SIAM J. COMPUT.
Vol. 32, No. 2, pp. 455–469

NEW BOUNDS FOR VARIABLE-SIZED ONLINE BIN PACKING*

STEVEN S. SEIDEN†, ROB VAN STEE‡, AND LEAH EPSTEIN§

The remaining authors would like to dedicate this paper to the memory of our friend Steve Seiden, who was killed in an accident on June 11, 2002

Abstract. In the variable-sized online bin packing problem, one has to assign items to bins one by one. The bins are drawn from some fixed set of sizes, and the goal is to minimize the sum of the sizes of the bins used. We present new algorithms for this problem and show upper bounds for them which improve on the best previous upper bounds. We also show the first general lower bounds for this problem. The case in which bins of two sizes, 1 and $\alpha \in (0, 1)$, are used is studied in detail. This investigation leads us to the discovery of several interesting fractal-like curves.

Key words. bin packing, online algorithms

AMS subject classifications. 68Q25, 68W25, 68W40

PII. S0097539702412908

1. Introduction.

In this paper, we investigate the bin packing problem, one of the oldest and most thoroughly studied problems in computer science [3, 5]. In particular, we investigate a natural generalization of the classical online bin packing problem known as online variable-sized bin packing. We show improved upper bounds and the first lower bounds for this problem and in the process encounter several strange fractal-like curves.

Problem definition. In the *classical bin packing* problem, we receive a sequence σ of *pieces* $p_1, p_2, \ldots, p_N$. Each piece has a fixed *size* in $(0, 1]$. In a slight abuse of notation, we use p_i to indicate both the ith piece and its size. We have an infinite number of *bins* each with *capacity* 1. Each piece must be assigned to a bin. Further, the sum of the sizes of the pieces assigned to any bin may not exceed its capacity. A bin is *empty* if no piece is assigned to it; otherwise, it is *used*. The goal is to minimize the number of bins used.

The *variable-sized bin packing* problem differs from the classical one in that the bins do not all have the same capacity. There are an infinite number of bins of each capacity $\alpha_1 < \alpha_2 < \cdots < \alpha_m = 1$. The goal now is to minimize the sum of the capacities of the bins used.

In the *online* versions of these problems, each piece must be assigned in turn, without knowledge of the next pieces. Since it is impossible in general to produce the best possible solution when computation occurs online, we consider approximation

*Received by the editors August 14, 2002; accepted for publication (in revised form) October 1, 2002; published electronically February 4, 2003. A preliminary version of this paper appeared in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, 2002, pp. 306–317.

http://www.siam.org/journals/sicomp/32-2/41290.html

†The author is deceased. Former address: Department of Computer Science, 298 Coates Hall, Louisiana State University, Baton Rouge, LA 70803. This author's research was supported by the Louisiana Board of Regents Research Competitiveness Subprogram.

‡Institut für Informatik, Albert-Ludwigs-Universität, Georges-Köhler-Allee, 79110 Freiburg, Germany (vanstee@informatik.uni-freiburg.de). This work was done while the author was at the CWI, The Netherlands. This author's research was supported by the Netherlands Organization for Scientific Research (NWO), project SION 612-30-002.

§School of Computer Science, The Interdisciplinary Center, Herzliya, Israel (lea@idc.ac.il). This author's research was supported by Israel Science Foundation grant 250/01.

algorithms. Basically, we want to find an algorithm which incurs cost which is within a constant factor of the minimum possible cost, no matter what the input is. This constant factor is known as the asymptotic performance ratio.

A bin packing algorithm uses *bounded space* if it has only a constant number of bins available to accept items at any point during processing. These bins are called *open* bins. Bins which have already accepted some items, but which the algorithm no longer considers for packing, are *closed* bins. While bounded space algorithms are sometimes desirable, it is often the case that unbounded space algorithms can achieve lower performance ratios.

We define the asymptotic performance ratio more precisely. For a given input sequence σ, let $\text{cost}_{\mathcal{A}}(\sigma)$ be the sum of the capacities of the bins used by algorithm $\mathcal{A}$ on σ. Let $\text{cost}(\sigma)$ be the minimum possible cost to pack pieces in σ. The *asymptotic performance ratio* for an algorithm $\mathcal{A}$ is defined to be

$$R_{\mathcal{A}}^{\infty} = \limsup_{n\to\infty} \max_{\sigma} \left\{ \frac{\text{cost}_{\mathcal{A}}(\sigma)}{\text{cost}(\sigma)} \;\middle|\; \text{cost}(\sigma) = n \right\}.$$

The *optimal asymptotic performance ratio* is defined to be

$$R_{\text{OPT}}^{\infty} = \inf_{\mathcal{A}} R_{\mathcal{A}}^{\infty}.$$

Our goal is to find an algorithm with asymptotic performance ratio close to R_{OPT}^{∞}.

Previous results. The online bin packing problem was first investigated by Johnson [9, 10]. He showed that the NEXT FIT algorithm has performance ratio 2. Subsequently, it was shown by Johnson et al. that the FIRST FIT algorithm has performance ratio $\frac{17}{10}$ [11]. Yao showed that REVISED FIRST FIT has performance ratio $\frac{5}{3}$ and further showed that no online algorithm has performance ratio less than $\frac{3}{2}$ [21]. Brown [1] and Liang [14] independently improved this lower bound to 1.53635. This was subsequently improved by van Vliet to 1.54014 [19]. Chandra [2] shows that the preceding lower bounds also apply to randomized algorithms.

Define

$$u_{i+1} = u_i(u_i - 1) + 1, \qquad u_1 = 2,$$

and

$$h_{\infty} = \sum_{i=1}^{\infty} \frac{1}{u_i - 1} \approx 1.69103.$$

Lee and Lee showed that the HARMONIC algorithm, which uses bounded space, achieves a performance ratio arbitrarily close to h_{∞} [13]. They further showed that no bounded space online algorithm achieves a performance ratio less than h_{∞} [13]. A sequence of further results has brought the upper bound down to 1.58889 [13, 15, 16, 17].

The variable-sized bin packing problem was first investigated by Friesen and Langston [7, 8]. Kinnersley and Langston gave an online algorithm with performance ratio $\frac{7}{4}$ [12]. Csirik proposed the VARIABLE HARMONIC algorithm and showed that it has performance ratio at most h_{∞} [4]. This algorithm is based on the HARMONIC algorithm of Lee and Lee [13]. Like HARMONIC, it uses bounded space. Csirik also showed that if the algorithm has two bin sizes 1 and $\alpha < 1$ and if it is allowed to pick α, then a performance ratio of $\frac{7}{5}$ is possible [4]. Seiden has recently shown that VARIABLE HARMONIC is an optimal bounded-space algorithm [18].

