A New and Improved Algorithm for Online Bin Packing

János Balogh
Department of Applied Informatics, Gyula Juhász Faculty of Education, University of Szeged, Hungary
balogh@jgypk.u-szeged.hu

József Békési
Department of Applied Informatics, Gyula Juhász Faculty of Education, University of Szeged, Hungary
bekesi@jgypk.u-szeged.hu

György Dósa
Department of Mathematics, University of Pannonia, Veszprém, Hungary
dosagy@almos.vein.hu

Leah Epstein
Department of Mathematics, University of Haifa, Haifa, Israel
lea@math.haifa.ac.il

Asaf Levin
Faculty of Industrial Engineering and Management, The Technion, Haifa, Israel
levinas@ie.technion.ac.il

Abstract

We revisit the classic online bin packing problem studied in the half-century. In this problem, items of positive sizes no larger than 1 are presented one by one to be packed into subsets called *bins* of total sizes no larger than 1, such that every item is assigned to a bin before the next item is presented. We use online partitioning of items into classes based on sizes, as in previous work, but we also apply a new method where items of one class can be packed into more than two types of bins, where a bin type is defined according to the number of such items grouped together. Additionally, we allow the smallest class of items to be packed in multiple kinds of bins, and not only into their own bins. We combine this with the approach of packing of sufficiently big items according to their exact sizes. Finally, we simplify the analysis of such algorithms, allowing the analysis to be based on the most standard weight functions. This simplified analysis allows us to study the algorithm which we defined based on all these ideas. This leads us to the design and analysis of the first algorithm of asymptotic competitive ratio strictly below 1.58, specifically, we break this barrier by providing an algorithm AH (Advanced Harmonic) whose asymptotic competitive ratio does not exceed 1.57829.

Our main contribution is the introduction of the simple analysis based on weight function to analyze the state of the art online algorithms for the classic online bin packing problem. The previously used analytic tool named *weight system* was too complicated for the community in this area to adjust it for other problems and other algorithmic tools that are needed in order to improve the current best algorithms. We show that the weight system based analysis is not needed for the analysis of the current algorithms for the classic online bin packing problem. The importance of a simple analysis is demonstrated by analyzing several new features together with all existing techniques, and by proving a better competitive ratio than the previously best one.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Bin packing, online algorithms, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.5

© Janos Balogh, Jozsef Bekesi, Gyorgy Dosa, Leah Epstein, and Asaf Levin;
licensed under Creative Commons License CC-BY

1 Introduction

Bin packing [5, 6] is the problem of packing a set of items of rational sizes in $(0, 1]$ into subsets of items, which are called bins, of total sizes no larger than 1. In the offline variant the list of items is given as a set, and in the online environment items are presented one by one and each item has to be packed into a bin irrevocably before the next item is presented.

For an algorithm A, we denote its cost, that is, the number of used bins in its packing on an input I by $A(I)$. The cost of an optimal solution OPT, for the same input, is denoted by $OPT(I)$. The *asymptotic approximation ratio* allows to compare the costs for inputs for which the optimal cost is sufficiently large. The asymptotic approximation ratio of A is defined as follows. $R_A = \lim_{N\to\infty} \left(\sup_{I:OPT(I)\geq N} \frac{A(I)}{OPT(I)} \right)$. In this paper we only consider the asymptotic approximation ratio, which is the common measure for bin packing algorithms. Thus we use the term approximation ratio throughout the paper, with the meaning of asymptotic approximation ratio. Moreover, the term *competitive ratio* often replaces the term "approximation ratio" in cases where online algorithms are considered. We will use this term for the asymptotic measure. When we discuss the absolute measure $\sup_I \frac{A(I)}{OPT(I)}$ (the absolute approximation ratio or the absolute competitive ratio), we will mention this explicitly. A standard method for proving an upper bound for the asymptotic approximation ratio or the asymptotic competitive ratio for an algorithm A is to show the existence of a constant $C \geq 0$ independent of the input, such that for any input I, $A(I) \leq R \cdot OPT(I) + C$ (and then the value of the asymptotic measure is at most R). Most work on upper bounds on the asymptotic competitive ratio provide in fact an upper bound using this last method, and we will follow this approach as well.

For the offline problem, algorithms with an approximation ratio of $1 + \varepsilon$ can be designed [10, 17, 9, 13] for any $\varepsilon > 0$. If the first definition is used, a 1-approximation is known [17], where the cost of the solution computed by the algorithm is $OPT(I) + o(OPT(I))$ (see also recent work on improving the sub-linear function of $OPT(I)$ [21, 12]).

The classic bin packing problem, which we study here, was presented in the early 1970's [25, 14, 15, 16]. It was introduced as an offline problem, but many of the algorithms initially proposed for it were in fact online. Johnson [14, 15] defined and analyzed the simple algorithm Next Fit (NF), which tries to pack the next item into the last bin that was used for packing, if such a bin exists (in which case such a bin is called "active") and the item can be packed there, and otherwise it opens a new bin for the item. The competitive ratio of this algorithm is 2 [14, 15]. Any Fit (AF) algorithms, as opposed to the behavior of NF which only tests at most one active bin for feasibility of packing a new item there, pack a new item into a nonempty bin unless this is impossible (in which case a new bin is opened). Such algorithms have competitive ratios of at most 2. Next, consider a sub-class of algorithms where one may not select a bin with smallest total size of currently packed items for packing a new item, unless this minimum is not unique or this is the only bin that can accommodate the new item except for an empty bin. The last class of algorithms is called Almost Any Fit (AAF), and they have competitive ratios of 1.7 [16, 15]. A well-known algorithm, which is in fact a special case of AAF is Best Fit (BF), which always chooses the fullest bin where the new item can be packed. First Fit (FF) is another important special case of AF (but not of AAF) which selects a minimum index bin for each new item (where it can be packed). The competitive ratio of FF is 1.7 [16, 7].

