SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1532–1551

ALGORITHMS FOR THE RELAXED ONLINE BIN-PACKING MODEL*

GIORGIO GAMBOSI†, ALBERTO POSTIGLIONE‡, AND MAURIZIO TALAMO§

Abstract. The typical online bin-packing problem requires the fitting of a sequence of rationals in $(0, 1]$ into a minimum number of bins of unit capacity, by packing the ith input element without any knowledge of the sizes or the number of input elements that follow. Moreover, unlike typical online problems, this one issue does not admit any data reorganization, i.e., no element can be moved from one bin to another.

In this paper, first of all, the "Relaxed" online bin-packing model will be formalized; this model allows a constant number of elements to move from one bin to another, as a consequence of the arrival of a new input element.

Then, in the context of this new model, two online algorithms will be described. The first presents linear time and space complexities with a 1.5 approximation ratio and moves, at most once, only "small" elements; the second, instead, is an $O(n \log n)$ time and linear space algorithm with a $1.33\ldots$ approximation ratio and moves each element a constant number of times. In the worst case, as a result of the arrival of a new input element, the first algorithm moves no more than three elements, while the second moves as many as seven elements. Please note that the number of movements performed is explicitly considered in the complexity analysis.

Both algorithms are below the theoretical $1.536\ldots$ lower bound, effective for the online bin-packing algorithms without the movement of elements. Moreover, our algorithms are "more online" than any other linear space online bin-packing algorithm because, unlike the algorithms already known, they allow the return of a (possibly relevant) fraction of bins before the work is carried out.

Key words. complexity, approximation algorithms, online algorithms, bin packing

AMS subject classifications. 68Q25, 68R05, 90C27

PII. S0097539799180408

1. Introduction.

1.1. The bin-packing problem. The bin-packing problem (see survey in [7] and in [8]) is a major issue in theoretical computer science: it consists of "packing" a set of nonoverlapping objects into a minimum number of well-defined areas. More formally [7], [8], given a positive integer C, it provides for the packing of a set of integer size elements $L = \{a_1, a_2, \ldots, a_n\}$, with $size(a_i) \in (0, C] \cap N_0$, into a minimum number of bins of equal capacity C.

This problem models the variable partitioning storage management in multiprogrammed computer systems and the assignment of commercials to mass media station breaks and truck packing. Bin-packing also models a variant of the scheduling problem in multiprocessors where the objective is to minimize the number of processors in

*Received by the editors January 1, 1999; accepted for publication (in revised form) January 28, 1999; published electronically November 17, 2000. This work was partially performed in the framework of Esprit BRA project 3075, "Algorithms and Complexity," and of Italian MURST 40% project, "Algoritmi e Strutture di Calcolo," and was partially supported by National Research Council project "Sistemi Informatici e Calcolo Parallelo."

http://www.siam.org/journals/sicomp/30-5/18040.html

†Dipartimento di Matematica, University "Tor Vergata," via della Ricerca Scientifica, 00133 Rome, Italy (gambosi@mat.uniroma2.it).

‡Dipartimento di Scienze della Comunicazione, University of Salerno, via Ponte don Melillo, 84084 Fisciano (SA), Italy (ap@unisa.it).

§Dipartimento di Informatica e Sistemistica, University "La Sapienza," via Salaria 113, 00198, Rome, Italy (talamo@dis.uniroma1.it).

which all tasks are to be completed within a given deadline. (When the common deadline is the capacity, processors are represented by bins and elements are represented by tasks whose size is given by the execution time.)

An interesting case is when $C = 1$ and $size(a_i) \in (0,1] \bigcap Q$, which results in a combinatorial optimization problem that is NP-hard in the strong sense[1] since it contains 3-Partition as a special instance [15].

We are interested in searching for approximated fast (polynomial) online bin-packing algorithms that require the packing of the ith element without information on the sizes or the number of the following input elements and whose solution is far from the optimal for a small, fixed, multiplicative constant.

All the known online algorithms share the approach that no element can be moved from the bin it was first inserted in. Moreover, all the $\Theta(n)$-*space* algorithms are offline on output, i.e., no algorithm releases any of the used bins until the end of the input list has been reached, while all the $\Theta(1)$-*space* algorithms release all the bins except for a constant number of them.[2]

As pointed out in [23] a bin-packing algorithm is an algorithm made up of two parts: the first part reorders the list according to a preprocessing rule; the second part generates the packing. An online algorithm has no preprocessing step.

1.2. The relaxed model for the online bin-packing problem. In this paper, we will focus our attention on online algorithms, according to the classical definition [1], which states the following.

Definition 1.1. *The online execution of a sequence of instructions σ requires that the instructions in σ be executed from left to right and that the ith instruction in σ be executed without looking at any of the instructions that follow.*

The above definition corresponds to the definition of online algorithms considered in task systems and server problems (see, for example, [4] and [28]). Please note that the above definition admits internal data reorganization, which is a frequent practice in most online algorithms.

According to these definitions, we introduce a new online bin-packing model, named "Relaxed," which allows a constant number of elements to be moved from one bin to another consequent to the arrival of new input elements. This new model calls for a careful definition of a cost function on the set of the possible item movements, in order to account explicitly for them in the overall algorithm complexity analysis.

Please note that a very limited number of applications of the bin-packing problem cannot be represented by our model. A typical example of such an application is the cutting stock problem, where the more abstract operation "assign an element of size s to a bin" is interpreted in terms of "cut a piece of length s from a stock element."

On the other side there are many real-life situations where the rearrangement of an allocated element is possible and this affects (by lowering) the cost of the resolution process, i.e., packing trucks and multiprocessor memory management strategies. The "Relaxed" model works very well in all situations in which the operation modeled by the assignment of an element to a bin can be "undone" by paying something. Such

[1] Note that, when the number of possible element sizes is a priori bounded, or (it is the same) in the integer formulation C is fixed, the problem can be exactly solved in polynomial time by exhaustive search, although the degree of the polynomial can be very high [7], [8] or it can be exactly solved asymptotically using special linear programming techniques [16], [17]; moreover, the decision problem "Is there a partition of L into disjoints sets $L_1, L_2, \dots, L_k$ such that $\sum_{a \in L_i} size(a) \leq C$, for each L_i" is NP-complete and solvable in pseudopolynomial time for each fixed $k \geq 2$.

[2] Online algorithms which are not offline on output were considered in [10].

a situation arises, for example, when the input list is not known in advance (it could be infinite, too) and each element (that could arrive with a considerable delay from the previous one) needs to be processed online, while the guaranteed performance needs to be maintained at any time for the set of elements currently involved. In this situation a classical online algorithm (BEST-FIT, $HARMONIC_M$, etc.) or an offline algorithm could be applied in correspondence to each element arrival. In the first case no known algorithm uses, in the worst case, more than 63% of the bin space, while no algorithm can use, in the worst case, more than 65% of the bin space (since 1.53 is the theoretical lower bound). In the second case whenever an element arrives all other elements can be moved out from the bin they are contained in and be assigned to some other bin, according to the new computed solution. In our model only a (known) limited number of element movements is admitted, in correspondence to the arrival of a new element.

In general, this new model is particularly suitable when the elapsed time between two consecutive input elements is $\geq \delta K$, where δ is the maximum cost for each element movement, and K is the maximum number of element movements occurring in correspondence to each input element (so this model permits us to take advantage of the "dead times" between two successive input elements).

In [11] and [13] we informally introduced the "Relaxed" model and gave an $O(n \log n)$-*time* $O(n)$-*space* class of algorithms that, for each prefix of the input sequence, returns a 1.5 asymptotical approximation ratio. This value is below the $1.53\ldots$ theoretical lower bound [5], [26] well grounded for the restricted case and indicates that the relaxation of the classical online bin-packing problem conditions is convenient and theoretically interesting. Some experimental simulations allow us to guess that this class of algorithms has (on the average) very good behavior.

