Noname manuscript No.
(will be inserted by the editor)

On-line Bin Packing with Restricted Repacking

János Balogh · József Békési · Gábor Galambos · Gerhard Reinelt

the date of receipt and acceptance should be inserted later

Abstract Semi-online algorithms for the bin-packing problem allow, in contrast to pure on-line algorithms, the use of certain types of additional operations for each step. Examples include repacking, reordering or lookahead before packing the items. Here we define and analyze a semi-online algorithm where for each step at most k items can be repacked, for some positive integer k. We prove that the upper bound for the asymptotic competitive ratio of the algorithm is a decreasing function of k, which tends to 3/2 as k goes to infinity. We also establish lower bounds for this ratio and show that the gap between upper and lower bounds is relatively small.
Keywords: bin-packing, semi-online algorithm, worst-case behaviour, competitive analysis.

1 Introduction

Bin-packing is a well-known combinatorial optimization problem. In the classical one-dimensional problem a list $L = \{x_1, x_2, \ldots, x_n\}$ of n items (or elements), and an infinite set of unit capacity bins is given. Each item x has a $size(x)$ where $size(x) \in (0, 1]$. The problem consists of assigning each item to a unique bin such that the sum of the items in a bin does not exceed the bin capacity and such that the total number of bins used is as small as possible. The problem is known

This study was supported by the HSC-DAAD Hungarian-German Research Exchange Programme (Project P-MÖB/837) and Gyula Juhász Faculty of Education, University of Szeged (Project CS-001/2010).

J. Balogh · J. Békési · G. Galambos
Department of Applied Informatics, Gyula Juhász Faculty of Education,
University of Szeged, H-6701 Szeged, POB 396, Hungary,
E-mail: {balogh,bekesi,galambos}@jgypk.u-szeged.hu,

G. Reinelt
Institute of Computer Science, University of Heidelberg,
Im Neuenheimer Feld 368, D-69120 Heidelberg, Germany,
E-mail: gerhard.reinelt@informatik.uni-heidelberg.de

to be NP-hard [14]. Consequently, a large number of papers on polynomial time algorithms with acceptable worst-case behaviour have been published. A particular class is the so-called *on-line algorithms*, where the list L is scanned and each item has to be packed immediately without having either information on the sizes of the subsequent items or their number. In particular, an item assigned to a bin cannot be repacked or moved as the algorithms packs later items. In contrast, *off-line algorithms* have complete knowledge about the list of items and can thus apply further strategies for packing. As an intermediate case, one can define *semi-on-line algorithms* [8]. For example these algorithms allow one of the following operations for each step or they have additional information about the input:

- repacking some finite number of already packed items [4,12,13,18,19],
- lookahead in the list before the current item is packed (i.e., some items are known before they are packed) [15,16]),
- preprocessing (e.g., order w.r.t. sizes) some items [11], or
- knowledge of the optimum value before packing the list [1,9].

Here, and throughout the paper a step means the arrival and packing of the successive item of the input list.

There are various methods available to measure the efficiency of bin-packing algorithms. Here we restrict ourselves to the asymptotic competitive ratio. If $A(L)$ denotes the number of bins used by algorithm A and $OPT(L)$ stands for the number of bins used in an optimal packing, then the *asymptotic competitive ratio* (ACR) of A is

$$R(A) := \limsup_{k\to\infty} \left\{ \max_L \left\{ \frac{A(L)}{k} \;\middle|\; OPT(L) = k \right\} \right\}.$$

For off-line algorithms the first APTAS (Asymptotic Polynomial Time Approximation Scheme) was developed in [10], where the authors demonstrate that for any given $\varepsilon > 0$, there is an algorithm which solves the problem in linear time in the length of the input list L (but exponential in $1/\varepsilon$) with ACR $1 + \varepsilon$.

Some on-line algorithms keep all bins open into which items have been placed, while others allow only a restricted finite number of open bins for each step. These are so-called *bounded-space algorithms*. Among bounded-space algorithms, those of the *harmonic fit type* play an important role. The first such algorithm was formulated by C.C. Lee and D.T. Lee in [20]. Their *HARMONIC(M)* algorithm is based on a special nonuniform partition of the interval $(0, 1]$ into i subintervals. A single active bin is assigned to each of these subintervals and only items belonging to this subinterval are packed into this corresponding bin. If some item do not fit into an assigned bin, this bin is closed and a new bin is opened. (We say that a bin is *closed* if we cannot pack items into it any more in the course of applying the algorithm.) They proved that $\lim_{M\to\infty} R(\mathit{HARMONIC(M)}) = T_\infty := \sum_{i=1}^{\infty} \frac{1}{k_i} = 1.69103\ldots$, where $k_1 = 1, k_{i+1} = k_i(k_i + 1), i \geq 1$ is the Salzer series, which is well–known in bin-packing [24]. On the negative side, Lee and Lee proved in [20] that no on-line bounded-space bin-packing algorithm can have an ACR less than the constant T_∞.

The on-line algorithm with the best ACR so far was developed by Seiden and has an ACR of 1.58889 [25]. This algorithm, called *Harmonic++*, belongs to the class of *Super Harmonic Algorithms*, defined in [25]. Note that the earlier champion on-line algorithms (from 1985) [20,22,23] belong to this class as well. The best known lower bound of 1.5403 for the ACR of on-line algorithms was presented in [2], while the lower bound of any Super Harmonic type algorithm is 1.58333 [22]. It means that to beat the 1.58333 bound by an on-line algorithm, one needs to change the technique used in the set-up of the current champion on-line algorithms for the last 20 years. Another possibility is to allow for an algorithm slightly more than the on-line rule (e.g. change to semi-on-line cases).

Chronologically, the first semi-on-line algorithm was given by Galambos [11] for the *bounded-space* version of the classical bin-packing problem where only a fixed number of bins are open while packing. This algorithm uses two "buffer-bins" for the temporary storing of items. The idea was further developed by Galambos and Woeginger [12] for an algorithm that uses three buffer-bins and has an ACR of T_∞. In [12], it was proved that there is no bounded-space on-line bin-packing algorithm that uses repacking and has an ACR better than T_∞.