The pre-sorted versions of these algorithms, called NFD, FFD, BFD, and AFD, were studied as well. Here items are still presented one by one, but they are sorted in a non

increasing order of sizes. For example, the approximation ratio of NFD is (approximately) 1.69103 [2] and that of FFD is $\frac{11}{9} \approx 1.22222$ [14]. For AFD in general, the approximation ratio is at most 1.25 [14, 15, 16]. These pre-sorted variants are not online algorithms.

We design and analyze a new algorithm AH (Advanced Harmonic) for online bin packing, and show that its competitive ratio does not exceed 1.57828956. This is the first algorithm whose asymptotic competitive ratio is below 1.58. We use a new type of analysis of algorithms which allows us to split the analysis into cases, while for every case we define only three values (and even just one value in a large number of cases), and based on those we calculate weights for items. The analysis is split into cases in recent previous work as well, but the analysis of each case is much more difficult. Items are partitioned into classes according to sizes. As in previous work, we sometimes do not pack the maximum number of items of some class into a bin, and leave space for items of another class (possibly arriving later). One new feature of AH is that in previous papers, in the algorithms there were at most two options for every class. For any given class, one option was a bin with the maximum number of items of this class fitting into a bin. For some of the classes there was a second option consisted of a very small number of items from this class (with reserved spaces for items of another class, possibly arriving later). We allow intermediate values as well with more than two options for some classes and not only two kinds of bins for a given class.

We use simple weight functions for the analysis, rather than the much more complicated tool called *weight systems* [23]. Weight functions are an auxiliary tool used for the analysis of bin packing (and other) algorithms (this technique is also called dual fitting). In this method, a weight is defined for each item (usually, based on its size, and sometimes it is also based on its role in the packing). If there are multiple kinds of outputs, it is possible to define a weight function for each one of them. The total weight of items is then used to compare the numbers of bins in the output of the algorithm and in an optimal solution. The list of weights of one item for different output types, also called scenarios, can be seen as a vector associated with the item. Thus, the weights can be seen as one function from the items to vectors whose dimension is the number of scenarios. Briefly, a weight system is a generalization where the weight function also maps items (or item sizes) to vectors, but in order to compute the weight of some item for a given scenario, another function, called a consolidation function, is used. This last function is a piecewise linear function (mapping real vectors to reals). The slightly simplified approach is to use convex combinations of weights according to subsets of scenarios. It has not been proved that weight systems are a stronger tool than just weights defined for the different scenarios. However, for simple weights every scenario can be analyzed independently from other scenarios. We exploit the simplicity of weight functions to obtain a clean and full analysis, which is easier to implement and verify (compared to the analysis resulting from weight systems). The main advantage is that every case is analyzed in a separate calculation using a standard knapsack solver without considering any other cases at that time. This simplicity allows us to analyze the new features that we introduce. Obviously, as these are cases for one algorithm, they have a common set of parameters, but once the algorithm has been fixed, there is no connection between the various cases.

The significance of our approach is that we combine many existing methods, including that of Babel et al. [1] (recently used by Heydrich and van Stee [22, 11] for classic bin packing), adding several new features, and applying a simple analysis, which can be verified easily and is robust to further changes of the algorithm. We define the action of our algorithm AH, we prove a number of invariants and properties of AH in detail, and then we provide the specific parameters and compact representations of the lists of weights. For every possible

output type and scenario, there is a small number of values used for the calculation of weights for it that we choose based on solving an auxiliary linear program. We also provide explicit lists of weights calculated based on the values and the parameters.

To explain the new features of our work, we discuss the harmonic type algorithms. Already in much of the previous work on online algorithms for bin packing, items were partitioned into classes by size. The simplest such classification is based on harmonic numbers, leading to the Harmonic algorithm of Lee and Lee [18]. In the harmonic algorithm of index k (for an integer parameter $k \geq 2$), subset j is the intersection of the input and $(\frac{1}{j+1}, \frac{1}{j}]$ (where $1 \leq j \leq k-1$), and subset k of tiny items is the intersection of the input and $(0, \frac{1}{k}]$.

In these algorithms each subset is packed independently from other subsets using NF (so for $j \leq k-1$, any bin for subset j, except for possibly the last such bin, has j items, but for subset k, every bin except for the last bin for this subset has a total size of items above $\frac{k-1}{k}$), and for k growing to infinity, the resulting competitive ratio is approximately 1.69103 [18]. The drawback of those algorithms is that bins of subsets with small values of j can be packed with small sizes of items (for example, a bin of subset 2 may have total size just above $\frac{2}{3}$ and a bin of subset 1 may have just one item of size just above $\frac{1}{2}$).