The related problem of variable-sized bin covering has been solved by Woeginger and Zhang [20] and extended by Epstein [6].

Our results. In this paper, we present new algorithms for the variable-sized online bin packing problem. By combining the upper bounds for these algorithms, we improve the upper bound for this problem from 1.69103 to 1.63597. Our technique extends the general packing algorithm analysis technique developed by Seiden [17]. We also show the first lower bounds for variable-sized online bin packing. We focus on the case in which there are two bin sizes. However, our techniques are applicable to the general case. We think that our results are particularly interesting because of the unusual fractal-like curves that arise in the investigation of our algorithms and lower bounds.

2. Upper bounds. To begin, we present two different online algorithms for variable-sized bin packing.

We focus in on the case in which there are two bin sizes, $\alpha_1 < 1$ and $\alpha_2 = 1$, and examine how the performance ratios of our algorithms change as a function of α_1. Since it is understood that $m = 2$, we abbreviate α_1 using α. Both of our algorithms are combinations of the HARMONIC and REFINED HARMONIC algorithms. Both have a real parameter $\mu \in (\frac{1}{3}, \frac{1}{2})$. We call these algorithms VRH1(μ) and VRH2(μ). VRH1(μ) is defined for all $\alpha \in (0, 1)$, but VRH2(μ) is defined only for

$$\alpha > \max\left\{\frac{1}{2(1-\mu)}, \frac{1}{3\mu}\right\}. \tag{2.1}$$

First, we describe VRH1(μ). Define $n_1 = 50$, $n_2 = \lfloor n_1\alpha \rfloor$, $\epsilon = 1/n_1$, and

$$T = \left\{\frac{1}{i} \;\middle|\; 1 \le i \le n_1\right\} \cup \left\{\frac{\alpha}{i} \;\middle|\; 1 \le i \le n_2\right\} \cup \{\mu, 1-\mu\}.$$

Define $n = |T|$. Note that it may be that $n < n_1 + n_2 + 2$ since T is not a multiset. Rename the members of T as $t_1 = 1 > t_2 > t_3 > \cdots > t_n = \epsilon$. For convenience, define $t_{n+1} = 0$. The interval I_j is defined to be $(t_{j+1}, t_j]$ for $j = 1, \ldots, n+1$. Note that these intervals are disjoint and that they cover $(0, 1]$. A piece of size s has *type* j if $s \in I_j$. Define the *class* of an interval I_j to be α if $t_j = \alpha/k$ for some positive integer k; otherwise, the class is 1.

The basic idea of VRH1 is as follows: When each piece arrives, we determine the interval I_j to which it belongs. If this is a class 1 interval, we pack the item in a size 1 bin using a variant of REFINED HARMONIC. If it is a class α interval, we pack the item in a size α bin using a variant of HARMONIC.

VRH1 packs bins in *groups*. All the bins in a group are packed in a similar fashion. The groups are determined by the set T. We define

$$g = \begin{cases} 3 & \text{if } \alpha > 1-\mu, \\ 2 & \text{otherwise.} \end{cases} \qquad h = \begin{cases} 6 & \text{if } \alpha/2 > \mu, \\ 5 & \text{if } \alpha > \mu \text{ and } \alpha/2 \le \mu, \\ 4 & \text{otherwise.} \end{cases}$$

Note that these functions are defined so that $t_g = 1 - \mu$ and $t_h = \mu$. The groups are named $(g, h), 1, \ldots, g-1, g+1, g+2, \ldots, n$.

Bins in group $j \in \{1, 2, \ldots, n\} \setminus \{g\}$ contain only type j pieces.

Bins in group (g, h) all have capacity 1. Closed bins contain one type g piece and one type h piece.

```
VRH1
        Initialize x ← 0 and y ← 0.
        For each item p:
            j ← type of p.
            If j = n then pack p using NEXT FIT in a group n bin.
            Else, if j = g then PUT(p, (g, h)).
            Else, if j = h:
                x ← x + 1.
                If y < ⌊τx⌋:
                    y ← y + 1.
                    PUT(p, (g, h)).
                Else PUT(p, h).
            Else PUT(p, j).
```

```
PUT(p, G)
        If there is no open bin in G then allocate a new bin b.
        Else, let b be an arbitrary open bin in G.
        Pack p in b.
```

FIG. 2.1. *The* VRH1(μ) *algorithm and the* PUT *subroutine.*

Bins in group n all have capacity 1 and are packed using the NEXT FIT algorithm. There is always one open bin in group n. When a type n piece arrives, if the piece fits in the open bin, it is placed there. If not, the open bin is closed, the piece is placed in a newly allocated open group n bin.

For group $j \in \{1, 2, \ldots, n-1\} \setminus \{g\}$, the capacity of bins in the group depends on the class of I_j. If I_j has class 1, then each bin has capacity 1, and each closed bin contains $\lfloor 1/t_j \rfloor$ items of type j. Note that t_j is the reciprocal of an integer for $j \neq h$, and therefore $\lfloor 1/t_j \rfloor = 1/t_j$. If I_j has class α, then each bin has capacity α, and each closed bin contains $\lfloor \alpha/t_j \rfloor$ items of type j. Similarly to before, t_j/α is the reciprocal of an integer, and therefore $\lfloor \alpha/t_j \rfloor = \alpha/t_j$. For each of these groups, there is at most one open bin.

The algorithm has a real parameter $\tau \in [0, 1]$, which for now we fix to be $\frac{1}{7}$. Essentially, a proportion τ of the type h items are reserved for placement with type g items.

A precise definition of VRH1 appears in Figure 2.1. The algorithm uses the subroutine PUT(p, G), where p is an item and G is a group.

We analyze VRH1 using the technique of *weighting systems* introduced in [17]. A weighting system is a tuple $(\mathbb{R}^\ell, \mathbf{w}, \xi)$, where $\mathbb{R}^\ell$ is a real vector space, $\mathbf{w}$ is a *weighting function*, and ξ is a *consolidation function*. We shall simply describe the weighting system for VRH1 and assure the reader that our definitions meet the requirements put forth in [17].