Semi-on-line algorithms with repacking were studied by Gambosi et al. [13]. They gave two algorithms: one with an ACR of $\frac{3}{2}$ and time complexity $O(n)$ and a second one with ACR of $\frac{4}{3}$ and running time of $O(n \log n)$. The latter algorithm was improved by Ivkovič and Lloyd in [19] for an algorithm with an ACR of $\frac{5}{4}$. This algorithm also works for the *fully dynamic bin-packing* case. In dynamic bin-packing problems [6,7,17], items can also depart and the goal is to minimize the maximum number of bins used at any time. In [19] the arrival of the elements is considered as an Insert operation, while their departure is a Delete operation on the input. Fully dynamic means that items may be moved (repacked) among the bins as the packing is adjusted to accommodate arriving and departing items. Ivkovič and Lloyd's algorithm requires $\Theta(\log n)$ time per operations (i.e., for Insert and Delete of an item). Bundle of items is treated there as a whole, the number of items (either individual items or bundles of them) that needs to be repacked is bounded by a constant. It means that this semi-on-line algorithm does not repack a constant number of items in the strict sense, because repacking of a bundle of small items counts as one repacking. The models in [13,19] are different from the model we are studying in this paper, because we allow to repack a constant number of items per Insert operation in the strict sense (and of course, there is no Delete operation on the input in our model).

In this article we deal with semi-on-line algorithms that allow repacking. We will assume that repacking one item has a unit cost, independently of the size of the item. This model is completely different than the ones mentioned in the previous paragraph, because we assume that repacking of each element has the same cost. In addition, we will assume that the maximum number of items to be repacked for each step is bounded by a given constant k. We call such algorithms *k-repacking semi-on-line algorithms*. The first lower bound for this class was given by Ivkovič and Lloyd [18]. Their lower bound of $\frac{4}{3}$ is valid for all k and also valid for the fully dynamic bin packing problem. This general bound was improved in [3] to $1.3871\ldots$ which is also valid for both cases: for the fully dynamic bin packing problem and for any k-repacking semi-on-line bin packing problem for arbitrary k.

In this paper we improve the upper bounds given in the Hungarian language paper of the authors [4], which gives a similar but less sophisticated algorithm that in the present paper. The paper is organized as follows. In Section 2 we define our algorithm *HR-k* and analyze its time complexity, showing that the running time of the algorithm is $O(kn)$. Then in the next section we describe its asymptotic behaviour, demonstrating that the competitive ratio is slightly more than 3/2, where the exact value depends on k, establishing a monotone decreasing series in k. In the last section we briefly discuss the results obtained, then we suggest some possible directions for future study.

2 The algorithm and its time complexity

We will begin by making some formal definitions and defining our notations used here. First, let x denote an arbitrary item of the input list. Then we will refer to its size just by x, if the context is clear. For an open bin B, we denote by $level(B)$ the sum of the sizes of items presently packed into B. Clearly, $0 < level(B) \leq 1$. We call an item *small* if its size is in the interval $(0, \frac{1}{2}]$; otherwise the item is *large*. A bin B is a *large bin* if it contains a large item.

For each $k \in N^+$ denote by b_k the unique such solution of the equation $2kx^2 - (6k+3)x + 1 = 0$, which is in the interval $(0, \frac{1}{6k})$. (The precise way of getting the b_k values will be shown later.) We will write b instead of b_k if k is fixed and it is clear from the context.

Let k be an arbitrary nonnegative integer. The interval $(0, 1]$ is divided into $2k+3$ disjoint subintervals $I_j, j = 1, \ldots, 2k+3$, in the following way

$$
\begin{aligned}
(0, b_k] &=: I_1, \\
\left(b_k, \frac{1}{2} - kb_k\right] &=: I_2, \\
\left(\frac{1}{2} - jb_k, \frac{1}{2} - (j-1)b_k\right] &=: I_{k+3-j}, j = 1, \ldots, k, \\
\left(\frac{1}{2} + (j-1)b_k, \frac{1}{2} + jb_k\right] &=: I_{k+2+j}, j = 1, \ldots, k, \\
\left(\frac{1}{2} + kb_k, 1\right] &=: I_{2k+3}.
\end{aligned}
$$

It is easy to see that

$$
\begin{aligned}
\bigcup_{j=1,\ldots,k} I_{k+3-j} &= \bigcup_{j=1,\ldots,k} \left(\frac{1}{2} - jb_k, \frac{1}{2} - (j-1)b_k\right] = \left(\frac{1}{2} - kb_k, \frac{1}{2}\right], \\
\bigcup_{j=1,\ldots,k} I_{k+2+j} &= \bigcup_{j=1,\ldots,k} \left(\frac{1}{2} + (j-1)b_k, \frac{1}{2} + jb_k\right] = \left(\frac{1}{2}, \frac{1}{2} + kb_k\right].
\end{aligned}
$$

With each interval I_j we associate a class of bins denoted by $\mathcal{B}_j, j = 1, \ldots, 2k+3$. A bin will be assigned to class $\mathcal{B}_j$ if the size of the smallest item which has been put into it is in the interval I_j. Note that the classes of the bins may change during the packing, the exact rules will be given later.

We call two classes $\mathcal{B}_j$, $1 \leq j \leq k+2$, and $\mathcal{B}_l$, $k+3 \leq l \leq 2k+3$, *complementary classes* if $x_j + x_l \leq 1$ for all $x_j \in I_j$ and $x_l \in I_l$. The set of the complementary classes w.r.t. a class $\mathcal{B}_i$ will be denoted by $C(\mathcal{B}_i)$, $1 \leq i \leq 2k+3$. Note that $C(\mathcal{B}_{k+2})$ and $C(\mathcal{B}_{2k+3})$ are empty. Clearly, if l^* denotes the largest index of the classes in

$C(\mathcal{B}_j)$, $1 \leq j \leq k+1$, then $l^* = \min\{2k+2, 2k+4-j\}$. Similarly, if $\mathcal{B}_j \in C(\mathcal{B}_l)$ for $k+3 \leq l \leq 2k+2$, then $1 \leq j \leq 2k+4-l$.