The first idea which comes to mind is to try to combine items of those two subset into common bins. However, if items of class 2 arrive first, one cannot just pack them one per bin, as this immediately leads to a competitive ratio of 2 (if no items of subset 1 arrive afterwards). Lee and Lee [18] proposed the following method to overcome this. A fixed fraction of items of subset 2 (up to rounding errors) is packed one per bin and the remaining items are packed in pairs. Thus, there are two kinds of bins for subset 2. The items we refer to here can only be sufficiently small items, so there is a threshold $\Delta \in (\frac{1}{2}, \frac{2}{3})$ such that items of sizes in $(\Delta, 1]$ and $(1-\Delta, \frac{1}{2}]$ are packed as before, while the algorithm tries to combine an item of size in $(\frac{1}{3}, 1-\Delta]$ with an item of size in $(\frac{1}{2}, \Delta]$. Even if those two items (one item of each one of the two intervals) are relatively small, still their total size is above $\frac{5}{6} \approx 0.83333$. This last algorithm was called Refined-Harmonic, and its competitive ratio is smaller than 1.636. Ramanan et al. [19] designed two algorithms called Modified Harmonic and Modified Harmonic-2. The first one has a competitive ratio below 1.61562, and it allows to combine items of many subsets with items of sizes above $\frac{1}{2}$ (and at most Δ). The second algorithm does not use only a single value of Δ, but splits the interval $(\frac{1}{2}, 1]$ further, allowing additional kinds of combinations. Its competitive ratio is approximately 1.612. For most subsets of items (where k is chosen to be in $[20, 40]$ in all these algorithms), the last two algorithms pack some proportion of the items in groups of smaller sizes, to allow it to be combined with an item of size above $\frac{1}{2}$. Intuitively, for an illustrative example, assume that $\Delta = 0.6$, and consider the items of sizes in $(\frac{1}{11}, \frac{1}{10}]$. The items that are not packed into groups of ten items should be packed into groups of four items (the parameters of the algorithms are different from those of this example). For some of the subsets the proportion is zero, and they are still packed using NF. The drawback of such algorithms (as it is exhibited by Ramanan et al. [19]) is that no matter how many thresholds there are, there can be pairs of items that can be combined into bins of optimal solutions while the algorithm does not allow it as it has fixed thresholds. Specifically, such algorithms allow to combine items of different intervals only in the case that the largest items of the two intervals fit together into a bin. This is the case with the next two harmonic type algorithms as well.

The next two papers, that of Richey [20] and that of Seiden [23] deal with a more complicated algorithm where many more subsets can be combined. The general structure is proposed in [20], and a full and corrected algorithm with its analysis is provided in [23]. For illustration, the items packed into smaller groups are called red and those packed into

bins with maximum numbers of items of the subset are called blue. The goal is to combine as many bins with blue items with bins having red items as possible. Bins with red items always have small numbers of items, to allow them to be combined with relatively large items of sizes above $\frac{1}{2}$. The analysis is far from being simple, though it leads to a competitive ratio of at most 1.58889 (Heydrich and van Stee [22, 11] mention that this last value can be decreased very slightly). The analysis of [23] is based on a complicated notion called weight system. The complicated details of this analytic tool basically did not allow the research community to introduce new algorithmic methods for dealing with the online bin packing problem. We expect that our simplified analysis will not suffer from this major disadvantage.

The carefully designed subset structure eliminates many worst-case examples, but the drawback mentioned above still remains. Recently, Heydrich and van Stee [22, 11] proposed to use a method introduced by Babel et. al [1], where some items are packed based on their exact size rather than by their subset. The approach of [22, 11] which we adopt is to apply the methods of Babel et. al [1] on the largest items, of sizes in $(\frac{1}{3}, 1]$. This approach means to combine items of sizes above $\frac{1}{2}$ with items of sizes in $(\frac{1}{3}, \frac{1}{2}]$ based on their exact sizes. Moreover, the approach involves combining pairs of items of subsets of sizes contained in $(\frac{1}{3}, \frac{1}{2}]$ while keeping the smallest items of such a subset to be matched with items of sizes above $\frac{1}{2}$ (and larger items of such a subset are used to be packed into pairs), as much as possible. Prior to the work of [22, 11], all previous algorithms for classic bin packing that partition items into classes always assumed that an item of a certain subset has the maximum size when its possible packing was examined. This method simplifies the algorithm and its analysis, but it is not always a good strategy as this excludes the option of combining items that can fit together into a bin in many cases. This approach is very different from that of AF algorithms and even from NF. Moreover, an approach similar to that of Babel et. al [1] was used in an online algorithm designed in [3]. Heydrich and van Stee [22, 11] claim a competitive ratio of 1.5816 but we were not able to verify this claim.

In algorithm AH, we do not just have red and blue items, but we potentially allow several kinds of bins (that is, several and potentially a large number of colors for items of a given class, and furthermore items may change their colors once further items arrive. Due to these differences we will not use the illustration via colors of items in the description of our algorithm). For example, for the subset of items of sizes in $(\frac{1}{15}, \frac{1}{14}]$ we group items into subsets of 14 items or three items or just one item. We also use bins of the smallest items (our value of k is 43) where the total size of items is at most $\frac{17}{60}$, to allow them to be combined (among others) with items of sizes in $(\frac{1}{2}, \frac{43}{60}]$. These two features are possible due to the simple nature of our analysis, and they are crucial for getting the improved bound. Note that all items of sizes in $(0, \frac{1}{43}]$ are treated together (by the algorithm and its analysis).