Our paper shows how this result is improved in two different ways by giving two linear space algorithms: the first presents a 1.5 approximation ratio with an $O(n)$ time complexity; the second presents a $1.33\ldots$ approximation ratio with an $O(n \log n)$ time complexity (they fill each bin in the worst case at least for 66% and 75%). Moreover, at the arrival of each input element, in the worst case the first algorithm moves no more than three elements while the second may move up to seven elements.

Last, please note that these algorithms are "more online" than all the other linear space online bin-packing ones because, unlike the known algorithms, they allow the return of a (possibly relevant) fraction of the bins before the work is carried out.

Section 2 gives definitions and a brief summary of the previous results on the online bin-packing problem; section 3 shows the "Relaxed" model, with regard to element movement, small element grouping operations, and definition of movement evaluation function; section 4 introduces the linear algorithm A_1 while section 5 gives an analysis of its performance; the $O(n \log n)$ algorithm A_2 is introduced in section 6 and its performance is analyzed in section 7; section 8 examines some conclusions and open problems.

2. Definitions and previous results.

2.1. Problem definition. The classical one-dimensional bin-packing problem can be stated [15] as follows.

DEFINITION 2.1. *Given a finite set* $L = \{a_1, a_2, \ldots, a_n\}$ *of "elements" and a rational "size,"* $size(a) \in (0, 1]$, *for each element* $a \in L$, *find a partition of* L *into disjoint subsets* $L_1, L_2, \ldots, L_k$ *such that the sum of the sizes of the elements in each* L_i *is no greater than* 1 *and such that* k *is as small as possible.*

Since the bin-packing problem contains 3-partition as a special case, it is an NP-hard problem in the strong sense [15], [21]. It is therefore very unlikely that there are fast (polynomial) algorithms for finding the best solution, unless P = NP, even if the magnitude of the numbers involved is bounded by a polynomial in n.

Given an (approximate) algorithm A for the bin-packing problem and a set L of elements, let $A(L)$ be the number of bins used by A to pack L. Therefore

$$OPT(L) \geq \sum_{a_i \in L} size(a_i) \tag{2.1}$$

is a lower bound for the number of bins necessary to pack L.

Now we are able to give some algorithm performance definitions [7], [8].

Definition 2.2. *The performance of A with respect to OPT on the list L is*

$$R_A(L) \equiv \frac{A(L)}{OPT(L)}. \tag{2.2}$$

Definition 2.3. *The absolute performance ratio R_A of the algorithm A is*

$$R_A \equiv \inf\{r \geq 1 | R_A(L) \leq r, \ \forall \ list \ L\}. \tag{2.3}$$

Definition 2.4. *The asymptotic performance ratio R_A^∞ of the algorithm A is*

$$R_A^\infty \equiv \inf\{r \geq 1 | \ for \ some \ N > 0, R_A(L) \leq r, \ \forall \ L \ with \ OPT(L) \geq N\} \tag{2.4}$$

Note that $R_A \geq R_A^\infty$.

2.2. Previous results on the online version. The classical problem presents a variety of cases [7], [8]. In the online version [10] the following definition exists.[3]

Definition 2.5. *"Items are assigned to bins in order $(a_1, \ a_2, \ \ldots)$, with item a_i assigned only according to the size of the previous items and the bins to which they were assigned, without considering the size or number of items that follow."*

The simplest bin-packing algorithm is Next-Fit [7], [8] which is $O(n)$-*time* and $O(1)$-*space*, but whose asymptotical performance, both in the worst and in the average cases, is very poor, respectively, 2.0 and 1.33 [7], [8], [9], [12].

The bin-packing algorithms most extensively used are BEST-FIT and FIRST-FIT [23]. Both algorithms are $O(n \log n)$-*time* and $O(n)$-*space* and present an acceptable asymptotical worst-case performance (i.e., 1.7, [23]) but an optimal asymptotically average performance [2], [3], [12], [32].[4]

Until now, the best online algorithms for the bin-packing problem, without moving elements from the bins they have been assigned to, belonged to the HARMONIC class, first introduced in [25], where an approximated algorithm called $HARMONIC_M$ was introduced; this algorithm is the optimal among all the $O(1)$-*space* algorithms. Such an algorithm has an $O(n)$-*time* complexity and a ratio $R_H^\infty(M) \leq 1.692$ for all $M \geq 12$. Lee and Lee [25], moreover, proved that $R_A^\infty \geq 1.6910$ for all constant space algorithms and that $\lim_{M\to\infty} R_H^\infty(M) = 1.6910$.

[3] More appropriately in such a case, we deal with a sequence of elements to be packed and not with a set.

[4] Note that in [10] it is proved that BEST-FIT obtains its worst-case performance even if a constant ($k \geq 2$) number of bins is maintained online; this reduces the computation time to $O(n \log k)$, that is, $O(n)$ since k is a small constant.

The same authors gave a more complex $O(n)$ algorithm, REFINED HARMONIC, which uses $O(n)$ space and presents a ratio $R^{\infty}_{RH} = \frac{373}{228} = 1.636\ldots$.

Later, the MODIFIED HARMONIC algorithm was introduced in [31], which is $O(n)$ both in time and in space complexity with a ratio $R^{\infty}_{MH} = 1.61(561)^*$. The authors also showed how an online algorithm with $R^{\infty}_{A} < 1.59$ can be obtained.

3. The relaxed model.

3.1. Motivations and previous results. All the algorithms mentioned in section 2.2 introduce the additional limit that the solution for $\langle a_1\ a_2\ \ldots\ a_i\rangle$ must derive from the one for $\langle a_1\ a_2\ \ldots\ a_{i-1}\rangle$ without performing any reorganization of the elements in the bins; that is, none of the elements among $a_1, a_2, \ldots, a_{i-1}$ can be moved from the bin it belongs to. In other words, all of these algorithms only search for a suitable bin to which to assign element a_i in order to obtain a good asymptotic approximation of the optimal solution. In this context some interesting lower bounds have been proved, as already pointed out.

A question arising from the above considerations is the following: "What happens if we interpret the online property of bin-packing in a less restricted way, just like the large majority of online models? Is it possible to obtain more efficient performances if a bin-packing algorithm can move the elements a certain number of times from one bin to another?"

In [11] and [13] an affirmative answer to this question is given by presenting a class of online algorithms, $HARMONIC_{REL}(M)$,[5] with time complexity $O(n \log n)$, space complexity $O(n)$, and asymptotic ratio $R^{\infty}_{HREL}(M) \leq 1.5$. In the worst case, the approximation ratio is independent of M, for $M \geq 3$, and the number of movements is limited in an amortized way by a (small) constant (2, for $M = 3$).

3.2. Grouping elements. In this paper we introduce a new operation: the "grouping of elements," i.e., *we assume that a certain number of very small items in the same bin can be collected together and considered as a single unit.* More formally,

Given a constant $0 < c < 1$, *we assume that any set of elements smaller than* c *in the same bin can be collected together in a single group of overall size* $\leq c$. *This group will be considered as a single unit from now on.*

Obviously, the grouping of elements does not modify the approximation ratio of OPT, since OPT is measured as the sum of the elements in the input list. We also assume that there is no kind of movement inside any group.

In the bin-packing problem the grouping operation is possible and convenient. For example, in the truck-packing problem it is useful to fit a collection of very small elements in the same box and then move them as a whole by moving the box. In multiprocessor storage management strategies, the grouping simply consists in collecting a subset of pages.

3.3. Moving elements. In the relaxed model, the critical operation regards moving (part of) the contents from one bin to another.

In the following, i, j are two bins and σ is a subsequence of (not necessarily contiguous) input element(s), all contained in the same bin.