Basically, our algorithm *HR-k (Harmonic Repacking)* will pack the items using the so-called *harmonic fit rule* (HF rule) [20]. Let x be the current item to be packed and suppose that $x \in I_j$. If B is the last opened bin in class $\mathcal{B}_j$ and $level(B)+x \leq 1$, then we put x into B. Otherwise, we open a new bin of class $\mathcal{B}_j$ and put x into this bin. When applying this HF rule, it is easy to see that in each class $\mathcal{B}_i$, $i \leq k+2$ or $i = 2k+3$, all bins except for possibly the last one being filled up at least to level $\frac{1}{2}+kb_k$. (In the case of $i = 2k+3$ it is trivial. In the remaining cases, if $i > 2$, then each bin of a class — except for possibly the last opened one — contains at least two items, and their total size is at least $1 - 2kb_k$, which with the choice of b_k is at least $\frac{1}{2} + kb_k$. In the case of $i \leq 2$ it can be readily seen that any bin of these classes — except for possibly one — has been filled up to a level $\frac{1}{2} + kb_k$.) So, if the list does not contain items belonging to other intervals, then one can see with little effort (observing that kb_k never exceeds $1/6$ and approaches this value for large k) that an ACR of value $\frac{3}{2}$ can be attained if k goes to infinity. In the rest of the paper we will mostly focus on the other case (when the list contains item(s) from the interval $I_{k+3} \cup \ldots \cup I_{2k+2}$), enhancing the HF rule in order to handle this case.

Evidently, the HF rule only places one single item into bins of classes $\mathcal{B}_{k+3}, \ldots, \mathcal{B}_{2k+2}$. Our algorithm will improve this by filling up these bins with possibly other items, if their sizes allow it. Furthermore, repacking is possible, i.e., for each step the bin assignment can vary for a limited number of items. Consequently, in our algorithm the number of currently used bins can change dynamically since we may open new bins and bins may become empty. To compute $A(L)$, we consider only those bins which are nonempty after the algorithm has packed all of the items of the list.

Our algorithm also differs from the "classical" harmonic fit algorithm in the treatment of bins within the classes: we never close them, so their content will be accessible during the whole execution of the algorithm (additional elements can be packed into the bin or repacked from the bin into another one). Thus, it is not a bounded-space algorithm.

We enhance the standard harmonic-fit rule based on the following two refinements.

- **Rule 1 (Refill)**: If x is the current item to be packed and $x \in I_j$ is a small item with $1 \leq j \leq k+1$, then we examine the complementary classes of $\mathcal{B}_j$ in increasing order of their indices. If we find a nonempty class $\mathcal{B}_l$, then we put the item into the last opened bin within the class. Let this bin be B. It is clear that $level(B) \in I_l$ before we pack x into B. If $level(B)+x \notin I_l$ then we reassign B to the class $\mathcal{B}_t$ where $level(B)+x \in I_t$.
  Clearly, if $x \notin I_1$, then after having packed it into a bin B this bin will always change its class.
  If all complementary classes are empty then we use the HF rule to pack this small item, but we do not close the bins opened previously in its class.
- **Rule 2 (Repack)**: If x is the current item to be packed and $x \in I_l$ is a large item with $k+3 \leq l \leq 2k+2$, then we open a new bin in class $\mathcal{B}_l$ and put x into this bin. Furthermore, we look for nonempty bins in the complementary class $\mathcal{B}_j \in C(\mathcal{B}_l)$ in decreasing order of their indices. If there exists such a bin, then we repack one item from this bin into the last opened bin in the class $\mathcal{B}_l$.

If the level of the large bin changes its interval (which will always happen if $j \neq 1$), then we reassign it to the respective class.
(We need to take into account the fact that bins may become empty if their last small item is repacked.)

It is worth noting that as a final step while packing an item (after applying both rules), we try to carry out a further possible repacking if the bin in class $\mathcal{B}_l$, $k+3 \leq l \leq 2k+2$, has changed its class, and for the new $\mathcal{B}_t$, $t \neq 2k+3$. This results in the call of Rule 2 (Repack) from both rules and it means that Repack is a recursive procedure (because after the first call of Repack, it can be called recursively by itself). In both cases it must be ensured that for each step at most k number of items are repacked.

Algorithm *HR-k*

main program;
  while there exist unpacked items **do**
    input next x;
    if $x \in I_l$, $k+2 \leq l \leq 2k+3$, **then**
      $x_{HF} \to \mathcal{B}_l$;
      if $x \notin I_{k+2}$ and $x \notin I_{2k+3}$ **then**
        call Repack(l, $2k+3-l$);
    else if $x \in I_j$, $1 \leq j \leq k+1$, **then**
      call Refill(x);

procedure Refill(x);
  $j :=$ the index of the class for which $x \in I_j$;
  $l^* := \min\{2k+4-j, 2k+2\}$; (the largest index of the classes of complements of $\mathcal{B}_j$)
  do $l = k+3$ **to** l^*
    if $\mathcal{B}_l \neq \emptyset$ **then**
      $x \to \mathcal{B}_l$; Let B be the bin into which x is packed.
      if $level(B) \in I_p$, $p \neq l$, **then**
        $B \to \mathcal{B}_p$;
        if $p < 2k+3$ **then**
          call Repack(p, $2k+4-p$);
      return; (either no change between classes or we have returned from Repack)
  $x_{HF} \to$ to the last opened bin of the class $\mathcal{B}_j$; (there was no complementary class in the do-loop)

procedure Repack(l, j);
  (We try to repack a small item from a bin of the class $C(\mathcal{B}_l)$ into the last bin of $\mathcal{B}_l$, $l \geq k+3$. Here j is the first index of the classes in $C(\mathcal{B}_l)$ where we need to start the search.)
  do $t = j$ **to** 1
    if $\mathcal{B}_t \neq \emptyset$ **then**
      $x \to \mathcal{B}_l$, where $x \in B, B \in \mathcal{B}_t$,; ($B$ is the last opened bin of $\mathcal{B}_t$ and x is its topmost item);
      if $level(B) \in I_s$, $l+1 \leq s \leq 2k+3$, **then**
        $B \to \mathcal{B}_s$;
        if $s < 2k+3$ **then**
          call Repack(s, t);

It should be mentioned that the above description of the two rules (and the third rule which is the recursive call of Repack) is only a brief description of the algorithm. The precise description of the algorithm (parameter values passed to procedures, etc.) is given below in the pseudocode listing. In the description we use the notations $x_{HF} \to \mathcal{B}_t$ and $x \to \mathcal{B}_t$ if we pack the actual item x using the HF rule, or if we simply pack x into the last opened bin of the class $\mathcal{B}_t$, respectively. Similarly, we will use the notation $B \to \mathcal{B}_t$ to mean the rearrangement of a bin from its previous class to the class $\mathcal{B}_t$.