In order to use just a small number of values (one or three) for each scenario that we choose by solving an auxiliary linear program, we use the concept of *containers*. A container is a set of items of one class (in the partition of potential inputs into items of similar sizes, called classes), and it can be complete if its planned number of items has arrived already or incomplete otherwise (but it is treated in the same way in both cases). Containers are of two types, where a container is either positive or negative, and a bin may contain at most one of each of them. The goal is to have as many bins as possible with both a positive and a negative container. Roughly speaking, positive containers have total sizes above $\frac{1}{2}$ and negative containers have total sizes of at most $\frac{1}{2}$. This last statement is imprecise as in most cases we consider volumes and not exact sizes, where volumes are based on the maximum sizes for the corresponding classes. There is one exception which is containers with one item of size above $\frac{1}{3}$, where the exact size is taken into account (both by the algorithm and the

analysis), and it is defined to be the volume. A positive container and a negative one fit together if their total volumes does not exceed 1, and does not depend only on the classes. Our positive containers and negative containers have some relation to concepts used in [23].

In our weight based analysis, we assign weights to containers, where the number of different weights is small. Specifically, let the minimum volume of any positive container not packed with a negative container be denoted by a. We have two cases. In the simple case where all positive containers packed without negative containers have volumes of at least $\frac{2}{3}$ (i.e., $a \geq \frac{2}{3}$), we define weights as follows. Assign weights of 1 to positive containers packed without negative containers and negative containers packed without positive containers. Since we later base our weights of items on sizes, we assign these weights of 1 to all positive containers of volume at least a and all negative containers of volumes above $1-a$. We have a variable w ($0 \leq w \leq 1$) such that other positive containers have weights of w and other negative containers have weights of $1-w$. Those weights are called the required weights of containers (the actual weights can be larger but not smaller). Given the approximate proportions of items of each class packed in every type of container, we compute a weighted average (based on the containers of every item) to define weights of items using the required weights of containers. The case where $a < \frac{2}{3}$ is more interesting as a negative container with one item of size in $(\frac{1}{3}, \frac{1}{2}]$ and a positive container with one item of size above $\frac{1}{2}$ can be packed into one bin if the total size of the two items does not exceed 1 (i.e., the volumes of their containers are the exact sizes of these two items). Thus, the exact value a is crucial and not only its class, and additionally the class and even the exact value of $1-a$ play an important role. This is indeed more interesting as the analysis cannot be done for an infinite set of scenarios and thus we need to analyze intervals of a together. Here, for other classes we do the same as in the previous case, but for one class we perform a more careful analysis. This is the class containing the value $1-a$. For this class we define weights of items directly. We let the weight of an item of this class of size at most $1-a$ be a variable u, and otherwise it is a variable v, where $v \geq u$ (this class is contained in $(\frac{1}{3}, \frac{1}{2}]$). For the analysis, we found suitable values for the variables for all scenarios (this was done separately for each scenario), that is, for all possible values of a (the number of scenarios is still finite, as they are based on the dividing points of the algorithm, though not only on the classes). For every scenario where $a < \frac{2}{3}$, there are additional constraints on u, v, and w. As we do not use weights of containers in this case (for the class containing $1-a$), while the packing of pairs of items of classes contained in $(\frac{1}{3}, \frac{1}{2}]$ is performed carefully for all such classes. After selecting suitable values for those variables, all other item weights are also computed using the parameters of the algorithm.

There are also improved algorithms based on First Fit. Yao [27] designed a $\frac{5}{3}$-competitive algorithm where certain size based subsets are packed separately. Later, an algorithm of absolute competitive ratio $\frac{5}{3}$ was designed [3], which is the best possible with respect to this last measure [28] (see also [24, 7, 8]). The (asymptotic) competitive ratios should be compared to lower bounds on the competitive ratio. The current best such lower bound is 1.5403 [4] (see also [26]).

2 Algorithm AH

Notation and definitions. Similarly to previous algorithms' definitions, AH has a sequence of boundary points that are used in its precise definition: $1 = t_0 > t_1 = \frac{1}{2} > t_2 > \cdots > t_b = \frac{1}{3} > \cdots > t_M > t_{M+1} = 0$. That is 1/2 and 1/3 are always boundary points, and there is no boundary point in $(1/2, 1)$.

For every j, all items of sizes in the interval $(t_j, t_{j-1}]$ are called items of class j. We say that a class of items (and every item of this class) is *huge* if $j = 1$, it is *large* if $1 < j \leq b$ (these are all items of sizes above 1/3 and at most 1/2), *small* if $b < j \leq M$, and *tiny* if this is the class of items of size at most t_M (i.e., the last class which is the class of tiny items is class $M + 1$, and in general the index of a class corresponds to the index j such that t_j is the infimum size of any item of the class).

Our algorithm will pack items into containers and pack containers into bins. As the algorithm is online, a container will be packed into a bin immediately when it is created, even though it may receive additional items later. In the last case, when we say that an item is packed into a container, this means that the bin containing the container receives that item. Any container will contain items of a single class, and at most two different containers can be combined (packed) into a bin. We provide additional details on combining two containers into a bin later. Every container of items that are not tiny has a cardinality associated with it, and this is the (maximum) number of items that it is supposed to receive.