For VRH1, we use $\ell = 3$ and define $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ to be orthogonal unit basis vectors.

The weighting function is

$$\mathbf{w}(x) = \begin{cases} \mathbf{b} & \text{if } x \in I_g, \\ (1-\tau)\dfrac{\mathbf{a}}{2} + \tau\,\mathbf{c} & \text{if } x \in I_h, \\ \dfrac{\mathbf{a}\,x}{1-\epsilon} & \text{if } x \in I_n, \\ \mathbf{a}\,t_i & \text{otherwise.} \end{cases}$$

The consolidation function is $\xi(x\,\mathbf{a} + y\,\mathbf{b} + z\,\mathbf{c}) = x + \max\{y, z\}$. The following lemma allows us to upper bound the performance of VRH1 using the preceding weighting system.

LEMMA 2.1. *For all input sequences σ,*

$$\text{cost}_{\text{VRH1}}(\sigma) \le \xi\left(\sum_{i=1}^{n} \mathbf{w}(p_i)\right) + O(1).$$

Proof. We count the cost for bins in each group.

First, consider bins in group n. Each of these is packed using NEXT FIT and contains only pieces of size at most ϵ. By the definition of NEXT FIT, each closed bin contains items of total size at least $1-\epsilon$, and there is at most one open bin. Therefore, the number of bins used is at most

$$\frac{1}{1-\epsilon}\sum_{p_i \in I_n} p_i + 1 = \mathbf{a} \cdot \sum_{p_i \in I_n} \mathbf{w}(p_i) + O(1).$$

Now consider group j with $j \notin \{h, (g,h), n\}$. There is at most one open bin in this group. The capacity x of each bin is equal to the class of I_j. The number of items in each closed bin is $\lfloor x/t_j \rfloor$. Since $j \notin \{h, (g,h), n\}$, we have $\lfloor x/t_j \rfloor = x/t_j$. Putting these facts together, the cost is at most

$$\sum_{p_i \in I_j} \frac{x}{\lfloor x/t_j \rfloor} + 1 = \sum_{p_i \in I_j} t_j + 1 = \mathbf{a} \cdot \sum_{p_i \in I_j} \mathbf{w}(p_i) + O(1).$$

Next, consider group h. Let k be the number of type h items in σ. The algorithm clearly maintains the invariant that $\lfloor \tau k \rfloor$ of these items go to group (g,h). The remainder are packed two to a bin in capacity 1 bins. At most one bin in group h is open. The total is at most

$$\frac{k - \lfloor \tau k \rfloor}{2} + 1 = \sum_{p_i \in I_h} \frac{1-\tau}{2} + O(1) = \mathbf{a} \cdot \sum_{p_i \in I_h} \mathbf{w}(p_i) + O(1).$$

Finally, consider group (g,h). Let f be the number of type g items in σ. The number of bins is

$$\max\{f, \lfloor \tau k \rfloor\} = \max\{f, \tau k\} + O(1) = \max\left\{\mathbf{b} \cdot \sum_{p_i \in I_g} \mathbf{w}(p_i),\ \mathbf{c} \cdot \sum_{p_i \in I_h} \mathbf{w}(p_i)\right\} + O(1).$$

Putting all these results together, the total cost is at most

$$\mathbf{a} \cdot \sum_{i=1}^{n} \mathbf{w}(p_i) + \max\left\{\mathbf{b} \cdot \sum_{i=1}^{n} \mathbf{w}(p_i),\ \mathbf{c} \cdot \sum_{i=1}^{n} \mathbf{w}(p_i)\right\} + O(1) = \xi\left(\sum_{i=1}^{n} \mathbf{w}(p_i)\right) + O(1). \qquad \square$$

From [17], we also have the following lemma.

LEMMA 2.2. *For any input σ on which* VRH1 *achieves a performance ratio of c, there exists an input σ' where* VRH1 *achieves a performance ratio of at least c and*

1. *every bin in an optimal solution is full, and*
2. *every bin in some optimal solution is packed identically.*

Given these two lemmas, the problem of upper bounding the performance ratio of VRH1 is reduced to that of finding the single packing of an optimal bin with maximal weight/size ratio. We consider the following integer program: Maximize $\xi(\mathbf{x})/\beta$ subject to

$$\mathbf{x} = \mathbf{w}(y) + \sum_{j=1}^{n-1} q_j \mathbf{w}(t_j); \tag{2.2}$$

$$y = \beta - \sum_{j=1}^{n-1} q_j \, t_{j+1}, \tag{2.3}$$

$$y > 0, \tag{2.4}$$

$$q_j \in \mathbb{N} \qquad \text{for } 1 \le j \le n-1, \tag{2.5}$$

$$\beta \in \{1, \alpha\}, \tag{2.6}$$

over variables $\mathbf{x}, y, \beta, q_1, \ldots, q_{n-1}$. Intuitively, q_j is the number of type j pieces in an optimal bin. y is an upper bound on space available for type n pieces. Note that strict inequality is required in (2.4) because a type j piece is strictly larger than t_{j+1}. Call this integer linear program $\mathcal{P}$. The value of $\mathcal{P}$ upper bounds the asymptotic performance ratio of VRH1.

The value of $\mathcal{P}$ is easily determined using a branch and bound procedure very similar to those in [17, 18]. Define

$$\psi_i = \max\left\{(\mathbf{a}+\mathbf{b}+\mathbf{c})\cdot\mathbf{w}(t_i),\ \frac{1}{1-\epsilon}\right\} \qquad \text{for } 1 \le i \le n-1; \qquad \psi_n = \frac{1}{1-\epsilon}.$$

Intuitively, ψ_i is the maximum contribution to the objective function for a type i item relative to its size. We define π so that

$$\psi_{\pi(1)} \ge \psi_{\pi(2)} \ge \cdots \ge \psi_{\pi(n)}.$$

The procedure is displayed in Figure 2.2. The heart of the procedure is the subroutine TRYALL, which basically finds the maximum weight which can be packed into a bin of size β. Using π, we try first to include items which contribute the most to the objective relative to their size. This is a heuristic. The variables $\mathbf{v}$ and y keep track of the weight and total size of items included so far. The variable j indicates that the current item type is $\pi(j)$. In the For loop at the end of TRYALL, we try each possible number of type $\pi(j)$ items, starting with the largest possible number. First packing as many items as possible is a heuristic which seems to speed up computation. The current maximum is stored in x. When we enter TRYALL, we first compute an upper bound given the packing so far, which is stored in z. When $j = n$, this upper bound is exactly the objective value. If $z \le x$, we do not have to consider any packing reachable from the current one, and we drop straight through. In the main routine, we simply initialize x, call TRYALL for the two bin sizes, and return x.