The main program and the procedure Refill are self-explanatory. The parameter of Refill is the (size of the) item which we attempt to fit into a large bin. For Repack, the most important thing is that this procedure can call itself recursively. Its first parameter l is the index of the class of the large item that we try to match from a complementary class. In the first call, the second parameter is always $2k + 4 - l$, but in a series of recursive calls it is monotonically decreasing.

The next three statements are helpful for analyzing the time complexity of the algorithm.

Lemma 1 *Algorithm HR-k terminates in a finite time.*

Proof The main loop will be executed exactly n times, namely once for each item in L. The Refill procedure will be called at most once for each item of the list.

Although Repack may call itself recursively, this can only happen if the currently used bin changes its class and its level is at most $\frac{1}{2} + kb_k$. It follows that in the sequence of possible recursive calls the first parameters of Repack form a strictly monotonically increasing sequence: in the first call we have $l \geq k + 3$ and for the last one $l \leq 2k + 2$. Therefore the number of recursive calls is at most $k - 1$. From this, we may conclude that the algorithm calls Repack at most k times for each item of the list.

Since Repack is called either from the main loop or at most once for every each of Refill, the statement follows. $\square$

Having demonstrated the truth of this lemma, the next statement follows immediately.

Corollary 1 *HR-k repacks at most k items for each step.*

Lemma 2 *The number of inspections of the contents of nonempty classes $\mathcal{B}_i$ is at most $(2k + 2)n$.*

Proof We will show that for each step the algorithm cannot test the emptiness of more than $2k + 2$ bin classes.

In the main program there is no such test, and in Refill there are at most k comparisons for each item.

The procedure Repack has two parameters. The first one is l, where $l = k+2+i$, for some $1 \leq i \leq k$, and the second one is at most $2k + 4 - l = k + 2 - i$. The values of the second parameters of the recursive calls are monotonically, but not strictly decreasing. (The second parameter j is an index of a bin-class of small items, from which we attempt to repack a small item for each call of Repack. If this bin-class is empty, then we continue the examination of the bin-classes of small items in decreasing index-order.) Since for each item of a given list the procedure Repack will be called at most $k + 1 - i$ times, the algorithm performs at most $(k + 2 - i) + (k + 1 - i) = 2k + 3 - 2i$ comparisons.

While packing the current item, if Repack is called for the first time from Refill, it makes at most $i-1$ additional comparisons in that procedure. Otherwise, if the first call of Repack is executed directly from the main loop, then there is no additional comparison. □

Combining these three statements yields

Theorem 1 *The time complexity of HR-k is $O(kn)$.*

3 The asymptotic competitive ratio of *HR-k*

After having packed all of the items of a given list L, we see there are two distinct cases:

Case A: All of the classes $\mathcal{B}_{k+3}, \dots, \mathcal{B}_{2k+2}$ are empty.
Case B: There is at least one nonempty class among $\mathcal{B}_{k+3}, \dots, \mathcal{B}_{2k+2}$.

Lemma 3 *If all items of a given list L have been packed by HR-k and Case A holds, then*

$$HR\text{-}k(L) \le \frac{2}{1+2kb_k}\ OPT(L) + (k+2).$$

Proof In this case all opened bins have been packed at least to level $\frac{1}{2}+kb_k$, except for possibly the last opened bins of the classes $\mathcal{B}_1, \dots, \mathcal{B}_{k+2}$. The number of such bins is at most $k+2$. Therefore,

$$OPT(L) \ge \left(\frac{1}{2}+kb_k\right) HR\text{-}k(L) - (k+2)$$

and the statement follows. □

For Case B we will apply the standard weight function technique. (In Lemma 3 we analyzed Case A without dealing explicitly with a weight function. However, a weight function would obviously be defined in Case A as well, assigning a weight $w(x) = \frac{1}{\frac{1}{2}+kb_k}x$ to each item $x(\in (0,1])$. Then the weight of an item would depend on whether Case A or Case B holds after the algorithm has finished the packing of the input list.) Before we define the weight function for Case B, we need to state a lemma.

Lemma 4 *Suppose all items of a given list L have been packed by HR-k and at least one of the classes $\mathcal{B}_{k+3}, \dots, \mathcal{B}_{2k+2}$ is nonempty (i.e., Case B holds). Let $k+2+i^*$ be the smallest index of such a nonempty class. Then all complementary classes of $\mathcal{B}_{k+2+i^*}$, $1 \le i^* \le k$, are empty except for possibly $\mathcal{B}_1$.*

Proof Since $\mathcal{B}_{k+2+i^*}$ is not empty, it contains at least one nonempty bin B_1 and B_1 is a large bin.

Suppose on the contrary that there exists a nonempty complementary class of $\mathcal{B}_{k+2+i^*}$ and it is not $\mathcal{B}_1$. Let j_0 be the index of the largest nonempty class and let $B_2 \in \mathcal{B}_{j_0}$ be the last opened bin in this class. Then B_2 contains at least one small item. Let x be the top item in B_2. There may be two different scenarios. In the first case, the arrival of $x \in L$ precedes the packing (or repacking) of y, where item y has been packed as the last item into B_1. Since x preceded the packing (or repacking) of y, *HR-k* will pack x (or another small item) into B_1. Otherwise, if the arrival of x was later than the packing (or repacking) of item y, then the algorithm has packed x into B_1 (or into another large bin) after it had placed y into B_1. Both cases lead to a contradiction. □

Now we will define our weight function $w(x)$. Let i^* be defined as in Lemma 4. In the following we will just write b instead of b_k if k is fixed (and clear from the context).
Then

$$w(x) := \begin{cases} \frac{x}{1-b}, & x \in \{I_1 \cup I_2 \cup \ldots \cup I_{k+2-i^*}\}, \\ \frac{1}{2}, & x \in \{I_{k+3-i^*} \cup \ldots \cup I_{k+2}\}, \\ 1 - \frac{1}{1-b}(\frac{1}{2} + (i^*-1)b - x), & x \in \{I_{k+3} \cup \ldots \cup I_{k+1+i^*}\} (= \emptyset,\ \textit{if } i^* = 1), \\ 1, & x \in \{I_{k+2+i^*} \cup \ldots \cup I_{2k+3}\}. \end{cases}$$

Note that in the case of $i^* = 1$ the weight of any large item is 1. If some items have arrived from the intervals $I_2, I_3, \ldots, I_{k+2-i^*}$, then they must have been packed in a large bin, because the $\mathcal{B}_2, \mathcal{B}_3, \ldots, \mathcal{B}_{k+2-i^*}$ bin classes are empty (see Lemma 4).