Let $\gamma_j = \lfloor \frac{1}{t_{j-1}} \rfloor$ for $j \leq M$. For class j that is either large or small (but not huge or tiny, i.e., for values of j such that $2 \leq j \leq M$ holds), and for every i (where $1 \leq i \leq \gamma_j$) there is a nonnegative parameter α_{ij}, where $0 \leq \alpha_{ij} \leq 1$. The value α_{ij} will denote the proportion of number of containers of cardinalities i of class j items among the number of containers of class j (the term proportion corresponds to the property of the sum of proportions satisfies $\sum_i \alpha_{ij} = 1$ for all j). Such containers that will eventually receive i items of class j (unless the input terminate before this becomes possible) will be called *type i containers of class j*. That is, intuitively if we let x denote the number of containers for items of class j, we will have approximately $\alpha_{ij} \cdot x$ type i containers each of which having exactly i items of class j. For every j such that $2 \leq j \leq M$ and every i, we let $A_{i,j} = i \cdot t_{j-1}$. While the values α_{ij} are defined so far only for large and small classes, we see one huge item as a type 1 container. Note that the values of α_{ij} are not proportions of items but of containers for class j, and the resulting proportions of items can be computed from them (we will prove such bounds accurately later).

For classes of *large* items the notion of the cardinality of a container is slightly more delicate, and we will have exactly four possible types of containers. The first type is a *regular type 2* container (already) containing exactly two items of this class. The second type is a *declared type 2 container*, where this type consists of containers for which the algorithm already decided to pack two items of this class in the container (so the planned cardinality of the container is 2) but so far only one such item was packed into the container (one of the few next arriving items of this class, if they exist, will be packed there, in which case the type will be changed into a regular type 2 container). The third is a *regular type 1* container, where such a container has one item of the class and cannot ever have (in future steps) an additional item of this class (such a container will be already combined with a container of another class that is packed into the same bin). The fourth and last type of a container of large items is a *temporary type 1 container*. A container of this last type currently has one item of the class but sometimes it will get an additional item of this class in future steps (and in this case its type will be changed at that time to regular type 2, its type can change to declared type 2 or regular type 1 as well, but in those cases it does not happen as a result of packing a new item to this container). Given a class of large items, the number of declared type 2 containers will be at most four throughout the execution of the algorithm (as we will prove below) while the numbers of containers of type 1 (of both kinds) and containers of regular type 2 can grow unbounded as the length of the input grows, though we will show certain properties on the relations between their numbers maintained by the algorithm.

The set of the union of containers of regular type 2 and declared type 2 are called type 2 containers, and the set of the union of containers of regular type 1 and temporary type 1 are called type 1 containers. The parameters α_{1j} and α_{2j} of a large class j determine the approximate proportions of type 1 containers and type 2 containers, respectively.

For class $M+1$ (of the tiny items), instead of the definitions above, there is a sequence of p possible upper bounds on the total sizes of items packed into containers of this class: $1 \geq A_{p,M+1} > A_{p-1,M+1} > \cdots > A_{1,M+1} \geq t_M$, and we let the positive parameters $\alpha_{i,M+1} > 0$ for $i = 1, \ldots, p$ denote the proportion of numbers of containers of class $M+1$ with items of total size in the interval $(A_{i,M+1} - t_M, A_{i,M+1}]$ (this is the planned total size of items for such a container). Such containers will be called *type i containers of class $M+1$*. The values of α_{ij} for all i, j are selected heuristically via a search procedure carried out by a computer program. Any such set of parameters give a different algorithm and our proof of the numerical value of the upper bound is for one specific set of parameters that we provide explicitly.

The *volume* of a container of type i of class j is defined as follows: If $i = 1$ and $1 \leq j \leq b$ (that is, for items of sizes above 1/3), the volume of the container is the size of its (unique) item, and otherwise ($i = 2$ and $2 \leq j \leq b$ or $i \geq 1$ and $j > b$) it is $A_{i,j}$. That is, the volume is usually simply the largest total size that the container can occupy, but for a container that contains a single large or huge item, the volume is the *exact* size of the item (there is one exception where the bin already contains one large item and it is planned to contain another item of the same class). In most cases we would like the volume of a container to be known when it is created, which is possible for containers such that their planned contents are known (in the sense that for example type i containers of a non tiny class j are planned to contain i items finally). However, for large items such containers with a single item may be temporary type 1 containers, in which case there is still no planning of contents for them. In this last case, the volume of the container is the size of its unique item. However, the volume of such a container may change in the case the algorithm will decide to pack another item of the same class (no matter if it packs that other item immediately at the time of decision or whether we decide to pack such an item later) into this container and transform it into a type 2 container. The volume of a declared type 2 container of class j is $A_{2,j} = 2 \cdot t_{j-1}$ (the volume is based on its complete contents, no matter whether they are present already or not, as it is the case for classes of small or tiny items).

We say that a container is *negative* if its volume is at most 1/2 and otherwise it is *positive*. Obviously, two positive containers cannot be packed into one bin. We will also not pack two or more negative containers into a bin together. Thus, a bin containing two containers will contain one positive container and one negative container, and no bin will contain more than two containers.

The rules for packing containers. The algorithm AH which we define next will pack items into containers and pack containers into bins according to rules we will define. Recall that the packing of containers into bins will be such that every bin will have at most one positive container and at most one negative container. Obviously, a bin is nonempty if it has at least one container and at most two containers. We say that a nonempty bin is negative if it has a negative container and does not have a positive container, it is positive if it has a positive container and does not have a negative container, and it is neutral if it has both a negative container and a positive container.