The fundamental operation could therefore be stated as

$$\text{MOVE}(i, j, \sigma), \qquad i \neq j, \tag{3.1}$$

which means that σ is moved from bin i to bin j.

[5] Where REL stands for "Relocation."

This approach is quite natural for all the applications in which we may assume that several small elements can be "carried" from one bin to another in a single step.

3.4. MOVE operation cost. An online algorithm processes the input data one at a time, possibly modifying its internal data structures. Thus the evaluation of the performance of an algorithm is more realistic if it takes into account the number of movements of the elements in its data structures.

In a bin-packing algorithm when the elements move, several kinds of cost functions for the MOVE(i, j, σ) operation could be defined.

DEFINITION 3.1. *The cost of the* MOVE *operation is equal to the total size of all elements moved ($\sum_{x\in\sigma} size(x)$).*

DEFINITION 3.2. *The cost of the* MOVE *operation is equal to the number of elements moved ($|\sigma|$).*

In this paper we will consider a third way to define such a function. In our approach *we assume that each group can be moved at unitary cost.* That is, while moving a "large" element always has a cost equal to 1, we assume that "small" elements can be *grouped* together and moved as a whole, at unit cost. Therefore we have the following.

DEFINITION 3.3. *The cost associated to the* MOVE(i, j, σ) *operation is equal to the number of elements and groups contained in σ.*

If the element moving cost would only be a function of the size of the elements, any reasonable algorithm would tend to move a lot of small elements because the performance is better and there is no cost difference in moving a lot of small elements instead of a few big elements. If the element moving cost would only be a function of the number of the elements moved, there will be no cost difference between an algorithm that moves light elements and another that moves the same number of heavy elements. Therefore, the third cost function is the most likely. It should be clear that for any c, σ this function has a value which is in between the values assumed by the first two cost functions above defined.

3.5. Formal definition of grouping. Let us consider the following nonuniform partition of $(0, 1]$ in $M + 1$ subintervals:

$$(0,1] = \bigcup_{k=0}^{M} I_k,$$

$$I_0 = \left(\frac{3}{4}, 1\right];\ \ I_1 = \left(\frac{2}{3}, \frac{3}{4}\right];\ \ I_2 = \left(\frac{1}{2}, \frac{2}{3}\right];\ \ \ldots;\ \ I_{M-1} = \left(\frac{1}{M-1}, \frac{1}{M}\right]; I_M = \left(0, \frac{1}{M}\right].$$

Let $c = \frac{1}{M}$ be the *border item size.* The grouping operation consists of collecting a set of elements smaller than c in a single group g (that will be a sort of "superitem"), so that, in each bin B (let $size(g) = \sum_{a\in g} size(a)$),

- for all $g \in B$, $size(g) \le c$;
- there are no pairs of groups $g \in B, h \in B$, such that $size(g) + size(h) \le c$;
- each group $g \in B$, except at most one, has $size(g) \ge \frac{1}{2}c$.

3.6. Grouping primitives.

3.6.1. Create group. This primitive regards the arrival of a new element in $(0, \frac{1}{M}]$ that cannot be merged in any group of the target bin. The operation consists in creating an empty group and in inserting this new element in it. At all times, there will be no more than one I_M-bin open.

3.6.2. Append. This primitive regards the arrival of a new element in $(0, \frac{1}{M}]$ that has to be merged into an existing group.

3.6.3. Primitive performances. Since we are not interested in any kind of arrangement of the elements within the group, a suitable representation (i.e., linked lists) allows all of these operations to be executed in constant time and space. This leads to the "packing" of such elements together, so that they can/must be moved as a whole, in one single step.

3.7. Evaluation function. Below we will show that each bin will contain a constant number of groups. Since the algorithm performance is measured as a function of the space wasted with respect to the sum of the sizes of the elements in the input list, the grouping of the elements does not affect the performance in any manner.

We will not detail the operations involved in inserting and deleting elements to and from bins nor the ones involved in the maintenance of the support data structures, mainly in empty conditions, because they can be easily performed in constant time.

In the following, let

m be the maximum number of MOVE operations performed upon the arrival of a new element[6];

r be the asymptotic performance of the algorithm.

Thus, we can assign to an approximation algorithm a pair of numbers, such as

$$A(m, r).$$

For example, the well-known BEST-FIT algorithm is A(1,1.7) since its performance ratio is 1.7. In general, we can say that a classical online bin-packing algorithm is A(1,r) ($r \geq 1.53$) since it does not move the elements already fitted in the bins and 1.53 is the lower bound for this kind of algorithm. Please note that the exact algorithm is A(m,1), for some $m \geq 0$, while our first algorithm, A_1, is A(3,1.5), and A_2 is A(7,1.33).

4. The linear algorithm A_1. A_1 is based on a nonuniform partition of interval $(0, 1]$ into four subintervals (levels):

$$(0, 1] = \bigcup_{k=0}^{3} I_k,$$

$$I_0 = \left(\frac{2}{3}, 1\right]; \quad I_1 = \left(\frac{1}{2}, \frac{2}{3}\right]; \quad I_2 = \left(\frac{1}{3}, \frac{1}{2}\right]; \quad \text{and } I_3 = \left(0, \frac{1}{3}\right].$$

In order to describe it, let us introduce the following points:

- $S = \langle a_1\ a_2\ \ldots\ a_n \ldots \rangle$ is the "input list."
- Let $a_i \in I_k$ ($0 \leq k \leq 2$) be an element of S. Then a_i is called "I_k-element."
- Let $a_i \in I_3$ be an element of S. Then a_i is called "I_3-group."
- Let B be a bin. Then B is an "I_k-bin (I_3-bin)" ($0 \leq k \leq 2$) if the first element that was initially assigned to it were an I_k-element (I_3-group).
  By subinterval definition, each I_k-bin ($1 \leq k \leq 2$) contains no more than k I_k-elements and each I_0-bin contains no more than one I_0-element.

[6]the first insertion of a new element corresponds to a MOVE from outside into a bin

If an I_k-bin ($1 \le k \le 2$) exactly contains k I_k-elements it is *"filled"*; otherwise it is *"unfilled"*. An I_0-bin with an I_0-element is *"filled"* and an I_3-bin is filled only when its gap is $< \frac{1}{3}$.[7]

- For each k ($0 \le k \le 3$) let A_k be the name of the only unfilled I_k-bin.
- "*gap(B)*" is the space available in an I_k-bin, B ($0 \le k \le 2$), to insert I_l-elements (I_3-groups) ($k < l \le 2$). If B is filled, then $gap(B) = 1 - \sum_{a \in B} size(a)$; otherwise we conventionally assume that $gap(B) = 0$.

Algorithm A_1 is reported below (where l denotes the level of the next input element, x). Please note that if $x \in I_1$ is a "small" I_1-element (i.e., $size(x) < \frac{2}{3}$), A_1 tries to insert some I_3-groups in its gap; if this is not possible A_1 will mark this bin for a future I_3-group insertion. Please note that if $x \in I_3$, then A_1 first tries to insert it in the gap of some marked I_1-bin with enough room.

The algorithm uses two stacks of bins, L_1 and L_3, respectively associated with levels 1 and 3. L_1 maintains all the bins whose gap is still "fat" (i.e., $\ge \frac{1}{3}$), while L_3 maintains all the I_3-bins. If there is an unfilled I_3-bin, then it is the first bin in L_3. Please note that in every moment no more than one between L_1 and L_3 can be "not empty."

We do not explicitly consider the management of unfilled bins. For example, we assume that an unfilled bin is automatically generated at the arrival of an element which can be assigned to no other bin available at that time.

