 + 4k^6 + 10k^5 + 8k^4 + 10k^3 + 5k^2 + 3k + 1 > 4k^7 + 4k^6 + 10k^5 + 8k^4 + 9k^3 + 4k^2 + 2k$ and thus $Q(k) > 1 + \frac{k}{k^2+k+1}$. Our bound holds in the worst case, and not for every triple of values (where the triple consists of β, and the two options of α).

We state some values of $Q(k)$ in Table 2.

k	$\frac{(k+1)^2}{k^2+k+1}$	$Q(k)$
2	$\frac{9}{7} \approx 1.28571$	1.28899
3	$\frac{16}{13} \approx 1.23077$	1.23138
4	$\frac{25}{21} \approx 1.19048$	1.19065
5	$\frac{36}{31} \approx 1.16129$	1.16135

Table 2: Comparison between the best competitive ratio with two known sizes and $Q(k)$

Our construction is a generalization of the construction for a small number of sizes in [6] and reduces to it for $k = 1$.

Let $p > k$ be an integer and let $\varepsilon > 0$ be a value such that $\varepsilon < \frac{1}{p^2k(k+1)^3}$. We use the sizes $\beta = \frac{1}{p(k+1)} - \varepsilon$, $\alpha_1 = \frac{1}{k+1} + \varepsilon$, $\alpha_2 = \frac{1}{k+1} + \frac{1}{pk(k+1)} = \frac{pk+1}{pk(k+1)}$. We have $\alpha_1 < \alpha_2 < \frac{1}{k}$.

The first part of the input is N items of size β, where N is a large enough number, divisible by $p(p-1)(k+1)$. For convenience, we allow the algorithm to use a non-integer number of bins packed by each pattern. We do not allow the offline algorithm to do so, therefore this can only reduce the competitive ratio.

Let x_i^j be the maximum number of smaller items that can fit into a bin with i items of size α_j, for $0 \le i \le k$.

Claim 4 *For $i > 0$, $x_i^1 = p(k+1-i)$ and $x_i^2 = p(k+1-i) - 1$. Moreover, $x_{i+1}^1 < x_i^2$ for $1 \le i \le k-1$.*

Proof. Using $i \le k < p$, we conclude that

$$\begin{aligned}
i\alpha_1 + p(k+1-i)\beta &= \frac{i}{k+1} + i\varepsilon + \frac{p(k+1-i)}{p(k+1)} - p(k+1-i)\varepsilon \\
&= 1 - p(k+1)\varepsilon + i(p+1)\varepsilon \le 1 + (k-p)\varepsilon < 1, \\
i\alpha_1 + (p(k+1-i)+1)\beta &= 1 + \frac{1}{p(k+1)} - p(k+1)\varepsilon + i(p+1)\varepsilon - \varepsilon \\
&> 1 + \frac{1}{p(k+1)} - \frac{p(k+1)+1}{p^2k(k+1)^3} > 1, \\
i\alpha_2 + (p(k+1-i)-1)\beta &= i(\frac{1}{k+1} + \frac{1}{pk(k+1)}) + \frac{p(k+1-i)-1}{p(k+1)} \\
&- (p(k+1-i)-1)\varepsilon < 1 + \frac{i-k}{pk(k+1)} \le 1, \\
i\alpha_2 + p(k+1-i)\beta > 1 &+ \frac{i}{pk(k+1)} - p(k+1)\varepsilon > 1 + \frac{1}{pk(k+1)} - \frac{p(k+1)}{p^2k(k+1)^3} > 1.
\end{aligned}$$

For $i \leq k-1$, we get $x^1_{i+1} = p(k-i)$ and $x^2_i = p(k-i)+p-1 > p(k-i)$, thus the second claim follows. ■

For $i = 0$, clearly $x^1_0 = x^2_0 = p(k+1)$, since we have $p(k+1)\beta < 1$ and $(p(k+1)+1)\beta = 1 + \frac{1}{p(k+1)} - (p(k+1)+1)\varepsilon > 1 + \frac{1}{p(k+1)} - \frac{p(k+1)+1}{p^2k(k+1)^3} > 1$.