It is unknown whether a temporary type 1 container will eventually be positive or negative. Therefore, such a container will not be combined in a bin with another container as long as

its type is not changed. Moreover, it is seen as a negative container until it changes its type (so its bin is negative as long as the container is of temporary type 1). Specifically, it remains a negative container if a positive container joins it (and its bin becomes neutral), and in this case it becomes a regular type 1 container (and remains negative), and it becomes a positive container if its type changes to type 2. It can also happen that a temporary type 1 container will remain such till the termination of the input and the action of AH (and its bin remains negative). It is important to note that the difference between regular type 1 containers of a large class and temporary type 1 containers of the same class is that each of the former containers is already packed into a bin with a positive container (of some class), while the latter are not packed with other containers (in fact, the corresponding items are placed into their own bins, one item per bin).

For every class j, we denote by n_j the number of containers of class j. Let n_{ij} denote the number of containers of type i of class j. We also let N_j denote the number of items of class j at that moment. We often consider the values n_j and n_{ij} just prior to the packing of a new item, when N_j was already increased but the new item not packed yet so the values n_j and n_{ij} are not updated yet.

We say that two containers *fit together* if their total volume is at most 1. In what follows, when we refer to packing an item e - or more precisely, packing a container containing e (which was just created and therefore contains only e) into existing bins using Best Fit - we refer to packing e (or the container containing e) into the bin with a container of largest volume where the existing container and e (or the container containing e) fit together. For the original version of Best Fit, actual sizes are taken into account, but here we base this rule on volumes (as for a container with a single large or huge item the volume is equal to the size of the item, if we select one such container among a set of this last kind of containers, our action is equivalent to the standard application of Best Fit).

Packing rules of a new item. Next, we define the packing rules of the algorithm when a new item of class j arrives. The algorithm is defined for each step, based on the class of the new item.

A huge item. Recall that a huge item is immediately packed into a positive container containing only this item. Use Best Fit (applied on volumes, as explained above) to pack the created container into an existing bin, out of existing negative bins, such that the two containers (the new one with the huge item and the negative one of the negative bin) fit together. The only case where the new huge item joins a bin with a large item of some class j' is the case where the container of class j' is a temporary type 1 container, and in this case the type of this container of class j' is changed into regular type 1. If no bin can accommodate the container of the new item according to those packing rules, that is, for every negative bin, the total volume together with the new item is too big (or there is no negative bin at all), then use a new bin for the positive container of the new item (this new bin becomes a positive bin).

An item of a class of small or tiny items. For these classes we define the concept of an open container. Informally, an open container (of class j) can receive at least one additional item of class j. As a new container is introduced in order to pack an item, any container (of any type and class) already has at least one item of the corresponding class. If $b < j \leq M$, an open type i container of class j is one where the total number of the items in the container is strictly smaller than i. Once such a container receives i items, it is closed. For $j = M + 1$,

a type i container of this class will be open starting the time it is created and while the total size of items in it is positive and at most $A_{i,M+1} - t_M$. Once it reaches a total size above $A_{i,M+1} - t_M$, it will be closed. For all cases of packing a small or tiny item, a new container of some class will be used only if there is no open container of the same class, and thus, in particular, there will be at most one open container for each j (and the corresponding value of i will always be one such that $\alpha_{ij} > 0$).

When a new item of class j (such that $j > b$) arrives, if there is an open container of some type i of class j, then pack the item there (there can be at most one such container, so there are no ties in this case). Otherwise, open a new container for it (the details of the type are given below). After packing the new item into the container (and packing its container into a bin if it is a new container), close the container if necessary, based on its type and the rules above.

In the case that a new container is used for the item, we define the process of packing the item in more detail. Prior to packing the item, we define the type of the new open container. As the item is not packed yet, n_j is the number of containers of class j excluding the container opened for the new item. Find the minimum value of i such that $\alpha_{ij} > 0$ and so far there are at most $\lfloor \alpha_{ij} \cdot n_j \rfloor$ type i containers of class j (i.e., $n_{ij} \leq \lfloor \alpha_{ij} \cdot n_j \rfloor$, where the values n_{ij} do not include the new container which will be opened). Such an index i exists as otherwise there are more than n_j containers of class j. More precisely, since $\sum_i \alpha_{i,j} = 1$, there is always a value of i satisfying that $\alpha_{ij} > 0$ such that so far we opened at most $\lfloor \alpha_{i,j} \cdot n_j \rfloor$ type i containers of class j. Open a new type i container of class j containing the new item (increasing both n_j and n_{ij}). Observe that this opening of a new container defines its volume as well as whether it is a positive container or a negative container.

Next, we decide where to pack this new container. First consider the case where this container is a negative container. Then, if there is a positive bin, such that the new container fits into the bin according to its volume, then use that bin to pack the new container. This last case includes the possibility that the positive container is a type 2 container of a large class (regular or declared). If there are multiple options for choosing a bin, one of them is chosen arbitrarily.