In the case $i^* > 1$ the classes $\mathcal{B}_{k+2+j}$, $j = 1 \ldots, i^* - 1$ are empty at the end. That is, if some items have arrived from these intervals after finishing the packing they must be in bins of classes for which the index of the class is at least $k+2+i^*$. For such a large item we assign a weight w, which is less than 1. To guarantee that the weight of such a bin is at least 1, for the small items of this bin we assign a weight in such a way that the total weight of the small elements contained by the bin is at least $1 - w$. It ensures that the weight of these bins is at least 1 as well. In the proof of the next lemma we will show this, and that in Case B the weight of each bin is at least 1, except for at most one bin of the class $\mathcal{B}_1$ and one bin of each of the classes $\mathcal{B}_{k+3-i^*}, \ldots, \mathcal{B}_{k+2}$.

Lemma 5 *If Case B holds and $k + 2 + i^*$ is the smallest index for which $\mathcal{B}_{k+2+i^*}$ contains at least one nonempty bin, then for any list L*

$$w(L) = \sum_{x \in L} w(x) \geq \textit{HR-k}(L) - (k + 1).$$

Proof We will show that for each class, except for at most one bin in each class, the sum of the weights of the items in a bin is at least 1.

- If $B \in \mathcal{B}_1$ and it is not the last opened non-empty bin, then $\textit{level}(B) > 1 - b$. If $x \in B$ then $w(x) = \frac{1}{1-b}x$. Therefore $w(B) = \sum_{x \in B} w(x) > 1$.
- The class $\mathcal{B}_i$ is empty, where $2 \leq i \leq k + 2 - i^*$ (see Lemma 4).
- If $B \in \mathcal{B}_j$, $k + 3 - i^* \leq j \leq k + 2$, and it is not the last opened non-empty bin in its class, then it contains exactly two items x and y with $w(x) = w(y) = \frac{1}{2}$.
- If $B \in \{\mathcal{B}_{k+2+i^*}, \ldots, \mathcal{B}_{2k+3}\}$, then $\textit{level}(B) > \frac{1}{2} + (i^* - 1)b$ and the bin contains one large item x. Here, there are two distinct cases.
  - If $x > \frac{1}{2} + (i^* - 1)b$, then $w(x) = 1$, so in this case $w(B) \geq 1$.

– If $x \in \left(\frac{1}{2}, \frac{1}{2} + (i^* - 1)b\right]$, then $w(x) = 1 - \frac{1}{1-b}(\frac{1}{2} + (i^* - 1)b - x)$ and the bin contains small items with cumulative size $level(B) - x \geq \frac{1}{2} + (i^* - 1)b - x$. The weight of these small items (from the definition of weight function) is at least $\frac{1}{1-b}\left(\frac{1}{2} + (i^* - 1)b - x\right)$. Thus $w(B) \geq 1$ in the second case as well.

Noting that the number of "exceptional" bins is at most 1 in the non-empty bin classes and a bin containing a large item cannot be an exception, the total number of exception bins is at most $k+1$, hence the lemma is true. □

Lemma 6 *If Case B holds and S is a subset of the items from L with $\sum_{x\in S} x \leq 1$, then*

$$w(S) = \sum_{x\in S} w(x) \leq \frac{3}{2} + \frac{b}{1-b}.$$

Proof If S contains only small items, then $w(x) \leq \frac{3}{2}x$ for $x \in S$ by definition of w. Therefore

$$w(S) = \sum_{x\in S} w(x) < \sum_{x\in S} \frac{3}{2}x \leq \frac{3}{2}.$$

If S contains a large item x_1, for which $x_1 > \frac{1}{2} + kb$ holds, then $w(x_1) = 1$ and $\sum_{x\in S, x\neq x_1} x \leq 1 - x_1 = \frac{1}{2} - kb$. But if $x < \frac{1}{2} - kb$, then $w(x) \leq \frac{1}{1-b}x$, so

$$\begin{aligned} w(S) := \sum_{x\in S} w(x) &\leq 1 + \sum_{x\in S, x\neq x_1} \frac{1}{1-b}x \\ &\leq 1 + \frac{1}{1-b}\left(\frac{1}{2} - kb\right) = 1 + \frac{1}{1-b}\left(\frac{1}{2} - \frac{1}{2}b + \frac{1}{2}b - kb\right) \\ &= \frac{3}{2} - \frac{(k-\frac{1}{2})b}{1-b} \leq \frac{3}{2} - \frac{b}{2(1-b)} < \frac{3}{2}. \end{aligned}$$

If $x_1 \in S$ and $\frac{1}{2} < x_1 \leq \frac{1}{2} + kb$ then x_1 is the largest item in S. With the assumption that Case B holds, there exists a positive integer i, $1 \leq i \leq k$, such that $x_1 \in I_{k+2+i}$.