Now we describe VRH2(μ). Redefine

$$T = \left\{\frac{1}{i} \;\middle|\; 1 \le i \le n_1\right\} \cup \left\{\frac{\alpha}{i} \;\middle|\; 1 \le i \le n_2\right\} \cup \{\alpha\mu,\ \alpha(1-\mu)\}.$$

```
x ← 1.
TRYALL(1, 0, 1, 1).
TRYALL(1, 0, α, α).
Return x.
```

```
TRYALL(j, v, y, β)
    z ← (ξ(v) + y ψ_{π(j)}) /β.
    If z > x then:
    If j = n then:
        x ← z.
    Else:
        For i ← ⌈y/t_{π(j)+1}⌉ − 1, ..., 0:
            TRYALL(j + 1, v + i w(t_{π(j)}), y − i t_{π(j)+1}, β).
```

FIG. 2.2. *The algorithm for computing $\mathcal{P}$ along with subroutine* TRYALL.

Define n_1, n_2, ϵ, and n as for VRH1. Again, rename the members of T as $t_1 = 1 > t_2 > t_3 > \cdots > t_n = \epsilon$. Equation (2.1) guarantees that $1/2 < \alpha(1-\mu) < \alpha < 1$ and $1/3 < \alpha\mu < \alpha/2 < 1/2$, so we have $g = 3$ and $h = 6$. The only difference from VRH1 is that (g, h) bins have capacity α. Otherwise, the two algorithms are identical. We therefore omit a detailed description and analysis of VRH2.

We display the upper bound on the performance ratio achieved using the best of VRH1(μ), VRH2(μ), and VARIABLE HARMONIC in Figure 4.3. This upper bound is achieved by optimizing μ for each choice of α. Our upper bound is at most $\frac{373}{228} < 1.63597$ for all α, which is the performance ratio of REFINED HARMONIC in the classic bin packing context.

3. Lower bounds. We now consider the question of lower bounds. Prior to this work, no general lower bounds for variable-sized online bin packing were known.

Our method follows that of Brown [1], Liang [14], and van Vliet [19]. We give some unknown online bin packing algorithm $\mathcal{A}$ one of k possible different inputs. These inputs are defined as follows: Let $\varrho = s_1, s_2, \ldots, s_k$ be a sequence of *item sizes* such that $0 < s_1 < s_2 < \cdots < s_k \leq 1$. Let ϵ be a small positive constant. We define σ_0 to be the empty input. Input σ_i consists of σ_{i-1} followed by n items of size $s_i + \epsilon$. Algorithm $\mathcal{A}$ is given σ_i for some $i \in \{1, \ldots, k\}$.

A *pattern* with respect to ϱ is a tuple $p = \langle \text{size}(p), p_1, \ldots, p_k \rangle$, where size($p$) is a positive real number and p_i, $1 \leq i \leq k$, are nonnegative integers such that

$$\sum_{i=1}^{k} p_i \, s_i < \text{size}(p).$$

Intuitively, a pattern describes the contents of some bin of capacity size(p). Define $\mathcal{P}(\varrho, \beta)$ to be the set of all patterns p with respect to ϱ with size(p) = β. Further define

$$\mathcal{P}(\varrho) = \bigcup_{i=1}^{m} \mathcal{P}(\varrho, \alpha_i).$$

Note that $\mathcal{P}(\varrho)$ is necessarily finite. Given an input sequence of items, an algorithm is defined by the numbers and types of items it places in each of the bins it uses.

Specifically, any algorithm is defined by a function $\Phi : \mathcal{P}(\varrho) \mapsto \mathbb{R}_{\geq 0}$. The algorithm uses $\Phi(p)$ bins containing items as described by the pattern p. We define $\phi(p) = \Phi(p)/n$.

Consider the function Φ that determines the packing used by online algorithm $\mathcal{A}$ for σ_k. Since $\mathcal{A}$ is online, the packings it uses for $\sigma_1, \ldots, \sigma_{k-1}$ are completely determined by Φ. We assign to each pattern a *class*, which is defined as

$$\text{class}(p) = \min\{i \mid p_i \neq 0\}.$$

Intuitively, the class tells us the first sequence σ_i, which results in some item being placed into a bin packed according to this pattern. That is, if the algorithm packs some bins according to a pattern which has class i, then these bins will contain one or more items after σ_i. Define

$$\mathcal{P}_i(\varrho) = \{p \in \mathcal{P}(\varrho) \mid \text{class}(p) \leq i\}.$$

Then, if $\mathcal{A}$ is determined by Φ, its cost for σ_i is simply

$$n \sum_{p \in \mathcal{P}_i(\varrho)} \text{size}(p)\phi(p).$$

Since the algorithm must pack every item, we have the following constraints:

$$n \sum_{p \in \mathcal{P}(\varrho)} \phi(p)\, p_i \geq n \qquad \text{for } 1 \leq i \leq k.$$

For a fixed n, define $\chi_i(n)$ to be the optimal offline cost for packing the items in σ_i. The following lemma gives us a method of computing the optimal offline cost for each sequence.

LEMMA 3.1. *For $1 \leq i \leq k$, $\chi^* = \lim_{n\to\infty} \chi_i(n)/n$ exists and is the value of the following linear program: Minimize*

$$\sum_{p \in \mathcal{P}_i(\varrho)} \text{size}(p)\phi(p) \tag{3.1}$$

subject to

$$1 \quad \leq \quad \sum_{p \in \mathcal{P}(\varrho)} \phi(p)\, p_j \qquad \text{for } 1 \leq j \leq i \tag{3.2}$$

over variables χ_i and $\phi(p)$, $p \in \mathcal{P}(\varrho)$.