Otherwise (there is no positive bin where the new negative container can be added), the algorithm checks the option of using a bin with a temporary type 1 container of some class of large items. Assume that there is a negative bin B such that the following two conditions are satisfied. The first condition is that the bin B has a temporary type 1 container of class j' such that a positive container of class j' (with two items) will fit together with the new (negative) container. The second condition is that there are at most $\lfloor \alpha_{2j'} \cdot n_{j'} \rfloor - 1$ type 2 containers of class j' (before the packing of the new item is performed). Then, pack the new negative container into B, and define the container of class j' packed into B as a declared type 2 container. This last container of class j' will get one of the next items of class j' that will arrive, which will happen before any new container is opened for any new class j' item, see below. If there are multiple options for choosing B, one of the classes of large items is chosen arbitrarily (among those that can be used), and a temporary type 1 container of this class with maximum volume is selected, i.e., we use Best Fit in this case. This last packing step is possible as a temporary type 1 container is never packed with another container into a bin (if another container joins it, its type is changed).

Otherwise (if there is no suitable positive bin and no class of large items has a suitable temporary type 1 container that can be used under the required conditions), pack the new negative container into a new bin.

Finally, consider the case where the new container is a positive container. Then, if there is a negative bin whose container is not a temporary type 1 container, such that the new container fits together with it, then use such a bin to pack the new container. Otherwise, if there is a temporary type 1 container with one large item of a class j' where the new container fits, then pack the new positive container into this bin and define the container of class j' in this bin as a regular type 1 container. The class j' can be chosen arbitrarily if there are multiple options, and among the temporary type 1 containers of class j', one of maximum volume (out of those that can be used) is selected, i.e., once again we use Best Fit. Otherwise, pack the new positive container into a new bin.

A large item of a class j. If there is a declared type 2 container of class j, pack the item there (as a second item) and change it into a regular type 2 container (breaking ties arbitrarily). This packing rule is checked first, and we apply it whenever possible. We continue to the other cases in the situation where there is no such declared type 2 container.

If the number of type 2 containers equals $\lfloor \alpha_{2j} \cdot n_j \rfloor$ (that is, we should not increase the number of type 2 containers at this stage), then pack the new item into a new negative container. To pack the container into a bin, do as follows. If there is a positive bin where the new negative container fits, then use Best Fit to pack it as a regular type 1 container of class j (its volume is defined accordingly as the size of the new item) together with a positive container (this positive container is not of large items, as three large items cannot be packed into a bin together). Otherwise the new container is packed into a new bin, in which case it is defined to be a temporary type 1 container.

Otherwise (that is, the number of type 2 containers is strictly smaller than $\lfloor \alpha_{2j} \cdot n_j \rfloor$), we will increase the number of regular type 2 containers or the number of declared type 2 containers of this class in the current iteration as follows. If there is a negative bin B where a type 2 container of class j fits, then pack the item into a new declared type 2 container of class j and pack this container into this bin B. Otherwise, if there is a temporary type 1 container of class j, then we pack the new item using Best Fit (considering only temporary type 1 containers of class j, and selecting such a container of largest volume) and change the type of this container into a regular type 2 container. Otherwise (all containers of class j are either regular type 1 or regular type 2, we should increase the number of type 2 containers, and a new container with two items of this class cannot be packed into an existing bin), we open a new declared type 2 container for the new item and open a new bin for this declared type 2 container (and pack it there).

A sketch of the analysis. In the analysis, we see a pair of a negative container and a positive container, packed together in a bin, as matched to each other, and each one of them is seen as matched (while every container packed into a bin without another container is unmatched). Let $a' = 1 - s_{\min}/2$ where $s_{\min}$ is the smallest item size in the examined input, and let a be the smallest volume of a positive container that is unmatched, if it exists. If no unmatched positive container exists, let $a = a'$. If $a > a'$, decrease the value of a to be a'. A simple property of the algorithm is that it tries to match a positive container and a negative container whenever possible. Thus every positive container of volume smaller than a is matched and every negative container of volume at least $1 - a$ is matched.

We define a *finite* set of scenarios according to the value of a. To do that we define a set of values V as follows. $V = \{A_{i,j}, 1 - A_{i,j} : j = 2, 3, \ldots, M+1, \alpha_{ij} > 0\} \cup \{t_1, t_2, \ldots, t_M, t_{M+1}\}$ and $V' = \{x \in V : x \leq 1/2\}$ (in particular, $\frac{1}{2} \in V'$). Note that the set V' contains (among other) all boundary points t_j (for all $j \geq 1$), even for values of j for which $\alpha_{1j} = 0$. The

name of a scenario is an interval $(x, y]$ between consecutive values in V'. Using this partition, we ensure that if the scenario is $(x, y]$, then there is no $i \geq 2$ and class j such that $\alpha_{ij} > 0$ and the volume of a container of type i of class j is in (x, y) or in $(1 - y, 1 - x)$.

The first step for analyzing each scenario is to obtain a good weight function for the scenario, in the sense that the analysis will be as tight as possible and can be done using a computer assisted proof within a small running time. The weight function defines size based weights for values in $(0, 1]$. The goal is to define weights such that the cost of the algorithm is roughly the total weight of all input items (a weight function satisfying this requirement is called here *valid*), and if the target competitive ratio is R, the cost of an optimal solution is at least the total weight divided by R (this can be proved by showing that no bin can contain items of total weight above R). Then, for an input I, letting $w(I)$ denote its total weight, (and as defined above, letting $OPT(I)$ the optimal cost for I, and $A(I)$ the number of bins used by A), we will have $A(I) \leq w(I) + c$, $OPT(I) \geq \frac{w(I)}{R}$, which shows that $A(I) \leq R \cdot OPT(I) + c$. This last argument is the standard argument for weight functions based analysis [14, 15, 16, 18, 19].