After the first part of the input, there are three cases. The input may stop, or be augmented by either $\frac{kN}{p}$ items of size α_1 or $\frac{kN}{p-1}$ items of size α_2. We find an optimal packing for each case, the cost of an optimal packing for case i is denoted OPT_i. Clearly, in the first case, each bin contains $p(k+1)$ items of size β. Thus $\text{OPT}_1 = \frac{N}{p(k+1)}$. In the second case, a bin can contain at most k items of size α_1. Indeed, we have $\text{OPT}_2 = \frac{N}{p}$, since each such bin can receive p smaller items. In the third case, a bin can contain at most k items of size α_2. Indeed, we have $\text{OPT}_3 = \frac{N}{p-1}$, since each such bin can receive $p-1$ smaller items.

Next, we consider the packing patterns of an online algorithm and we show the following.

Lemma 5 *We may assume without loss of generality that after the first part of the input, a bin contains one of the following three numbers of smaller items; $p(k+1)$, p, $p-1$. These are the values $x^1_0 = x^2_0$, x^1_k and x^2_k.*

Proof. If the algorithm opens some bin which contains q smaller items, where $q \neq x^j_i$ for all i, j, then let q' be the smallest number such that $q' > q$ and $q' = x^j_i$ for some pair i, j. Then replacing each bin with q items by $\frac{q}{q'}$ bins with q' items (recall that we allow a fractional number of packed bins) does not harm the packing of the second part of the input, and reduces the number of bins after the first part of the input.

Consider next a bin packed with x^j_i items, where $1 < i < k$. If $j = 1$, $x^j_i = p(k+1-i)$, and we replace this bin by $\frac{i}{k}$ bins with p items and $\frac{k-i}{k}$ bins with $p(k+1)$ items (the number of resulting bins may be fractional). The number of smaller items remains $\frac{p}{k}(i + (k-i)(k+1)) = p(k-i+1)$. If the second part of the input consists of items of size α_1, before the transformation, there was space for i items. After the transformation, there is space for k items in $\frac{i}{k}$ bins and no room in the other resulting bins. Thus there is no change.

If the second part of the input consists of items of size α_2, before the transformation, there was space for $i-1$ items. After the transformation, there is space for $k-1$ items in $\frac{i}{k}$ bins and no room in the other resulting bins. We have $\frac{i(k-1)}{k} = i - \frac{i}{k} > i-1$. Thus the transformation creates additional space.

If $j = 2$, $x^j_i = p(k+1-i)-1$. Let $\mu = \frac{p(k-i)}{pk+1}$. We have $0 < \mu < 1$, and $1-\mu = \frac{1+pi}{pk+1}$. We replace this bin by $1-\mu$ bins with $p-1$ items and μ bins with $p(k+1)$ items. The number of smaller items remains $\frac{(p-1)(1+pi)+p(k-i)p(k+1)}{pk+1} = \frac{p-1+p^2i-pi+p^2k^2-ikp^2+kp^2-ip^2}{pk+1} = \frac{(pk+1)(pk-ip+p-1)}{pk+1} = p(k+1-i)-1$. Before the transformation, there was space for i larger items, no matter which larger items arrive. After the transformation, there is space for k items in $1-\mu$ bins and no room in the other resulting bins. This gives space of $\frac{k+pik}{pk+1} > i$. Thus the transformation creates additional space. ■

Finally, we show the following lemma.