Proof. Clearly, the linear program always has a finite value between $\sum_{j=1}^{i} s_j$ and i. For any fixed n, the optimal offline solution is determined by some ϕ. It must satisfy the constraints of the linear program, and the objective value is exactly the cost incurred. Therefore, the linear program lower bounds the optimal offline cost. The linear program is a relaxation in that it allows a fractional number of bins of any pattern, whereas a legitimate solution must have an integral number. Rounding the relaxed solution up to get a legitimate one, the change in the objective value is at most $|\mathcal{P}(\varrho)|/n$. □

Given the construction of a sequence, we need to evaluate

$$c = \min_{\mathcal{A}} \max_{i=1,\ldots,k} \limsup_{n\to\infty} \frac{\text{cost}_{\mathcal{A}}(\sigma_i)}{\chi_i(n)}.$$

As $n \to \infty$, we can replace $\chi_i(n)/n$ by χ_i^*. Once we have the values $\chi_1^*, \ldots, \chi_k^*$, we can readily compute a lower bound for our online algorithm.

LEMMA 3.2. *The optimal value of the linear program: Minimize c subject to*

$$\begin{aligned} c &\geq \frac{1}{\chi_i^*} \sum_{p \in \mathcal{P}_i(\varrho)} \text{size}(p)\phi(p) \qquad \text{for } 1 \leq i \leq k, \\ 1 &\leq \sum_{p \in \mathcal{P}(\varrho)} \phi(p)\, p_i \qquad \text{for } 1 \leq i \leq k \end{aligned} \tag{3.3}$$

over variables c and $\phi(p)$, $p \in \mathcal{P}(\varrho)$, is a lower bound on the asymptotic performance ratio of any online bin packing algorithm.

Proof. For any fixed n, any algorithm $\mathcal{A}$ has some Φ which must satisfy the second constraint. Further, Φ should assign an integral number of bins to each pattern. However, this integrality constraint is relaxed, and $\sum_{p \in \mathcal{P}_i(\varrho)} \text{size}(p)\phi(p)$ is $1/n$ times the cost to $\mathcal{A}$ for σ_i as $n \to \infty$. The value of c is just the maximum of the performance ratios achieved on $\sigma_1, \ldots, \sigma_k$. ∎

Although this is essentially the result we seek, a number of issues are left to be resolved.

The first is that these linear programs have a variable for each possible pattern. The number of such patterns is potentially quite large, and we would like to reduce the linear program size if possible. We show that this goal is indeed achievable. We say that a pattern p of class i is *dominant* if

$$s_i + \sum_{j=1}^{k} p_j\, s_j > \text{size}(p).$$

Let p be a nondominant pattern with class i. There exists a unique dominant pattern q of class i such that $p_j = q_j$ for all $i \neq j$. We call q the *dominator* of p with respect to class i.

LEMMA 3.3. *In computing the values of the linear programs in Lemmas* 3.1 *and* 3.2*, it suffices to consider only dominant patterns.*

Proof. We transform a linear program solution by applying the following operation to each nondominant pattern p of class i: Let $x = \phi(p)$ in the original solution. We set $\phi(p) = 0$ and increment $\phi(q)$ by x, where q is the dominator of p with respect to i. The new solution remains feasible, and its objective value has not changed. Further, the value of $\phi(p)$ is zero for every nondominant p; therefore, these variables can be safely deleted. ∎

Given a sequence of item sizes ϱ, we can compute a lower bound $L_m(\varrho, \alpha_1, \ldots, \alpha_{m-1})$ using the following algorithm:

1. Enumerate the dominant patterns.
2. For $1 \leq i \leq k$, compute χ_i via the linear program given in Lemma 3.1.
3. Compute and return the value of the linear program given in Lemma 3.2.

Step 1 is most easily accomplished via a simple recursive function. Our concern in the remainder of the paper shall be to study the behavior of $L_m(\varrho, \alpha_1, \ldots, \alpha_{m-1})$ as a function of ϱ and $\alpha_1, \ldots, \alpha_{m-1}$.

4. Lower bound sequences. Up to this point, we have assumed that we were given some fixed item sequence ϱ. We consider now the question of choosing ϱ. We again focus on the case in which there are two bin sizes and examine properties of $L_2(\varrho, \alpha_1)$. We again abbreviate α_1 using α and L_2 using L.

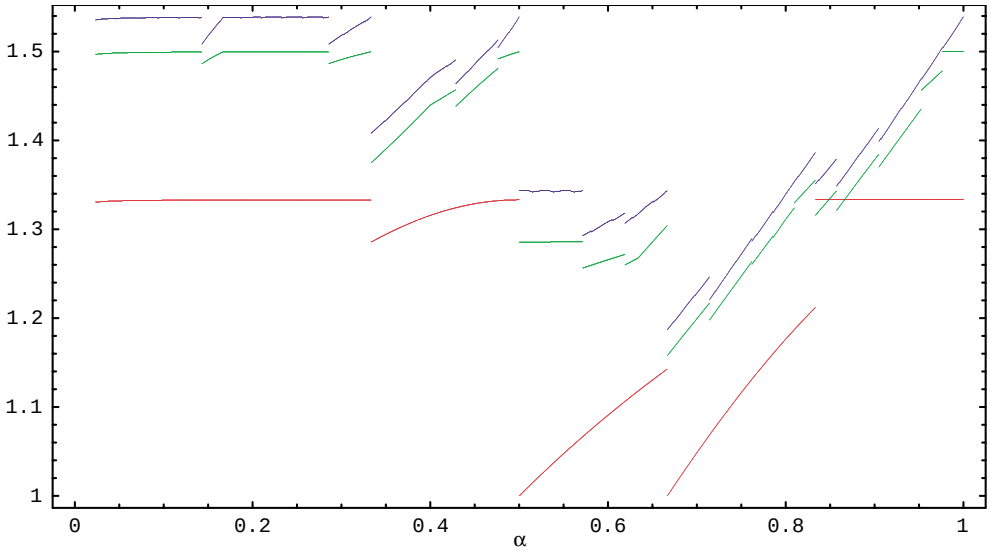

FIG. 4.1. *The evolution of the curves given by the greedy item sequence. The lowest curve is* $\frac{1}{2}, \frac{1}{3}$; *the middle curve is* $\frac{1}{2}, \frac{1}{3}, \frac{1}{7}$; *the highest curve is* $\frac{1}{2}, \frac{1}{3}, \frac{1}{7}, \frac{1}{43}$.