Algorithm A_1

For each input element x**:**
  if $x \in I_0$ **then** "Insert x in A_0".
  if $x \in I_1$ **then**
  - "Insert x in A_1";
  - **while** ($gap(A_1) \ge \frac{1}{3}$) **AND** ("There still exists an I_3-group", g) **do** "Move g to A_1."
  - **if** "There is no more I_3-groups" **AND** ($gap(A_1) \ge \frac{1}{3}$) **then** "Push A_1 in L_1."

  if $x \in I_2$ **then**
  - "Insert x in A_2";

  if $x \in I_3$ **then**
  - **if** "There exists an I_1-bin, B, in L_1"
    then "Insert x in B, removing B from L_1 if its gap becomes $< \frac{1}{3}$."
    else "Insert x in A_3"

5. Performance analysis of A_1.

In order to analyze the performance of A_1 we must first consider the total number of element movements within the bins at the arrival of a new element. Next, we will consider its asymptotic performance ratio.

5.1. Time, space, and movements.

LEMMA 5.1. *Each filled I_1-bin contains no more than two groups in its gap.*

Proof. Let us assume there are more than two groups in a filled I_1-bin. Let x, y, z be three of them. Since an I_1-bin B has $gap(B) < \frac{1}{2}$, it follows that $size(x) + size(y) + size(z) < \frac{1}{2}$. By definition, we know that in every bin all the groups except for one (at most) are $\ge \frac{1}{6}$ in size. Without loss of generality (w.l.o.g.) let us assume that $size(x) \ge \frac{1}{6}$. Therefore

$$\frac{1}{2} > size(x) + size(y) + size(z) \ge \frac{1}{6} + size(y) + size(z) \Rightarrow size(y) + size(z) < \frac{1}{3},$$

which is a contradiction, since every pair of groups in each bin has a total size $> \frac{1}{3}$. ∎

[7] Note that we distinguish among "I_k-filled bins" (that is, bins no more able to receive all possible items of their class, but still active) and "I_k-full bins" (that is, bins whose gap is empty or that are never used afterwards).

Please note that this bound is tight. It is easy to show that two groups can be fitted together in the gap of this bin. An example is the following: $(\frac{1}{2}+\epsilon), (\frac{1}{6}-2\epsilon), (\frac{1}{6}+3\epsilon)$.

COROLLARY 5.2. *No more than three movements will be performed at each insertion.*

Proof. The above lemma proves how the movements only occur at the arrival of I_1-elements with $size < 2/3$. However, the algorithm performs no more than three movements since each I_1-bin has a $gap < \frac{1}{2}$ and each pair of groups has $size > \frac{1}{3}$. This implies that, in the worst case, it is sufficient to move one group from A_3 and two groups from another bin in L_3. ☐

THEOREM 5.3. *Algorithm A_1 has space complexity $O(n)$ and time complexity $\Theta(n)$.*

Proof. The space complexity easily derives from the observation that each element is represented no more than once in L_1 and L_3.

As far as time complexity, according to the above lemma we know that the maximum number of element insertions in a bin is bounded by $3n$. Each insertion can be performed in $O(1)$ time. Moreover, the movement of an existing element is performed in $O(1)$ time since this movement uses the first element in the first bin on the list, accessed in constant time. Therefore, the time complexity is easily derived. ☐

5.2. Performance ratio. In order to derive the approximation ratio for A_1, the following lemmas are needed.

LEMMA 5.4. *If, after all elements have been considered, L_3 is not empty, then $R^{\infty}_{A_1} < \frac{3}{2}$.*

Proof. The gaps of I_0-bin and I_k-bin ($k \geq 2$) are $< \frac{1}{3}$, by definition.

As far as I_1-bins please note that there is at least one element in L_3 whose size is $\leq \frac{1}{3}$, which has not been moved to the gap of any I_1-bin; this implies that all the gaps of I_1-bins have size $\leq \frac{1}{3}$.

In conclusion, the maximum gap in each bin is $< \frac{1}{3}$ and, consequently,

$$R^{\infty}_{A_1} < \frac{3}{2}. \qquad \square$$

LEMMA 5.5. *If, after all the elements have been considered, L_3 is empty and in the input sequence $L = \{a_1, a_2, \ldots, a_n\}$ there was no pair of a_i, a_j, so that $a_i \in I_1$ and $a_j \in I_2$, then $R^{\infty}_{A_1} = 1$.*

Proof. In this case A_1 uses $N_0 + N_1$ or $N_0 + \frac{N_2}{2}$ bins, where N_j is the number of I_j-elements in the input set, since no I_1-elements or I_2-elements could be inserted in any bin which already has an I_0-element and 2 I_2-elements are inserted in the same I_2-bin. Since OPT cannot use fewer bins, then

$$R^{\infty}_{A_1} = 1. \qquad \square$$

LEMMA 5.6. *If, after all the elements have been considered, L_3 is empty and in the input sequence $L = \{a_1, a_2, \ldots, a_n\}$ there was at least one pair a_i, a_j, so that $a_i \in I_1$ and $a_j \in I_2$, then $R^{\infty}_{A_1} \leq \frac{3}{2}$.*

Proof. Let B_i be the number of bins of level i used by OPT. We can derive the maximum number of bins used by A_1 as a function of the B_i's.

By definition,

$$OPT = B_0 + B_1 + B_2.$$

Since in each I_1-bin OPT could have inserted no more than one I_2-element, the extra bins for A_1 are no more than $\frac{B_1}{2}$. Thus,

$$A_1 \leq B_0 + B_1 + B_2 + \frac{B_1}{2} \leq \frac{3}{2}(B_0 + B_1 + B_2) = \frac{3}{2}\text{OPT}. \qquad \square$$

THEOREM 5.7. *Algorithm* A_1 *has a ratio* $R^{\infty}_{A_1} \leq \frac{3}{2}$.

Proof. The proof derives directly from the previous three lemmas. $\square$

THEOREM 5.8. *Algorithm* A_1 *is* $A(3, 1.5)$.

Proof. The proof derives directly from Corollary 5.2 and Theorem 5.7. $\square$

6. The $O(n \log n)$ Algorithm A_2.

6.1. Main features. A_2 is based on a nonuniform partition of the interval $(0, 1]$ into six subintervals (levels):

$$(0,1] = \bigcup_{k=0}^{5} I_k, \tag{6.1}$$

$$I_0 = \left(\frac{3}{4}, 1\right]; I_1 = \left(\frac{2}{3}, \frac{3}{4}\right]; I_2 = \left(\frac{1}{2}, \frac{2}{3}\right]; I_3 = \left(\frac{1}{3}, \frac{1}{2}\right]; I_4 = \left(\frac{1}{4}, \frac{1}{3}\right]; I_5 = \left(0, \frac{1}{4}\right]. \tag{6.2}$$

The definitions of I_k-element, I_k-bin, A_k-bin, "filled" bin, and gap are similar to the ones given for Algorithm A_1, while the definition of I_5-group is similar to the definition of I_3-group given for Algorithm A_1. Please note that an I_5-group B is "filled" if $gap(B) < \frac{1}{4}$. Thus, A_2 considers I_5-elements as "little" elements which can be collected in groups g_i and moved together. As pointed out in section 3.6, all the grouping primitives are constant in time and space.

6.2. Packing strategy. The algorithm operates as reported in Algorithm A_2, where l denotes the level of the next input element, x. When the input element is a "big" one (i.e., $size(x) > \frac{1}{2}$), A_2 inserts it in a new bin and tries to "fill" its gap with smaller elements from some other bin(s). If the input element is a "small" one (i.e., $size(x) \leq \frac{1}{2}$), the algorithm first tries to insert it in the gap of a filled bin which already exists; only if there is no room for x in any other existing bin, the algorithm inserts it in a new bin. During its execution, A_2 refers to an unfilled bin for each level.[8] The guidelines of the algorithm are the following:

- A_2 encourages the pairing of elements x, y, where $(x \in I_1, y \in I_4)$ or $(x \in I_2, y \in I_3)$.
- A_2 tries to fill the gap of I_1-, I_2-, I_3-filled bins with smaller elements (I_4-elements or I_5-groups) since there are no more I_5-groups or no bin B has $gap(B) > \frac{1}{4}$.