There are two cases that need to be examined:

a) If $x_1 > \frac{1}{2} + (i^* - 1)b$, then $i \geq i^*$. If x_2 is the second largest item in S, then we have two subcases:

– if $x_2 \in \{I_1 \cup \ldots \cup I_{k+2-i}\}$ then for each item x $(x \neq x_1)$, $x \leq x_2$, so $x \in \{I_1 \cup \ldots \cup I_{k+2-i}\}$ holds as well. As regards the weight of a small item like x, $w(x) \leq \frac{1}{1-b}x$ is valid, so

$$\begin{aligned} w(S) = \sum_{x\in S} w(x) &= w(x_1) + \sum_{x\in S, x\neq x_1} w(x) \\ &\leq 1 + \frac{1}{1-b} \cdot \frac{1}{2} = 1 + \frac{1}{2-2b} = \frac{3-2b}{2-2b} = \frac{3}{2} + \frac{b}{2(1-b)}. \end{aligned}$$

– if $x_2 \in \{I_{k+3-i} \cup \ldots \cup I_{k+2}\}$, then $x_2 \in I_{k+3-i}$, because otherwise $x_1 + x_2 > 1$. But in this case $\sum_{x\in S, x\neq x_1, x\neq x_2} x \leq 1 - x_1 - x_2 \leq 1 - (\frac{1}{2} + (i-1)b) - (\frac{1}{2} - ib) = b$. It means that $x \in I_1$. Hence

$$\begin{aligned} w(S) &= \sum_{x\in S} w(x) = w(x_1) + w(x_2) + \sum_{x\in S, x\neq x_1, x\neq x_2} w(x) \leq 1 + \frac{1}{2} + \frac{b}{1-b} \\ &= \frac{3}{2} + \frac{b}{1-b}. \end{aligned}$$

b) if $x_1 \le \frac{1}{2} + (i^* - 1)b$, then $x_1 \in I_{k+2+i}$, when $1 \le i < i^*$. If x_2 is the second largest item in S, then we also have two subcases:
   – If $x_2 > \frac{1}{2} - i^* b$, then $w(x_2) = \frac{1}{2}$, and $\sum_{x \in S, x \neq x_1, x \neq x_2} x \le 1 - x_1 - x_2 < 1 - x_1 - (\frac{1}{2} - i^* b) = \frac{1}{2} + i^* b - x_1$.
   From this

$$\begin{aligned} w(S) &= \sum_{x \in S} w(x) = w(x_1) + w(x_2) + \sum_{x \in S, x \neq x_1, x \neq x_2} w(x) \\ &\le 1 - \frac{1}{1-b} \cdot \left(\frac{1}{2} + (i^* - 1)b - x_1 \right) + \frac{1}{2} + \frac{1}{1-b} \left(\frac{1}{2} + i^* b - x_1 \right) \\ &= \frac{3}{2} + \frac{b}{1-b}. \end{aligned}$$

   – If $x_1 \le \frac{1}{2} + (i^* - 1)b$ and $x_2 \le \frac{1}{2} - i^* b$, then for all $x \neq x_1$ items of S $w(x) \le \frac{1}{1-b} x$ holds. As before (case a, first subcase), it can be shown that $w(S) \le \frac{3}{2} + \frac{b}{2(1-b)}$ also applies in this case.

This completes the proof of the lemma. □

If we compare lemmas 3 and 6, we see that in both cases for given $k \in N^+$ we can fix a real number b which satisfies the desired condition $kb < \frac{1}{6}$. But then we will get different upper bounds for *HR-k*. The two multiplicative constants will be equal if the equation

$$\frac{2}{1 + 2kb_k} = \frac{3}{2} + \frac{b_k}{1 - b_k}$$

has a solution for each fixed k satisfying the condition $kb_k < \frac{1}{6}$. One can easily check that the equation $2kb_k^2 - (6k+3)b_k + 1 = 0$ has such a solution, and only one solution of this type. That is,

$$b_k = \frac{6k + 3 - \sqrt{\left(6k + \frac{7}{3}\right)^2 + \frac{32}{9}}}{4k}.$$

We list some values for the respective pairs k and b_k in Table 1.

Let $M_1 = k + 2$ from Lemma 3, and $M_2 = k + 1$ from Lemma 5, and let $M := \max\{M_1, M_2\} = k + 2$. Combining the results of lemma 3, 5, and 6 we get

$$HR\text{-}k(L) \le \max\left\{ \frac{2}{1 + 2kb_k}, \frac{3}{2} + \frac{b_k}{1 - b_k} \right\} OPT(L) + M.$$

Now, we can state an upper bound on the asymptotic competitive ratio of our algorithm for any k value and state the limes of these ratios, taking into consideration the fact that $b_k \in \left(0, \frac{1}{6k}\right)$, for any k value.

Theorem 2 *For any fixed $k \in N^+$*

$$R(HR\text{-}k) \le \frac{3}{2} + \frac{b_k}{1 - b_k},$$

and $R(HR\text{-}k) \to \frac{3}{2}$ for $k \to \infty$.

The next theorem provides a lower bound. Though this bound does not quite match the upper bound, the results are fairly close (this will be discussed later on).

Theorem 3 *For any fixed $k \in N^+$*

$$R(HR\text{-}k) \geq \left(1 - \frac{b_k^2}{(1-b_k)^2}\right)\left(\frac{3}{2} + \frac{b_k}{1-b_k}\right).$$

Proof For each t, $t \in N^+$, let $n = 2t(\lfloor \frac{1}{b_k} \rfloor - 1)$. For each given n we construct a list L with $N = 4n + 8t - 3$ items as a concatenation of four sublists, $L = L_1L_2L_3L_4$, as follows.

(i) $L_1 = (L_{11}L_{12}L_{13})^{2t}$ (the exponent gives the number of repetitions of the concatenated sublists) where
  - L_{11} contains $\lfloor \frac{1}{b_k} \rfloor - 1$ items, each with size $b_k - 2\varepsilon$,
  - L_{12} contains one item with size $1 - b_k\left(\frac{n}{2t} + 1\right) + \frac{n}{t}\varepsilon$,
  - L_{13} contains three items with sizes ε, where

$$\varepsilon < min\left\{\frac{b_k}{n}, \frac{1}{6t+n-3}, \frac{2b_kt + \frac{nb_k}{2} - t}{n}\right\}.$$

(ii) L_2 contains $n - 3$ items of size ε.
(iii) L_3 contains n items of size $\frac{1}{2} - b_k + \varepsilon$.
(iv) L_4 contains of n items of size $\frac{1}{2} + \varepsilon$.

HR-k packs the items of this list in the following way. First, it packs the items of the first list $L_{11}L_{12}L_{13}$ using the pure HF rule into bins of class $\mathcal{B}_1$. The levels of each of these bins will be exactly $1-b_k+3\varepsilon$, so the first item of the next $L_{11}L_{12}L_{13}$ must be put into a newly opened bin. Therefore the algorithm uses

$$2t = \frac{n}{\lfloor \frac{1-b_k}{b_k} \rfloor}$$

bins to pack the items of L_1.