In order to define a suitable function, we will solve a linear program defined below (this linear program has only four variables w, u, v and R, and in some cases it actually has only two variables w and R). More precisely, we will provide a feasible solution for this linear program that is very close to the optimal one (but we only use its feasibility and do not prove that it is almost optimal). The weights of specific sizes will be based on the values w, u, v (or just on w, if the others are undefined), and on some of the parameters of the algorithm (the α_{ij} values for the given class).

We define a quantity for each container called the *required weight* of the container, and its goal is to introduce a uniform value such that weights of items are defined based on these values, in order to satisfy all requirements. This quantity is defined for a class that is not the threshold class or is not a large class. If the threshold class k (the class containing $1 - a$) is a large class, we keep this quantity undefined for that class. For a positive container of volume at least a, the required weight of the container is 1. For a positive container of volume in the interval $(1/2, a)$, the required weight of the container is denoted as w. This will be a decision variable of the forthcoming linear program. The required weight of a negative container is 1 if its volume is larger than $1 - a$ and otherwise its required weight is $1 - w$. We ensure that the required weight of a container depends only on the index of the scenario $(x, y]$ and not the specific value of a in the interval $[1 - y, 1 - x)$ and there are only few exceptions that are handled separately.

The weight of a huge item is 1 if its size is at least a and w otherwise. The weight of an item of class $j \leq M$ such that either $j \neq k$ or $j > b$ is the ratio between the average required weight of a container of class j and the average number of items in a container of class j. The weight of a tiny item of size s is s times the ratio between the average required weight of a container of tiny items and the average (lower bounds on the) total size of items in a container of tiny items. The weight of items of class $j = k$ that is a large class is as follows. An item of this class has weight u if its size is at most $1 - a$ and otherwise a weight of v. We find linear inequalities on the variables u, v, w that ensure that the resulting weight function is valid. By solving a linear program we can find such values of u, v, w that minimize the corresponding competitive ratio that can be proven using this weight function. In this linear program the goal is to minimize R that is an upper bound on the total weight of items that can fit into one bin subject to the additional constraints on u, v, w ensuring that the resulting weight function is indeed valid.

In this way we get a table showing for each scenario the set of the values of u, v, w (or only w for scenarios where the threshold class is not large) that define the weight function that we use for the scenario. Using these weight functions we show the correctness of our main result, namely that the competitive ratio of AH is at most 1.57828956.

References

1. L. Babel, B. Chen, H. Kellerer, and V. Kotov. Algorithms for on-line bin-packing problems with cardinality constraints. *Discrete Applied Mathematics*, 143(1-3):238–251, 2004.
2. 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.
3. J. Balogh, J. Békési, Gy. Dósa, J. Sgall, and R. van Stee. The optimal absolute ratio for online bin packing. In *Proc. of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA2015)*, pages 1425–1438, 2015.
4. J. Balogh, J. Békési, and G. Galambos. New lower bounds for certain bin packing algorithms. *Theoretical Computer Science*, 1:1–13, 2012.
5. E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation algorithms*. PWS Publishing Company, 1997.
6. J. Csirik and G. J. Woeginger. On-line packing and covering problems. In *A. Fiat and G. J. Woeginger, editors,* Online Algorithms: The State of the Art, pages 147–177, 1998.
7. Gy. Dósa and J. Sgall. First Fit bin packing: A tight analysis. In *Proc. of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS2013)*, pages 538–549, 2013.
8. Gy. Dósa and J. Sgall. Optimal analysis of Best Fit bin packing. In *The 41st International Colloquium on Automata, Languages and Programming (ICALP2014)*, pages 429–441, 2014.
9. L. Epstein and A. Levin. A robust APTAS for the classical bin packing problem. *Mathematical Programming*, 119(1):33–49, 2009.
10. W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.
11. S. Heydrich and R. van Stee. Beating the harmonic lower bound for online bin packing. In *Proc. of 43rd International Colloquium on Automata, Languages, and Programming (ICALP2016)*, pages 41:1–41:14, 2016.
12. R. Hoberg and T. Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA2017)*, pages 2616–2625, 2017.
13. K. Jansen and K.-M. Klein. A robust AFPTAS for online bin packing with polynomial migration. In *Proc. of the 40th International Colloquium on Automata, Languages, and Programming (ICALP2013), part I*, pages 589–600, 2013.
14. D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973.
15. D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.
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*, 3:256–278, 1974.
17. N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proc. of the 23rd Annual Symposium on Foundations of Computer Science (FOCS'82)*, pages 312–320, 1982.

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

19 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.

20 M. B. Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics*, 34(1–3):203–227, 1991.

21 T. Rothvoss. Better bin packing approximations via discrepancy theory. *SIAM Journal on Computing*, 45(3):930–946, 2016.

22 R. van Stee S. Heydrich. Beating the harmonic lower bound for online bin packing. *The Computing Res. Rep. (CoRR)*, abs/1707.01728, 2017. `arXiv:1511.00876v3`.

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

24 D. Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Research Logistics*, 41(4):579–585, 1994.

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

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

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

28 G. Zhang. Private communication.