Both of these guidelines may imply the move of a few elements from one bin to another. Bins may be emptied as an effect of element moving: in this case, the emptied bins are considered as automatically disregarded. Finally, please note that the algorithm may return some of the used bins as output before the end of the input list.

[8] As for Algorithm A_1, we do not explicitly consider the management of unfilled bins.

6.3. Data structures. The algorithm requires the use of

- one stack S, containing all the I_5-bins; the unfilled bin is the top one;
- three dictionaries, D_2, D_3, and D_4, maintaining all the I_k-elements contained exclusively in I_k-bins (not necessarily filled), for $k = 2, 3, 4$.
- three dictionaries (tournaments), G_1, G_2, and G_3, maintaining the size of the gap of all the I_1-, I_2-, I_3-filled bins.

We will not give details of the operations involved in inserting and deleting elements to and from the bins and the ones involved in the lists and in I_5-groups maintenance, since they can be easily performed in constant time. Moreover, we will not give details of the operations regarding the tree data structures since they are well known and they may be executed in $O(\log n)$ time. G_1, G_2, and G_3 can be implemented as binary trees of depth $\lceil \log_2 n \rceil$ with n leaves corresponding to the n bins in sequence from left to right. Each internal node is labeled with the largest label among the labels of its sons and each leaf is labeled with the current gap of the bin it represents. Please note that bins containing pairs $x \in I_1$ and $y \in I_4$ or $x \in I_2$ and $y \in I_3$ are immediately returned as output by the algorithm, hence they are not represented in these directories. The tree representation chosen is similar to the one Johnson used to implement the FIRST-FIT algorithm [23]. Last, we will not refer to the possible output of any bin (e.g., either all the I_0-bins or all the bins containing an I_2-element and an I_3-element could be sent to the output).

6.4. Algorithm primitives.

6.4.1. Insert(b,A). This primitive inserts object b, which could be either an item or a group, into bin A and updates, if necessary, one or two of the dictionaries.

Insert (b,A)
• "**Insert** b in A."
• **if** $b \in I_5$ **then** "**Append** b to an existing group or **Create a new group** with only b."
• "**Update**, if necessary, D_2, D_3 or D_4 and G_1, G_2 or G_3"

The updating operation is an $O(\log n)$-time operation and will be carried out only if

- both $(b \in I_k)$ and $(A \in I_k)$ $(k = 2, 3, 4)$ ("enter b in D_k");
- both $(b \in I_k)$ and $(A \in I_k)$ $(k = 1, 2, 3)$ AND A is filled as a consequence of this new element insertion ("enter A in G_k");
- $(A \in I_k)$ and $(b \notin I_k)$ $(k = 1, 2, 3)$ ("Update the size of the gap of A in G_k"). This case occurs only if A is filled and b has to be inserted in its gap.

In conclusion, the Insert operation is $O(\log n)$ worst case time.

6.4.2. Extract(b,A). This primitive extracts object b, which could be either an item or a group, from bin A and updates, if necessary, one or two of the dictionaries.

Extract (b,A)
• "Pop b from A"
• "Update, if necessary, D_2, D_3 or D_4 and G_1, G_2 or G_3"

The updating operation is an $O(\log n)$-time operation and will be carried out only if

- both $(b \in I_k)$ and $(A \in I_k)$ $(k = 2, 3, 4)$ ("extract b from D_k");
- both $(b \in I_k)$ and $(A \in I_k)$ $(k = 1, 2, 3)$, AND A becomes "unfilled" as a consequence of this element extraction ("extract A from G_k");
- $(A \in I_k)$ and $(b \notin I_k)$ $(k = 1, 2, 3)$ ("Update the size of the gap of A in G_k). This case occurs only if A is filled and b has to be extracted from its gap.

In conclusion, the Extract operation is $O(\log n)$ worst case time.

6.4.3. Move(b,B,A). This primitive moves object b from bin B to bin A. It is a composition of Extract(b,B) and Insert (b,A), so it is $O(\log n)$ worst case time.

6.4.4. Fill(C). This primitive fills the gap of the "filled" bin $C \in I_k (1 \leq k \leq 3)$ with smaller elements since there is no room in C (i.e., $gap(C) < \frac{1}{4}$) or there are no more of these little elements. This operation easily is $O(\log n)$ worst case time.

Fill (C)

if "there exists an I_4-bin B containing an element $b \in D_4$ such that $size(b) \leq gap(C)$" **then**
  Move(b,B,C)
  if $B \neq A_4$ **then** Move(x,A_4,B), "for whatever element $x \in A_4$"
else while "there exists $g \in A_5$ such that $size(g) \leq gap(C)$ **AND** $gap(C) \geq \frac{1}{4}$" **do** Move(g,A_5, C)

6.4.5. MoveTheGap(C). This primitive moves all the objects (I_5-groups and eventually the only I_4-element) from the gap of bin C and distributes them among all the other bins. This operation easily is $O(\log n)$ worst case time.

MoveTheGap(C)

if "C contains an I_4-element, b" **then** Move(b,C,A)(where A is, in the sequence of checks, an I_1-bin, an I_2-bin, an I_3-bin or the A_4-bin)
for "every group $g \in C$" **do if** "there exists some I_1, I_2, I_3-bin C' with $size(g) \leq gap(C')$" **then** Move (g,C,C') **else** Move(g,C,A_5)

6.5. The algorithm.

Algorithm A_2

For each element x:
if $x \in I_0$ **then**
- Insert(x,A), where A is a new I_0-bin;

if $x \in I_1$ **then**
- Insert(x,A), where A is a new I_1-bin;
- Fill(A);

if $x \in I_2$ **then**
- Insert(x,A), where A is a new I_2-bin;
- **If** there is a $b \in I_3$ in some I_3-bin B, so that $size(b) + size(x) \leq 1$,
  then * Move (b,B,A); MoveTheGap (B);
  * **if** $B \neq A_3$ **then** "Move(b,A_3,B) for whatever element $b \in A_3$; Fill(B)"
  else * **Fill(A)**;

if $x \in I_3$ **then**
- **If** there is a $b \in I_2$ in some I_2-bin B so that $size(b) + size(x) \leq 1$
  then * **If** $size(x) > gap(B)$ **then** MoveTheGap (B);
  * Insert(x,B);
  else * Insert(x,A_3); **if** A_3 becomes filled, **then** Fill(A_3);

if $x \in I_4$ **then**
- **If** there is an I_1-bin B so that $size(x) \leq gap(B)$, **then** Insert(x,B);
- **else** Insert (b,A) (where A is, in the sequence of checks, an I_2-bin, an I_3-bin or, at last, the A_4-bin).

if $x \in I_5$ **then**
- Create a group g containing only x;
- **If** there is an I_1, I_2, I_3-bin B such that $size(g) \leq gap(B)$ **then** Insert(g,B) **else** Insert(g,A_5)

7. Performance analysis of A_2.

In order to analyze the performance of the algorithm we must first consider the number of element movements caused by the arrival of a new input element and then its asymptotic performance ratio.