Lemma 6 *For $p = 2(k^2+1)$, the resulting lower bound is $Q(k)$.*

Proof. Let y_L, y_M and y_S be the (possibly fractional) number of bins opened by the algorithm after the first part of the input, with $p(k+1)$, p and $p-1$ items (respectively). We compute the cost of the packing for each case, where the cost of the packing for case i is denoted ALG_i. We have $p(k+1)y_L + py_M + (p-1)y_S = N$ and $\text{ALG}_1 = y_L + y_M + y_S$. If the items of the second part are of size α_1, then bins with p and $p-1$ items can receive k additional items, and bins with $p(k+1)$ items receive no further items. If the items of the second part are of size α_2 the situation is similar, only bins with p items can receive only $k-1$ further items. Since bins with $p(k+1)$ smaller items cannot receive further items, and every other bin (including new ones) can contain at most k larger items, we have $\text{ALG}_2 \geq y_L + \frac{N}{p}$. We next compute a lower bound on the number of required bins in the last case. To this end, we show that the existing bins cannot contain all larger items. This holds since $(k-1)y_M + ky_S < \frac{kp}{p-1}y_M + kY_S$ and $p(k+1)y_L + py_M + (p-1)y_S = N$ implies $\frac{kp}{p-1}y_M + ky_S = \frac{kN}{p-1}$. Therefore additional bins are opened, which implies, $\text{ALG}_3 \geq y_L + y_M + y_s + \frac{\frac{kN}{p-1} - (k-1)y_M - ky_S}{k} = y_L + \frac{y_M}{k} + \frac{N}{p-1}$.

We use $(p-1)\text{ALG}_1 + (pk+1-k)\text{ALG}_2 + k\text{ALG}_3 \leq \mathcal{R}((p-1)\text{OPT}_1 + (pk+1-k)\text{OPT}_2 + k\text{OPT}_3)$, which implies

$$\begin{aligned}(p-1)(y_L+y_M+y_S) + (pk+1-k)(y_L+\frac{N}{p}) + k(y_L + \frac{y_M}{k} + \frac{N}{p-1}) \\ \leq \mathcal{R}((p-1)\frac{N}{p(k+1)} + (pk+1-k)\frac{N}{p} + k\frac{N}{p-1}) \, .\end{aligned}$$

Which gives

$$\begin{aligned}y_L(p-1+pk+1-k+k) + y_M(p-1+1) + y_S(p-1) + N\frac{(p-1)(pk+1-k)+kp}{p(p-1)} \\ = y_L \cdot p(k+1) + p \cdot y_M + y_S \cdot (p-1) + N\frac{p^2k - pk + p - 1 + k}{p(p-1)} \\ \leq \mathcal{R} \cdot N \cdot \frac{((p-1)^2 + (pk+1-k)(p-1)(k+1) + kp(k+1))}{p(p-1)(k+1)} \, .\end{aligned}$$

Using $p(k+1)y_L + py_M + (p-1)y_S = N$ we have,

$$\begin{aligned}N + N\frac{p^2k - pk + p - 1 + k}{p(p-1)} = N\frac{p^2k - pk - 1 + k + p^2}{p(p-1)} \\ \leq \mathcal{R} \cdot N \cdot \frac{((p-1)^2 + (pk+1-k)(p-1)(k+1) + kp(k+1))}{p(p-1)(k+1)} \, .\end{aligned}$$

and finally,

$$\mathcal{R} \geq \frac{(k+1)(p^2k - pk - 1 + k + p^2)}{p^2 - p + p^2k^2 - pk^2 + k^2 + p^2k} \, .$$

It is possible to verify that the best choice for p is $p = 2(k^2+1)$, which implies the lower bound we wanted to prove. ■

6 Towards a complete solution

In this section we demonstrate that the approach above cannot lead to an optimal solution for all pairs of α and β, as a direct function of these two values, even if we compute the resulting competitive ratio of the algorithm we presented as a function of the exact sizes.

We consider a simple example where $\alpha = \frac{4}{10}$ and $\beta = \frac{3}{10}$. There are three possible dominant patterns for these values, which are $(2, 0)$, $(1, 2)$ and $(0, 3)$. Clearly, the algorithm we used for these values is Greedy. We can compute the competitive ratio using the weights according to w_1, which are $\frac{1}{2}$ and $\frac{1}{3}$, and get the weights $1, \frac{7}{6}, 1$ for these three patterns. This implies a competitive ratio of $\frac{7}{6}$. Note that Combine cannot be used here, since $t = 0$.