The items from L_2 belong to I_1, so they will also use bins from $\mathcal{B}_1$. Since the last opened bin in class $\mathcal{B}_1$ is filled to level $1 - b_k + 3\varepsilon$, the elements of L_2 can be put into this bin.

The items of L_3 are packed pairwise into bins belonging to the class $\mathcal{B}_{k+2}$.

Lastly, when processing the items from L_4, the algorithm opens a new bin of class $\mathcal{B}_{k+3}$ for each item of L_4, and for each step it repacks one item with size ε from the last opened bin of $\mathcal{B}_1$.

Summarizing, *HR-k* uses

$$HR\text{-}k(L) = n + \frac{n}{2} + \frac{n}{\lfloor \frac{1-b_k}{b_k} \rfloor} \leq n\left(\frac{3}{2} + \frac{b_k}{1-b_k}\right)$$

bins. But, since the sum of the sizes of the items is at most

$$n + 2t\left(1 - b_k - \frac{n}{2t}(b_k - 2\varepsilon)\right) + 1 \leq n + 2tb_k + 1,$$

we get

$$OPT(L) \leq n + \frac{nb_k}{\lfloor \frac{1}{b_k} \rfloor - 1} + 1 \leq n + \frac{nb_k}{\frac{1}{b_k} - 2} + 1 \leq n\left(1 + \frac{b_k^2}{1-2b_k}\right) + 1 = n\frac{(1-b_k)^2}{1-2b_k} + 1.$$

Table 1 Values of b_k and the bounds for different values of k.

k	b_k	**LB**	**UB**	Δ
1	0.113999	1.601704	1.628667	0.026963
2	0.067895	1.564496	1.572842	0.008346
3	0.048285	1.546743	1.550735	0.003992
4	0.037452	1.536580	1.538910	0.002330
5	0.030586	1.530026	1.531552	0.001526
6	0.025846	1.525457	1.526532	0.001075
7	0.022378	1.522092	1.522890	0.000798
8	0.019729	1.519511	1.520127	0.000616
9	0.017642	1.517469	1.517959	0.000490
10	0.015953	1.515813	1.516212	0.000399
11	0.014560	1.514444	1.514775	0.000331
12	0.013390	1.513293	1.513572	0.000279
17	0.009553	1.509505	1.509646	0.000141
34	0.004838	1.504826	1.504862	0.000036
42	0.003926	1.503918	1.503942	0.000024
56	0.002952	1.502948	1.502961	0.000013
84	0.001973	1.501971	1.501978	0.000007
167	0.000995	1.500994	1.500996	0.000002

Therefore

$$R(HR\text{-}k) = lim_{n\to\infty} \frac{n\left(\frac{3}{2}+\frac{b_k}{1-b_k}\right)}{n\frac{(1-b_k)^2}{1-2b_k}+1} = lim_{n\to\infty} \frac{\frac{3}{2}+\frac{b_k}{1-b_k}}{\frac{(1-b_k)^2}{1-2b_k}+\frac{1}{n}}$$
$$= \left(\frac{3}{2}+\frac{b_k}{1-b_k}\right)\left(\frac{1-2b_k}{(1-b_k)^2}\right),$$

which proves the theorem. □

One consequence of the above two theorems is

Corollary 2 *For any fixed* $k \in N^+$

$$\left(\frac{3}{2}+\frac{b_k}{1-b_k}\right)\left(\frac{1-2b_k}{(1-b_k)^2}\right) \le R(HR\text{-}k) \le \frac{3}{2}+\frac{b_k}{1-b_k}.$$

Table 1 shows lower and upper bounds for the most interesting values of k, namely for $k = 1,\ldots,12$ and for those k values where the corresponding upper bound falls below the values 1.51, 1.505, 1.504, 1.503, 1.502 and 1.501 ($k = 17, 34, 42, 56, 84, 167$) for the first time.

4 Conclusions, future work and open questions

In this article we examined a family of semi-on-line algorithms for the classical one-dimensional bin-packing problem. These algorithms allow the repacking of at most a fixed number of items in each step. It is clear that this type of algorithm is worthwhile if the performance is better than that of standard on-line algorithms. Even though our algorithms are semi-on-line, we compare their results to the corresponding on-line case. With increasing values of k, the asymptotic competitive ratio of our algorithms quickly converges to 1.5 and our method gives better results than the classical on-line algorithms already for small k. Especially there are two interesting cases. $R(HR\text{-}2) = 1.5728\ldots$ is better than the asymptotic ratio $1.58889\ldots$ of the best known on-line algorithm [25]. $R(HR\text{-}4) = 1.5389\ldots$ is smaller than the best known lower bound $1.5403\ldots$ for on-line algorithms published in [2]. These results tell us that our algorithm is capable of exploiting the additional flexibility obtained from the chance of repacking items.

Although the algorithms described in the papers [13] and [19] have an ACR that is less than or equal to 1.5, we should point out again that we used a repacking definition that is different from theirs. Namely, in [13] and [19] a grouping operation is allowed where a set of small items can be created, and the repacking of a set of small items counts as 1-repacking (even the number of small items in a set is $O(\log n)$). Hence, there is no direct connection between the two types of algorithms from this point of view. Our algorithms do not use the grouping operation (because it is not allowed) and in our terminology the repacking of every single item is counted as 1-repacking in a step. Furthermore, we allow only a fixed number of repacking per step.

Another remark is that in the cost of our algorithms we counted only the number of non-empty bins when packing the whole list. We did this because it may happen that bins become empty as a consequence of repacking. Otherwise, the cost would be totally different in our algorithms. This way our algorithms work well for the Salzer series [24]. For the list L that is used for the proof of the lower bound 1.5 [26], it can be readily verified that any *HR-k* uses fewer than $\frac{7}{6}n$ bins (if $\varepsilon < \frac{1}{6} - b_1$). In this construction, L is the concatenation of n elements of $\frac{1}{6} - 2\varepsilon$, n elements of $\frac{1}{3} + \varepsilon$ and n elements of $\frac{1}{2} + \varepsilon$. If we take emptied bins into consideration, then we would have $\frac{5}{3}n$ bins. Similar results can be obtained for the lists used in [5] and [21]. It is an interesting question of whether T_∞ is an upper bound for every *HR-k* algorithm if emptied bins are counted.