7.1. Time, space, and movements.

LEMMA 7.1. *Each filled I_1-bin may contain, in its gap, no more than two I_5-groups.*

Proof. If we assume there are more than two I_5-groups in a filled I_1-bin and let x, y, z be three of them, by definition it follows that

$$x \leq \frac{1}{4}, \quad y \leq \frac{1}{4}, \quad z \leq \frac{1}{4}; \qquad x + y > \frac{1}{4}, \quad x + z > \frac{1}{4}, \quad y + z > \frac{1}{4}.$$

Moreover, an I_1-bin B, has $gap(B) < \frac{1}{3}$, thus $x + y + z < \frac{1}{3}$. Hence

$$\frac{1}{3} > x + y + z > \frac{1}{4} + z \Rightarrow z < \frac{1}{12}.$$

Therefore

$$x > \frac{1}{4} - z > \frac{1}{6}; \qquad y > \frac{1}{4} - z > \frac{1}{6}; \qquad x + y + z > \frac{1}{3},$$

which is a contradiction. ∎

LEMMA 7.2. *Each filled I_2-bin may contain no more than three I_5-groups or one I_4-element plus one I_5-group in its gap.*

Proof. Let us assume there are more than three I_5-groups in a filled I_2-bin. Let x, y, z, w be four of them. By definition of *group* we have that

$$x \leq \frac{1}{4}, \quad y \leq \frac{1}{4}, \quad z \leq \frac{1}{4}, \quad w \leq \frac{1}{4}, \quad x + y > \frac{1}{4}, \quad x + z > \frac{1}{4},$$
$$x + w > \frac{1}{4}, \quad y + z > \frac{1}{4}, \quad y + w > \frac{1}{4}, \quad z + w > \frac{1}{4}.$$

By construction, an I_2-bin, B, has $gap(B) < \frac{1}{2}$, so $x + y + z + w < \frac{1}{2}$.

Therefore

$$\frac{1}{2} > x + y + z + w > \frac{1}{4} + z + w \Rightarrow z + w < \frac{1}{4},$$

which is a contradiction.

Similarly, let us assume that the group contains one I_4-element together with two I_5-groups. Let x be the I_4-element and let y, z be the I_5-groups. By definition, we have that $x \geq \frac{1}{4}$, $y + z \geq \frac{1}{4}$.

By construction, an I_2-bin, B, has $gap(B) < \frac{1}{2}$, so $x + y + z < \frac{1}{2}$.

Therefore

$$\frac{1}{2} > x + y + z > \frac{1}{4} + y + z \Rightarrow y + z < \frac{1}{4},$$

which is a contradiction. ∎

LEMMA 7.3. *Each filled I_3-bin may contain no more than two I_5-groups or one I_4-element plus one I_5-group in its gap.*

Proof. The first bound is proved as in Lemma 7.1.

To prove the second bound let us assume that the group contains one I_4-element and one I_5-group. Let x be the I_4-element and let y, z be the I_5-groups. By definition, we have that $x \geq \frac{1}{4}$, $y + z \geq \frac{1}{4}$.

By construction, an I_3-bin, B, has $gap(B) < \frac{1}{3}$, so $x + y + z < \frac{1}{3}$.

Therefore

$$\frac{1}{3} > x + y + z > \frac{1}{4} + y + z \Rightarrow y + z < \frac{1}{12},$$

which is a contradiction. □

LEMMA 7.4. *Each I_5-bin may contain no more than seven I_5-groups.*

Proof. It follows from the definition of I_5-group. □

The bound of the previous lemma is tight. The sequence matching this bound is $\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}+\epsilon$, when no one of the these elements has to be fitted in any of the I_k-bins ($0 \le k \le 3$) and there is no (unfilled) I_5-bin at the arrival of the first element of the subsequence.

THEOREM 7.5. *In every moment, Algorithm A_2 maintains at most a constant number of I_5-groups and I_4-elements, and no other different level element, in the gap of each bin.*

Proof. It derives from Lemmas 7.1, 7.2, 7.3, and 7.4 and by the fact that I_0-bins contain only I_0-elements and I_4-bins contain only I_4-elements.

The remaining part of the theorem is proved by observing that no I_3-element can be fitted in the gap of any I_0-, I_1-, I_4-, I_5-bin while those fitted in the gap of I_2-bins are immediately returned as output. Note that, for the same reason, no I_4-element is maintained in the gap of any I_1-bin. □

LEMMA 7.6. *When procedure* **Fill** *is called to "fill" an I_k-bin* ($1 \le k \le 3$) *it performs no more than two I_4-element movements or three I_5-group movements.*

Proof. The first bound is simply inferred from the algorithm structure.

The second one is easily obtained by considering that, in every moment, there is only one "unfilled" I_5-bin, named A_5. If $SIZE(A_5) > \frac{1}{4}$ procedure **Fill** moves no more than two groups from it since, whatever is the I_5-bin, any pair of groups has a total size $\ge \frac{1}{4}$, each filled bin has a gap $< \frac{1}{2}$, and the loop ends when $gap(A) < \frac{1}{4}$. If $SIZE(A_5) \le \frac{1}{4}$ then it contains only one group. Let ϵ be its size. If $SIZE(A)+\epsilon < \frac{3}{4}$ and there is another I_5-bin (that is necessarily filled) the situation is the same as the above. Thus, the total number of group movements is three. □

The bound of the previous lemma is tight. In fact let us suppose that the input sequence starts with the following elements: $\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}+\epsilon, \epsilon, \frac{1}{2}+\epsilon$; in this case the first seven elements will be fitted in one I_5-bin, the eighth will be fitted in another I_5-bin and the last will be fitted in an I_2-bin. The **Fill** procedure will move ϵ, then $\frac{1}{8}+\epsilon$, and at last $\frac{1}{8}$ to the gap of the I_2-bin.

THEOREM 7.7. *In correspondence to each insertion, a constant number of element or group movements is performed.*

Proof. Let x be the current input element to be fitted in any bin. Then the algorithm makes

$x \in I_0$ • 0 movements

$x \in I_1$ • no more than three movements (Lemma 7.6) as a consequence of
(two I_4-element movements) OR (three I_5-group movements)

$x \in I_2$ • no more than seven movements[9] as a consequence of
- one I_3-element movement
- two movements (Lemma 7.3) as a consequence of
  (one I_4-element and one I_5-group movement) OR (two I_5-group movements)
- one I_3-element movement
- no more than three movements (Lemma 7.6) as a consequence of
  (two I_4-element movements) OR (three I_5-group movements)

OR, alternatively,

• no more than three movements (Lemma 7.6) as a consequence of

[9]If there exists a $b \in I_3$ such that $size(x) + size(b) \le 1$.

(two I_4-element movements) OR (3 I_5-group movements)

$x \in I_3$
- no more than three movements[10] (Lemma 7.2) as a consequence of (one I_4-element and one I_5-group movements) OR (three I_5-group movements)

OR, alternatively,
- no more than three movements (Lemma 7.6) as a consequence of (two I_4-element movements) OR (three I_5-group movements)

$x \in I_4$
- 0 movements.

$x \in I_5$
- 0 movements.

Therefore, in each case, the total number of element (or group) movements is constant. □

COROLLARY 7.8. *No more than seven element movements occur at the arrival of a new input element when A_2 is applied to a list of n elements.*

Proof. The proof is easily derived from Theorem 7.7. □

THEOREM 7.9. *Algorithm A_2 has space complexity $O(n)$.*

Proof. The theorem is easily proved when observing that each element is represented no more than once in the list S or in the data structures involved and that the maximum number of gaps is n. □

THEOREM 7.10. *Algorithm A_2 has time complexity $O(n \log n)$.*

Proof. According to Theorem 7.7, the time complexity is bounded by n times the cost of an element insertion or movement. Each insertion and each movement of an element already in the structure is performed at most in $O(\log n)$ time, when the tree data structures are involved. In fact the directory representation for the gaps of I_1-, I_2-, I_3-bins allows the finding of the right element in no more than $\lceil \log_2 n \rceil$ binary comparisons and the update operations may be performed by using no more than $\lceil \log_2 n \rceil$ comparisons [23]. This same principle is valid for the heaps maintaining the I_3-and I_4-elements [33]. □

7.2. Performance ratio. In the following items we assume that

- N_j is the total number of I_j-elements in the input list L.
- H is the size of the best matching between I_2-and I_3-elements, that is, the maximum number of pairs $x_2 \in I_2$, $x_3 \in I_3$ which can be coupled. Note that this corresponds to the maximum matching in a bipartite graph $G = (N_2 \cup N_3, E)$ so that
  - $N_2 = \{x \in I_2\}$,
  - $N_3 = \{x \in I_3\}$,
  - $E = \{(x, y) | x \in N_2, y \in N_3, size(x) + size(y) \leq 1\}$.
- K is the size of the best matching between I_1-and I_4-elements, that is, the maximum number of pairs $x_1 \in I_1$, $x_4 \in I_4$ which can be coupled. Please note that this corresponds to the maximum matching in a bipartite graph $G = (N_1 \cup N_4, E)$ so that
  - $N_1 = \{x \in I_1\}$,
  - $N_4 = \{x \in I_4\}$,
  - $E = \{(x, y) | x \in N_1, y \in N_4, size(x) + size(y) \leq 1\}$.