We next describe an algorithm called CombineBoth of competitive ratio $\frac{8}{7}$. This algorithm tries to pack some of the items using the pattern $(1, 2)$, into bins that are called "red bins". Thus, a fraction $\gamma = \frac{1}{7}$ of the larger items is colored red, and a fraction $\delta = \frac{1}{7}$ of the smaller items is colored red. The blue items of each size are packed using Greedy, into bins that are called "blue bins" (at each time, there is at most one active bin per size). A red larger item is packed into a bin with at least one smaller red item, if there exists such a bin that did not receive a larger item yet, and otherwise into a new red bin. A red smaller item is packed into a red bin which contains exactly one red smaller item, if such a bin exists, otherwise into a red bin containing only one item, which is a larger red item, and if such a bin does not exist either, we open a new red bin for the item.

Using this method, there is at most one bin that contains exactly one red smaller item. Moreover, there is at most one blue bin for each one of the two sizes that does not contain the full number of items (which is two, for larger items, and three, for smaller items). Each additional red bin, with less than three items, contains either a larger item, or two red items. Moreover, according to the algorithm, the output cannot contain both.

We define two weight functions, where the first function w_4 can be used for the scenario where all red bins (except possibly one) contain two red smaller items, and w_5 can be used for the case where all red bins contain a red larger item.

The functions are defined in Table 3.

	$w_4(x)$	$w_5(x)$
$x = \alpha$	$\frac{1-\gamma}{2} = \frac{3}{7}$	$\frac{1-\gamma}{2} + \gamma = \frac{4}{7}$
$x = \beta$	$\frac{1-\delta}{3} + \frac{\delta}{2} = \frac{5}{14}$	$\frac{1-\delta}{3} = \frac{2}{7}$

Table 3: The weight functions

For the analysis, we assume that the number of smaller items is divisible by 14 and that the number of larger items is divisible by 7. Otherwise, we neglect a constant number of items of each size. We compute the weight of every seven large items together and of every 14 smaller items together.

In the first scenario, seven larger items create three bins of blue larger items and one bin of red items. The required weight of red bins is covered by smaller items. Thus the total weight of these seven items is 3, which is exactly the number of blue bins these items create. In the second scenario, seven larger items create the same bins as before, but now they need to have enough weight for four bins. The total weight of these seven items is 4 as required.

In the first scenario, 14 smaller items create four blue bins of smaller items and one red bin. The total weight of these 14 items is indeed 5. In the second scenario, 14 smaller items need to only cover the weight of four blue bins, since larger items cover the cost of red bins. Their total weight is indeed 4.

We next compute the weight of each pattern according to the two weight functions. For the patterns $(2, 0)$, $(1, 2)$ and $(0, 3)$ we get totals of $\frac{6}{7}$, $\frac{8}{7}$ and $\frac{15}{14}$ according to w_4 and of $\frac{8}{7}$, $\frac{8}{7}$, and $\frac{6}{7}$ according to w_5. The maximum of all these values is $\frac{8}{7}$.

Another interesting property is that unlike the lower bound of Section 4, a matching lower bound is achieved here by using a sequence of larger items followed by smaller items and not smaller items followed by larger items. An attempt to apply the latter results in a lower bound of only $\frac{9}{8}$.

We briefly describe the lower bound of $\frac{8}{7}$. The input contains $2n$ larger items, possibly followed by $4n$ smaller items. We denote by OPT_i and ALG_i the costs of an optimal offline algorithm, and an online algorithm after i parts of the input ($i = 1, 2$). We have $\text{OPT}_1 = n$ and $\text{OPT}_2 = 2n$. Let y_1 and y_2 be the numbers of bins that contain one and two items respectively after the first part of the input. We have $y_1 + 2y_2 = 2n$. Note that $y_2 \leq n$ and $y_1 \leq 2n$.

For the algorithm, we have $\text{ALG}_1 = y_1 + y_2$. Each bin that is packed with two larger items cannot receive further items. Each bin that has one larger item can receive two smaller items. The number of smaller items that must be packed into new bins is thus $4n - 2y_1 \geq 0$. They can be packed at most three to a bin, and so we have $\text{ALG}_2 \geq y_1 + y_2 + \frac{4n-2y_1}{3} = \frac{y_1}{3} + y_2 + \frac{4n}{3}$. Let $\mathcal{R}$ be the competitive ratio of the algorithm. We get $y_1 + y_2 \leq \mathcal{R}n$ and $\frac{y_1}{3} + y_2 + \frac{4n}{3} \leq 2\mathcal{R}n$, or $y_1 + 3y_2 + 4n \leq 6\mathcal{R}n$. Taking the sum of these inequalities we have $2y_1 + 4y_2 + 4n \leq 7\mathcal{R}n$. We use $y_1 + 2y_2 = 2n$ to get $\mathcal{R} \geq \frac{8}{7}$.