There are other open questions for further study. One is that although the gap between the upper and lower bounds is already very small for $k = 1$, one would like to know if it can still be improved. Another is that a semi-on-line algorithm that may repack only one item in each step can have an asymptotic ratio better than 1.58889 or even 1.58333. These are the best known upper bounds on the ACR of the current best on-line algorithm and the lower bound of any Super Harmonic type on-line algorithm [25,22], respectively. For $k < 4$, it would be interesting to know whether an ACR better than $1.5403\ldots$ can be achieved. Unfortunately, our algorithm does not improve the upper bound of the best known online algorithm of Seiden [25] for $k = 1$. But in this case our algorithm with a simple interval structure uses only 6 bin classes, while the very sophisticated Harmonic++ of [25] uses a complicated interval structure with 76 subclasses. Furthermore, we strongly conjecture that the upper bound in the case of $k = 1$ can be improved to 1.59 by using repacking and about 15 bin classes. These are far fewer bin classes than the 76 used in [25]. This study is in progress.

It is also interesting to note that a lower bound is given for a class of pure on-line bin packing algorithms in [22]. These algorithms use $h + 1$ ($h = 1, 2, \ldots$) interval partitions of the $(1/2, 1]$ and of the $(1/3, 1/2]$ intervals. For the cases $k \geq 2$, the upper bound of our *HR-k* algorithm is below the lower bound stated in [22]. In the case of $h = 1$, the lower bound of [22] is $1.6111\ldots$, which can also be achieved by the above-mentioned modification of our $HR - 1$ algorithm, based on the modification of its interval structure and analyzing both cases of the proof using a weight function.

Acknowledgement

The authors are grateful to the reviewers' valuable comments, that improved the manuscript.

References

1. J. Balogh and J. Békési, Semi-on-line Bin Packing: A Short Overview and a New Lower Bound, submitted for publication, 2012.
2. J. Balogh, J. Békési, and G. Galambos, New Lower Bounds for Certain Classes of Bin Packing Algorithms, *Proceedings of WAOA 2010 (8th Workshop on Approximation and Online Algorithms), LNCS 6534,* pp. 25–36, 2011.
3. J. Balogh, J. Békési, G. Galambos, and G. Reinelt, Lower Bound for the Online Bin Packing Problem with Restricted Repacking, *SIAM J. Computing* **38**, 398–410, 2008.
4. J. Balogh and G. Galambos, Algorithms for the on-line bin packing problem with repacking, *Alkalmazott Matematikai Lapok*, **24**, 117–130, 2007. (in Hungarian)
5. D.J. Brown, A lower bound for on-line one-dimensional bin packing algorithms, Tech. Rept. R-864, Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1979.
6. J.W-T. Chan, T-W. Lam, P.W.H. Wong, Dynamic bin packing of unit fractions items, *Theoretical Computer Science* **409**, 521–529, 2008.
7. Coffmann, E.G., M.R. Garey, and D.S. Johnson, Dynamic bin packing, *SIAM Journal on Computing* **12**(2), 227–260, 1983.
8. E.G. Coffman, G. Galambos, S. Martello, and D. Vigo, Bin Packing Approximation Algorithms: Combinatorial Analysis. In: *Handbook of Combinatorial Optimization* (Eds. D.-Z. Du and P.M. Pardalos), pages 151–208. Kluwer Academic Publishers, 1999.
9. L. Epstein and A. Levin, On bin packing with conflicts, *SIAM Journal on Optimization*, 19:1270–1298, 2008.
10. W. Fernandez de la Vega, and G.S. Lueker, Bin Packing can be Solved Within $1+\varepsilon$ in Linear Time, *Combinatorica* **1**, 349–355, 1981.
11. G. Galambos, A New Heuristic for the Classical Bin Packing Problem, Technical Report 82, Institut für Mathematik, Universität Augsburg, 1985.
12. G. Galambos and G.J. Woeginger, Repacking Helps in Bounded Space On-line Bin Packing, *Computing* **49**, 329–338, 1993.
13. G. Gambosi, A. Postiglione, and M. Talamo, Algorithms for the Relaxed Online Bin-Packing Model, *SIAM J. Computing* **30**, 1532–1551, 2000.
14. M.R. Garey and D.S. Johnson, *Computers and Intractability (A Guide to the theory of NP-Completeness).* W.H. Freeman and Company, 1979.
15. E.F. Grove, Online Bin Packing with Lookahead, *SODA* 1995: 430–436.
16. G. Gutin, T. Jensen, and A. Yeo, Batched Bin Packing, *Discrete Optimization* **2**, 71–82, 2005.
17. X. Han, C. Peng, D. Ye, D. Zhang, and Y. Lan, Dynamic bin packing with unit fraction items revisited, *Information Processing Letters* **110**(23), 1049–1054, 2010.
18. Z. Ivkovič and E.L. Lloyd, A Fundamental Restriction on Fully Dynamic Maintenance of Bin Packing, *Information Processing Letters* **59**, 229–232, 1996.
19. Z. Ivkovič and E.L. Lloyd, Fully Dynamic Algorithms for Bin Packing: Being (Mostly) Myopic Helps, *SIAM J. Computing* **28**, 574–611, 1998.
20. C.C. Lee and D.T. Lee, A Simple On-line Bin Packing Algorithm, *J. of the ACM* **32**, 562–572, 1985.
21. F.M. Liang, A lower bound for on-line bin-packing, *Information Processing Letters* **10**, 76–79, 1980.
22. P. Ramanan, D.J. Brown, C.C. Lee, and D.T. Lee, On-line Bin Packing in Linear Time, *J. of Algorithms* **10**, 305–326, 1989.
23. M.B. Richey, Improved bounds for harmonic-based bin packing algorithms, *Discrete Applied Mathematics* **34**, 203–227, 1991.

24. H.E. Salzer, The approximation of numbers as sums of reciprocals, *Am. Math. Monthly* **54**, 135–142, 1947.
25. S.S. Seiden, On the Online Bin Packing Problem, *J. of the ACM* **49**, 640–671, 2002.
26. A.C. Yao, New algorithms for bin packing, *J. of the ACM* **27**, 207–227, 1980.