To begin, we define the idea of a *greedy* sequence. Let ϵ denote the empty sequence, and let $\wedge$ denote the sequence concatenation operator. The greedy sequence $\Gamma_\tau(\beta)$ for capacity β with cutoff τ is defined by

$$\gamma(\beta) = \frac{1}{\lfloor \frac{1}{\beta} \rfloor + 1}, \qquad \Gamma_\tau(\beta) = \begin{cases} \epsilon & \text{if } \beta < \tau, \\ \gamma(\beta) \wedge \Gamma_\tau(\beta - \gamma(\beta)) & \text{otherwise.} \end{cases}$$

The sequence defines the item sizes which would be used if we packed a bin of capacity β using the following procedure: At each step, we determine the remaining capacity in our bin. We choose as the next item the largest reciprocal of an integer which fits without using the remaining capacity completely. We stop when the remaining capacity is smaller than τ. Note that, for $\tau = 0$, we get the infinite sequence. We shall use Γ as a shorthand for Γ_0.

The recurrence u_i described in section 1, which is found in connection with bounded-space bin packing [13], gives rise to the sequence

$$\frac{1}{u_i} = \frac{1}{2}, \frac{1}{3}, \frac{1}{7}, \frac{1}{43}, \frac{1}{1807}, \ldots.$$

This turns out to be the infinite greedy sequence $\Gamma(1)$. Somewhat surprisingly, it is also the sequence used by Brown [1], Liang [14], and van Vliet [19] in the construction of their lower bounds. In essence, they analytically determine the value of $L_1(\Gamma_\tau(1))$. Liang and Brown lower bound the value, while van Vliet determines it exactly.

This well-known sequence is our first candidate. Actually, we use the first k item sizes in it, and we resort them so that the algorithm is confronted with items from smallest to largest. In general, this resorting seems to be a good heuristic since the algorithm has the most decisions to make about how the smallest items are packed but, on the other hand, has the least information about which further items will be received. The results are shown in Figure 4.1.

Examining Figure 4.1, one immediately notices that $L(\Gamma_\tau(1), \alpha)$ exhibits some very strange behavior. The curve is highly discontinuous. Suppose we have a finite sequence ϱ, where each item size is a continuous function of $\alpha \in (0,1)$. Tuple p is a *potential pattern* if there exists an $\alpha \in (0,1)$ such that p is a pattern. The set of breakpoints of p with respect to ϱ is defined to be

$$B(p,\varrho) = \left\{ \alpha \in (0,1) \;\middle|\; \sum_{i=1}^{k} p_i\, s_i = \text{size}(p) \right\}.$$

Let $\mathcal{P}^*$ be the set of all potential patterns. The set of all breakpoints is

$$B(\varrho) = \bigcup_{p \in \mathcal{P}^*} B(p,\varrho).$$

Intuitively, at each breakpoint, some combinatorial change occurs, and the curve may jump. In the intervals between breakpoints, the curve behaves nicely as summarized by the following lemma.

LEMMA 4.1. *Let ϱ be a finite item sequence with each item size a continuous function of $\alpha \in (0,1)$. In any interval $I = (\ell, h)$ which does not contain a breakpoint, $L(\varrho,\alpha)$ is continuous. Furthermore, for all $\alpha \in I$,*

$$L(\varrho,\alpha) \geq \min\left\{ \frac{\ell+h}{2h}, \frac{2\ell}{\ell+h} \right\} L\left(\varrho, \tfrac{1}{2}(\ell+h)\right).$$

This lemma follows as a corollary from the following lemma.

LEMMA 4.2. *Let ϱ be a finite item sequence with each item size a continuous function of $\alpha \in (0,1)$. Let I be any interval which does not contain a breakpoint, and let α be any point in I. The following two results hold:*

1. *If $\delta > 0$ is such that $\alpha + \delta \in I$, then*

$$L(\varrho,\alpha+\delta) \geq \left(1 - \frac{\delta}{\alpha+\delta}\right) L(\varrho,\alpha).$$

2. *If $\delta > 0$ is such that $\alpha - \delta \in I$, then*

$$L(\varrho,\alpha-\delta) \geq \left(1 - \frac{\delta}{\alpha}\right) L(\varrho,\alpha).$$

Proof. We first prove statement 1. Denote by $\chi_i^*(x)$ the value of χ_i^* at $\alpha = x$. For $1 \leq i \leq k$, we have

$$\chi_i^*(\alpha+\delta) \leq \frac{\alpha+\delta}{\alpha} \chi_i^*(\alpha).$$

To see this, note that any feasible Φ at α is also feasible at $\alpha+\delta$ since both points are within I and (3.2) does not change within this interval. Each term in (3.1) increases by at most $(\alpha+\delta)/\alpha$. Now consider the linear program of Lemma 3.2. Consider some arbitrary feasible solution ϕ at α. At $\alpha+\delta$, this solution is still feasible (except that possibly c must increase). In the sum $1/\chi_i^* \sum_{p \in \mathcal{P}_i(\varrho)} \text{size}(p)\phi(p)$, the factor $1/\chi_i^*$ decreases by at most $\alpha/(\alpha+\delta)$, and size(p) cannot decrease.

Now consider statement 2. The arguments are quite similar. For $1 \leq i \leq k$, we have

$$\chi_i^*(\alpha-\delta) \leq \chi_i^*(\alpha).$$

Again, a feasible solution remains feasible. Further, its objective value (3.1) cannot increase. Considering the linear program of Lemma 3.2, we find that, for each feasible solution, each sum $1/\chi_i^* \sum_{p\in\mathcal{P}_i(\varrho)} \text{size}(p)\phi(p)$ decreases by a factor of at most $(\alpha-\delta)/\alpha$. □

Considering Figure 4.1 again, there are sharp drops in the lower bound near the points $\frac{1}{3}$, $\frac{1}{2}$, and $\frac{2}{3}$. It is not hard to see why the bound drops so sharply at those points. For instance, if α is just larger than $\frac{1}{2}+\epsilon$, then the largest items in $\Gamma(1)$ can each be put in their own bin of size α. If $\alpha \geq \frac{2}{3}+2\epsilon$, two items of size $\frac{1}{3}+\epsilon$ can be put pairwise in bins of size α. In short, in such cases, the online algorithm can pack some of the largest elements in the list with very little wasted space—hence the low resulting bound.

This observation leads us to try other sequences in which the last items cannot be packed well. A first candidate is the sequence $\alpha, \Gamma(1-\alpha)$. As expected, this sequence performs much better than $\Gamma(1)$ in the areas described above.

It is possible to find further improvements for certain values of α. For instance, the sequence $\alpha/2, \Gamma(1-\alpha/2)$ also works well in some places, and we used other sequences as well. We give two examples in Figure 4.2.