We have proved the following.

Proposition 7 *For $\alpha = 0.4$ and $\beta = 0.3$, the best competitive ratio is $\frac{8}{7}$, and it is achieved using CombineBoth.*

7 Concluding remarks

In this paper we found the best overall competitive ratios for bin packing with two (known in advance) items sizes, where both sizes are at most $\frac{1}{k}$ for an integer k. A natural open problem would be to find the best competitive ratio as a function of the two sizes, either as a closed formula, or at least as a solution of a linear program. It seems that for any pair of sizes, it is likely that the usage of four configurations may be enough to achieve an algorithm of optimal competitive ratio. Moreover, it is likely that the lower bound for every pair would contain a stream of items of one size, followed by items of the second size. Proving or disproving these claims is left for future work.

We would like to point out that the best competitive ratio for a given pair of sizes α and β depends only on the finitely many dominant combinations of items which can be packed in a single bin. The set of all pairs with the same set of combinations forms an equivalence class. Thus, for example, the pair $(\frac{4}{10}, \frac{3}{10})$ in Section 6 is equivalent, e.g., to the pair $(\frac{3}{7}, \frac{2}{7})$. In particular, it is possible to assume that both sizes are rational, and that there exists at least one combination of these items which fills exactly one bin.

Acknowledgment. The authors thank an anonymous referee for suggesting the last paragraph of Section 7.

References

[1] B. S. Baker and E. G. Coffman, Jr. A tight asymptotic bound for Next-Fit-Decreasing bin-packing. *SIAM J. on Algebraic and Discrete Methods*, 2(2):147–152, 1981.

[2] D. J. Brown. A lower bound for on-line one-dimensional bin packing algorithms. Technical Report R-864, Coordinated Sci. Lab., Urbana, Illinois, 1979.

[3] B. Chandra. Does randomization help in online bin packing? *Information Processing Letters*, 43:15–19, 1992.

[4] J. Csirik. The parametric behaviour of the First Fit decreasing bin-packing algorithm. *Journal of Algorithms*, 15:1–28, 1993.

[5] U. Faigle, W. Kern, and G. Turán. On the performance of online algorithms for partition problems. *Acta Cybernetica*, 9:107–119, 1989.

[6] G. Gutin, T. Jensen, and A. Yeo. Batched bin packing. *Discrete Optimization*, 2(1):71–82, 2005.

[7] G. Gutin, T. Jensen, and A. Yeo. On-line bin packing with two item sizes. *Algorithmic Operations Research*, 1(2), 2006.

[8] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.

[9] 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*, 3:256–278, 1974.

[10] C. C. Lee and D. T. Lee. A simple online bin packing algorithm. *Journal of the ACM*, 32(3):562–572, 1985.

[11] F. M. Liang. A lower bound for online bin packing. *Information Processing Letters*, 10:76–79, 1980.

[12] S. T. McCormick, S. R. Smallwood, and F. C. R. Spieksma. A polynomial algorithm for multiprocessor scheduling with two job lengths. *Mathematics of Operations Research*, 26:31–49, 2001.

[13] P. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee. Online bin packing in linear time. *Journal of Algorithms*, 10:305–326, 1989.

[14] S. S. Seiden. On the online bin packing problem. *Journal of the ACM*, 49(5):640–671, 2002.

[15] J. D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University, Princeton, NJ, 1971.

[16] A. van Vliet. An improved lower bound for online bin packing algorithms. *Information Processing Letters*, 43(5):277–284, 1992.

[17] A. C. C. Yao. New algorithms for bin packing. *Journal of the ACM*, 27:207–227, 1980.