LEMMA 7.11. *If S is not empty after all the elements have been considered, then $R_{A_2} < \frac{4}{3}$.*

Proof. By hypothesis, since there is at least one group with size $\leq \frac{1}{4}$ which cannot be moved to a different gap, all of the I_0-, I_1-, I_2-, I_3-bins have a gap $< \frac{1}{4}$.

[10] If there exists a $b \in I_2$ such that $size(x) + size(b) \leq 1$.

Moreover, each I_k-bin ($k \geq 4$) has a gap $< \frac{1}{4}$.
Consequently, the maximum gap in each bin is $< \frac{1}{4}$, and

$$R_{A_2} < \frac{4}{3}. \qquad \square$$

LEMMA 7.12. *If $H = 0$ and S is empty after all the elements have been considered, then $R_{A_2} \leq \frac{4}{3}$.*

Proof. We can observe two different situations:

- In the input list there is no I_4-element.
  In this case A_2 uses no more than $N_0+N_1+N_2+\frac{N_3}{2}$ bins, since no I_2-element or I_3-element could be inserted in any bin already containing an I_1-element or an I_0-element and no more than two I_3-elements may be inserted in any other bin. OPT cannot use fewer bins, since $H = 0$ implies that no bin can contain both an I_2-element and an I_3-element. Therefore

$$R_{A_2} = 1.$$

- In the input list there were some I_4-elements.
  Let $\alpha = N_1 + N_2 + \frac{N_3}{2}$. In this case we can still obtain two situations:
  $N_4 \leq \alpha$ then
  - OPT uses at least $\alpha + N_0$ bins.
  - A_2 uses no more than $N_0+\alpha+\frac{N_4}{3}$ bins, in case it does not insert any I_4-element into some other bin. Then $A_2 \leq N_0 + \alpha + \frac{\alpha}{3} = N_0 + \frac{4}{3}\alpha$ and so

$$R_{A_2} \leq \frac{\frac{4}{3}\alpha + N_0}{\alpha + N_0} < \frac{4}{3}.$$

  $N_4 > \alpha$ then let $\beta = N_4 - \alpha$.
  - OPT cannot use fewer than $N_0 + \alpha + \frac{\beta}{3}$ bins, since no more than one I_4-element can fit into an I_1-, I_2-, or I_3-bin.
  - A_2 uses no more than $N_0+\alpha+\frac{N_4}{3}$ bins. Then $A_2 \leq N_0+\alpha+\frac{\alpha+\beta}{3} = N_0 + \frac{4}{3}\alpha + \frac{\beta}{3}$. Therefore we have

$$\frac{A_2}{OPT} \leq \frac{\frac{4}{3}\alpha + \frac{\beta}{3} + N_0}{\alpha + \frac{\beta}{3} + N_0} < \frac{4}{3}. \qquad \square$$

LEMMA 7.13. *If $H \neq 0$ and in the input sequence $L = \{a_1, a_2, \ldots, a_n\}$ there are no I_k-elements ($k \geq 4$), then $R_{A_2} < \frac{5}{4}$.*

Proof. OPT uses at least
N_0 bins to pack all the I_0-elements;
N_1 bins to pack all the I_1-elements;
N_2 bins to pack all the I_2-elements;
$\frac{N_3-H}{2}$ bins to pack all the I_3-elements, since H of them are inserted in the gap of the H I_2-elements.

Hence,

$$OPT \geq N_0 + N_1 + N_2 + \frac{N_3}{2} - \frac{H}{2}.$$

A_2 uses no more than

N_0 bins to pack all the I_0-elements;
N_1 bins to pack all the I_1-elements;
N_2 bins to pack all the I_2-elements;
$\frac{N_3 - \frac{H}{2}}{2}$ bins to pack all the I_3-elements, since $\frac{H}{2}$ of them are necessarily packed with a corresponding I_2-element [22].

Hence

$$A_2 \leq N_0 + N_1 + N_2 + \frac{N_3}{2} - \frac{H}{4}.$$

In conclusion, since $H \leq N_3$ and $H \leq N_2$, it follows that

$$\frac{A_2}{OPT} \leq \frac{4N_0 + 4N_1 + 4N_2 + 2N_3 - H}{4N_0 + 4N_1 + 4N_2 + 2N_3 - 2H} \leq \frac{4N_0 + 4N_1 + 5N_2 + 2N_3 - 2H}{4N_0 + 4N_1 + 4N_2 + 2N_3 - 2H} < \frac{5}{4}. \qquad \square$$

LEMMA 7.14. *If $H \neq 0$ and in the input sequence $L = \{a_1, a_2, \ldots, a_n\}$ there are some I_k-elements ($k \geq 4$), and S is empty after all elements have been considered, then $R_{A_2} \leq \frac{4}{3}$.*

Proof. Let B_i be the number of bins of level i used by OPT and let B_i' be the number of bins of level i used by A_2. We calculate the maximum number of bins used by A_2 as a function of the B_i's.

We can obtain two different situations:

- In the case that no I_4-bins are returned as output, since in such a case there are no I_5- and I_4-bins, we can say that $OPT = B_0 + B_1 + B_2 + B_3$:
  $B_0' = B_0$;
  $B_1' = B_1$;
  $B_2' = B_2$;
  $B_3' \leq B_3 + \frac{\frac{H}{2}}{2}$;
  Therefore, since $H \leq B_2$,

$$A_2 \leq B_0 + B_1 + \frac{5}{4}B_2 + \frac{5}{4}B_3 \leq \frac{5}{4}\text{OPT}.$$

- In the case that some I_4-bins are returned as output, since in such a case there are no I_5-bins, we can say that $OPT = B_0 + B_1 + B_2 + B_3 + B_4$:
  $B_0' = B_0$;
  $B_1' = B_1$;
  $B_2' = B_2$;
  $B_3' \leq B_3 + \frac{\frac{H}{2}}{2}$;
  $B_4' = B_4 + [K + (B_2 - H) + B_3]\frac{1}{3} = \frac{B_2}{3} + \frac{B_3}{3} + B_4 + \frac{K}{3} - \frac{H}{3}$.
  Therefore, since $K \leq B_1$ and $H \geq 0$,

$$\begin{aligned} A_2 &\leq B_0 + B_1 + \tfrac{4}{3}B_2 + \tfrac{4}{3}B_3 + B_4 + \tfrac{k}{3} - \tfrac{H}{12} \\ &\leq B_0 + \tfrac{4}{3}B_1 + \tfrac{4}{3}B_2 + \tfrac{4}{3}B_3 + B_4 \leq \tfrac{4}{3}\text{ OPT}. \end{aligned} \qquad \square$$

THEOREM 7.15. *Algorithm A_2 has a ratio of $R_{A_2} \leq \frac{4}{3}$.*

Proof. The proof is easily given by the previous lemmas. $\square$

THEOREM 7.16. *Algorithm A_2 is A $(7, 1.33)$.*

Proof. The proof is given by Corollary 7.8 and Theorem 7.15. $\square$

8. Conclusions and open problems. This paper focuses its attention on the possibility of maintaining a guaranteed approximation of the optimal solution for the online bin-packing problem in terms of time computation and element movements and with a limited reorganization of the previous solutions.