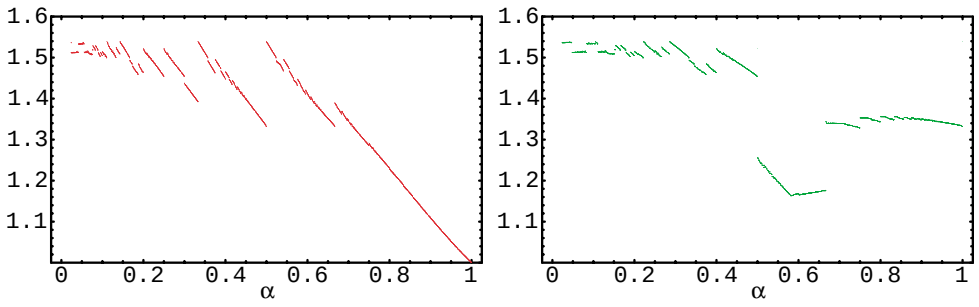

FIG. 4.2. *Two lower bound sequences for* $\tau = 1/1000$: *On the left is* $\alpha, \Gamma_\tau(1-\alpha)$, *and on the right is* $\frac{\alpha}{2}, \Gamma_\tau(1-\frac{\alpha}{2})$.

As a general guideline for finding sequences, items should not fit too well in either bin size. If an item has size x, then $\min\{1 - \lfloor \frac{1}{x} \rfloor x,\ \alpha - \lfloor \frac{\alpha}{x} \rfloor x\}$ should be as large as possible. In areas where a certain item in a sequence fits very well, that item should be adjusted (e.g., use an item $1/(j+1)$ instead of the item $1/j$), or a completely different sequence should be used. (This helps explain why the algorithms have a low competitive ratio for α close to 0.7: in that area, this minimum is never very large.)

Furthermore, as in the classical bin packing problem, sequences that are bad for the online algorithm should have very different optimal solutions for each prefix sequence. Finally, the item sizes should not increase too quickly or too slowly: If items are very small, the smallest items do not affect the online performance much, while if items are close in size, the sequence is easy because the optimal solutions for the prefixes are alike.

In addition to the three sequences already described, namely, the greedy sequence, $\alpha, \Gamma(1-\alpha)$, and $\alpha/2, \Gamma(1-\alpha/2)$, we have found that the following sequences yield good results in restricted areas: $\alpha, \frac{1}{3}, \frac{1}{7}, \frac{1}{43}$; $\frac{1}{2}, \frac{1}{4}, \frac{1}{5}, \frac{1}{21}$; and $\frac{1}{2}, \frac{\alpha}{2}, \frac{1}{9}, \Gamma_\tau(\frac{7}{18} - \frac{\alpha}{2})$.

Using Lemma 4.2, we obtain the following main theorem of this section.

THEOREM 4.3. *Any online algorithm for the variable-sized bin packing problem with* $m = 2$ *has asymptotic performance ratio at least* $495176908800/370749511199 > 1.33561$.

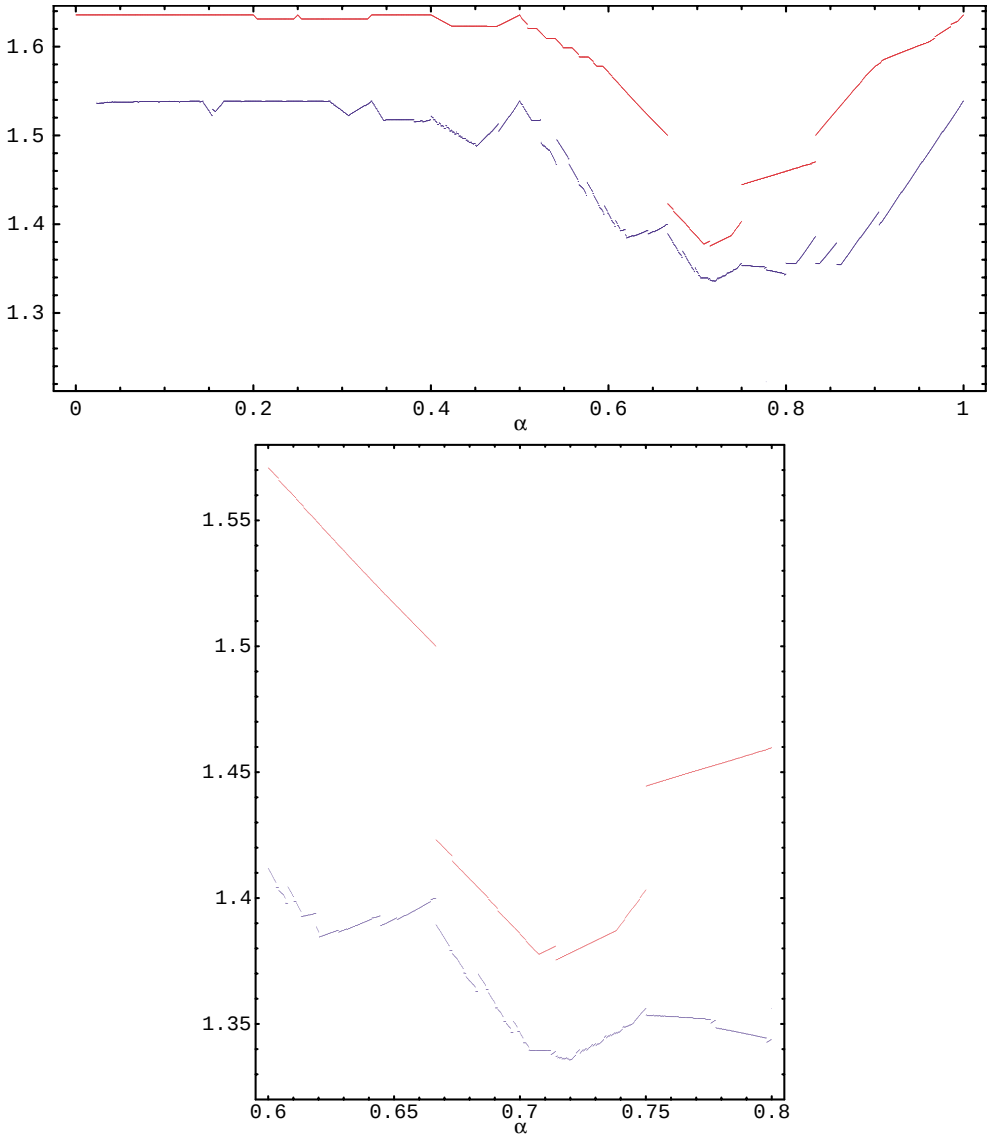

FIG. 4.3. *The best upper and lower bounds for variable-sized online bin packing. The bottom figure is a closeup of* $[.6, .8]$. *The upper bound is best of the* VRH1, VRH2, *and* VARIABLE HARMONIC *algorithms.*

Proof. First, note that, for $\alpha \in (0, 1/43]$, the sequence $\frac{1}{2}, \frac{1}{3}, \frac{1}{7}, \frac{1}{43}$ yields a lower bound of $217/141 > 1.53900$ as in the classic problem: Bins of size α are of no use.