The problem is equivalent to the one considered in the more general bin-packing model where the elements can move (for a limited number of times) from the bin they are currently assigned to. Please note that this model still fits the general definition of online algorithms.

It would be interesting to see if these algorithms frequently touch any particular element and move it many times: it is also possible to demonstrate that the first algorithm (A_1) moves an element no more than once.

Contrarily to the offline model, the requests arriving to this model reach it online and the bins are always ready to be closed with no additional effort. This model is suitable in many different fields (for example in multiprocessor storage management and in packing trucks).

In the environment of this less restricted model, we have presented two new algorithms with the best approximation ratio available at this time, respectively, for linear and $O(n \log n)$ algorithms.

These algorithms are also "more" online than all the other linear space online bin-packing algorithms, because they allow the return of a fraction of the bins before the end of the execution.

There are still a lot of problems which remain to be solved in this area. First, it would be interesting to check if algorithms more efficient than A_2 can be found (as far as approximation ratio and/or time complexity are concerned). Generally speaking, it would also be interesting to define the lower bounds of the approximation ratio of the $O(n \log n)$ algorithms that allow the element to freely move between the bins.

It would also be interesting to verify if there is some kind of relation between the (amortized) number of movements allowed at the arrival of each input element and the asymptotic performance ratio. In other words: is there an algorithm that for each ϵ constant is $A(\frac{1}{\epsilon}, f(\epsilon))$?

It would also be useful to gain more knowledge about whether the approximation ratio can be maintained and if it is possible to send output the bins containing "enough" of the elements contained therein (e.g., what happens when all the bins with gap less than $\frac{k}{3}$ $(k \leq 1)$ are send output?).

Other interesting questions concern the capacity of deleting elements from the bins maintaining the guaranteed algorithm performance (the target is to minimize the space wasted considering the actual element involved) and the analytical evaluation of the average performance of this algorithm compared to the ones characterized in [27], [30], [29], [32]. It would be very interesting to study the performances of the algorithm as a function of the number of movements admitted.

Acknowledgments. We are grateful to two anonymous referees for their useful remarks and to Giuseppe Alesio and Gennaro Gravina for useful discussions.

We also wish to thank Lucio Bianco of CNR IASI, Rome, for suggesting the transportation applications.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] J. L. Bentley, D. S. Johnson, F. T. Leighton, and C. C. McGeoch, *An experimental study of bin packing*, in Proceedings of the 21st Annual Allerton Conference on Communication, Control, and Computing, University of Illinois, Urbana, IL, 1983, pp. 51–60.

[3] J. L. Bentley, D. S. Johnson, F. T. Leighton, C. C. McGeoch, and L. A. McGeoch, *Some unexpected expected behavior results for bin-packing*, in Proceedings of the 16th ACM Symposium on the Theory of Computing, Washington, DC, 1984, pp. 279–288.

[4] A. Borodin, N. Linial, and M. Saks, *An optimal online algorithm for metrical task systems*, in Proceedings of the 19th ACM Symposium on the Theory of Computing, New York, 1987, pp. 373–382.

[5] D. J. Brown, *An improved BL lower bound*, Inform. Process Lett., 11 (1980), pp. 37–39.

[6] D. J. Brown and P. Ramanan, *Online bin packing in linear time*, in Proceedings of the 1984 Conference on Information Sciences and Systems, Princeton University, Princeton, NJ, 1984, pp. 328–332.

[7] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin-packing—An updated survey*, in Algorithm Design for Computer System Design, G. Ausiello and M. Lucertini, eds., Springer-Verlag, New York, 1984, pp. 49–106.

[8] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin-packing: A survey*, in Approximation Algorithms of NP-Hard Problems, D. S. Hochbaum ed., PWS Publishing Company, Boston, 1996, pp. 46–93.

[9] E. G. Coffman, Jr., M. Hofri, K. So, and A. C. Yao, *A stochastic model of bin packing*, Informat. Control, 44 (1980), pp. 105–115.

[10] J. Csirik and D. S. Johnson, *Bounded space online bin-packing: Best is better than First*, in Proceedings of the Second ACM–SIAM Symposium on Discrete Algorithms, San Francisco,1991, pp. 309–319.

[11] G. Gambosi, A. Postiglione, and M. Talamo, *On the Online Bin-Packing Problem*, IASI Technical Report R.263, Roma, 1989.

[12] G. Gambosi, A. Postiglione, and M. Talamo, *New algorithms for online bin-packing*, in Algorithms and Complexity, G. Ausiello, D. Bovet, and R. Petreschi, eds., World Scientific Publishing, Roma, 1990, pp. 44–59.

[13] G. Gambosi, A. Postiglione, and M. Talamo, *Online maintenance of an approximate bin-packing solution*, Nordic J. Comput., 4 (1997), pp. 151–166.

[14] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao, *Resource constrained scheduling as generalized bin-packing*, J. Combin. Theory Ser. A, 21 (1976), pp. 257–298.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, CA, 1979.

[16] P. C. Gilmore and R. E. Gomory, *A linear programming approach to the cutting stock problem*, Oper. Res., 9 (1961), pp. 849–859.

[17] P. C. Gilmore and R. E. Gomory, *A linear programming approach to the cutting stock problem—Part* II, Oper. Res., 11 (1963), pp. 863–888.

[18] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1581.

[19] R. L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

[20] N. Karmarkar and R. M. Karp, *An efficient approximation scheme for the one-dimensional bin-packing problem*, in Proceedings of the 23th IEEE Symposium on Foundation of Computer Science, Chicago, 1982, pp. 312–320.

[21] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. M. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–105.

[22] B. Korte and D. Hausmann, *Matroids and independence systems*, in Modern Applied Mathematics: Optimization and Operations Research, B. Korte, ed., North-Holland, Amsterdam, 1982, pp. 517–553.

[23] D. S. Johnson, *Fast algorithms for bin-packing*, J. Comput. System Sci., 8 (1974), pp. 272–314.

[24] 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 J. Comput., 3 (1974), pp. 299–325.

[25] C. C. Lee and D. T. Lee, *A simple online bin-packing algorithm*, J. ACM, 3 (1985), pp. 562–572.

[26] F. M. Liang, *A lower bound for on-line bin packing*, Inform. Process. Lett., 10 (1980), pp. 76–79.

[27] R. Loulou, *Probabilistic behavior of optimal bin-packing solutions*, Oper. Res. Lett., 3 (1984), pp. 129–135.

[28] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, *Competitive algorithms for server problems*, J. Algorithms, 11 (1990), pp. 208–230.

[29] P. Ramanan and K. Tsuga, K., *Average-case of the modified harmonic algorithm*, Algorithmica, 4 (1989), pp. 519–533.

[30] P. Ramanan, *Average-case analysis of the SMART-NEXT-FIT algorithm*, Inform. Process. Lett., 31 (1989), pp. 221–225.

[31] P. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee, *On-line bin-packing in linear time*, J. Algorithms, 10 (1989), pp. 305–326.

[32] P. W. Shor, *The average-case of some on-line algorithms for bin-packing*, Combinatorica, 6 (1986), pp. 179–200.

[33] R. E. Tarjan, *Data Structures and Network Algorithms*, CMBS-NSF Regional Conf. Ser. in Appl. Math. 44, SIAM, Philadelphia, 1983.

[34] A. C. Yao, *New algorithms for bin-packing*, J. ACM, 27 (1980), pp. 207–227.