We use the sequences described in the preceding paragraphs. For each sequence ϱ, we compute a lower bound on $(1/43, 1)$ using the following procedure.

Define $\varepsilon = 1/10000$. We break the interval $(0, 1)$ into subintervals using the lattice points $\varepsilon, 2\varepsilon, \ldots, 1 - \varepsilon$. To simplify the determination of breakpoints, we use a constant sequence for each subinterval. This constant sequence is fixed at the upper limit of the interval. That is, throughout the interval $[\ell\varepsilon, \ell\varepsilon + \varepsilon)$, we use the sequence $\varrho|_{\alpha=\ell\varepsilon+\varepsilon}$. Since the sequence is constant, a lower bound on the performance ratio of any online bin packing algorithm with $\alpha \in [\ell\varepsilon, \ell\varepsilon + \varepsilon)$ can be determined by the following algorithm:

1. $\varrho' \leftarrow \varrho|_{\alpha=\ell\varepsilon+\varepsilon}$.
2. Initialize $B \leftarrow \{\ell\varepsilon, \ell\varepsilon + \varepsilon\}$.
3. Enumerate all the patterns for ϱ' at $\alpha = \ell\varepsilon + \varepsilon$.
4. For each pattern:
   (a) $z \leftarrow \sum_{i=1}^{k} p_i\, s_i$.
   (b) If $z \in (\ell\varepsilon, \ell\varepsilon + \varepsilon)$, then $B \leftarrow B \cup \{z\}$.
5. Sort B to get $b_1, b_2, \ldots, b_j$.
6. Calculate and return the value:

$$\min_{1 \le i < j} \min \left\{ \frac{b_i + b_{i+1}}{2b_{i+1}},\ \frac{2b_i}{b_i + b_{i+1}} \right\} L\left(\varrho',\, \tfrac{1}{2}(b_i + b_{i+1})\right).$$

We implemented this algorithm in `Mathematica` and used it to find lower bounds for each of the aforementioned sequences. The results are shown in Figures 4.2 and 4.3. The lowest lower bound is 495176908800/370749511199 in the interval $[0.7196, 0.7197)$. □

5. Conclusions. We have shown new algorithms and lower bounds for variable-sized online bin packing with two bin sizes. By combining these algorithms with Variable Harmonic, choosing for each size α of the second bin the best algorithm for that size, we find an algorithm with asymptotic performance ratio of at most $\frac{373}{228} < 1.63597$ for all α. The best previous upper bound was $h_\infty \approx 1.69103$.

The largest gap between the performance of the algorithm and the lower bound is 0.18193 achieved for $\alpha = 0.9071$. The smallest gap is 0.03371 achieved for $\alpha = 0.6667$. Note that, for $\alpha \le \frac{1}{2}$, there is not much differing from the classical problem: having the extra bin size does not help the online algorithm much. To be more precise, it helps about as much as it helps the offline algorithm.

Our work raises the following questions: Is there a value of α where it is possible to design a better algorithm and show a matching lower bound? Or, can a lower bound be shown anywhere that matches an existing algorithm? Note that at the moment there is also a small gap between the competitive ratio of the best algorithm and the lower bound in the classical bin packing problem.

Another interesting open problem is analyzing variable-sized bin packing with an arbitrary number of bin sizes.

REFERENCES

[1] D. J. Brown, *A Lower Bound for On-Line One-Dimensional Bin Packing Algorithms*, Tech. report R-864, Coordinated Science Laboratory, Urbana, IL, 1979.

[2] B. Chandra, *Does randomization help in on-line bin packing?*, Inform. Process. Lett., 43 (1992), pp. 15–19.

[3] E. G. Coffman, M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin packing: A survey*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS Publishing, Boston, 1997, Chap. 2.
[4] J. Csirik, *An on-line algorithm for variable-sized bin packing*, Acta Inform., 26 (1989), pp. 697–709.
[5] J. Csirik and G. Woeginger, *On-line packing and covering problems*, in On-Line Algorithms—the State of the Art, Lecture Notes in Comput. Sci. 1442, A. Fiat and G. Woeginger, eds., Springer-Verlag, New York, 1998, pp. 147–177.
[6] L. Epstein, *On-line variable sized covering*, Inform. and Comput., 171 (2001), pp. 294–305.
[7] D. K. Friesen and M. A. Langston, *A storage-size selection problem*, Inform. Process. Lett., 18 (1984), pp. 295–296.
[8] D. K. Friesen and M. A. Langston, *Variable sized bin packing*, SIAM J. Comput., 15 (1986), pp. 222–230.
[9] D. S. Johnson, *Near-Optimal Bin Packing Algorithms*, Ph.D. thesis, MIT, Cambridge, MA, 1973.
[10] D. S. Johnson, *Fast algorithms for bin packing*, J. Comput. System Sci., 8 (1974), pp. 272–314.
[11] 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.
[12] N. Kinnersley and M. Langston, *Online variable-sized bin packing*, Discrete Appl. Math., 22 (1989), pp. 143–148.
[13] C. Lee and D. Lee, *A simple on-line bin-packing algorithm*, J. ACM, 32 (1985), pp. 562–572.
[14] F. M. Liang, *A lower bound for online bin packing*, Inform. Process. Lett., 10 (1980), pp. 76–79.
[15] P. Ramanan, D. Brown, C. Lee, and D. Lee, *On-line bin packing in linear time*, J. Algorithms, 10 (1989), pp. 305–326.
[16] M. B. Richey, *Improved bounds for harmonic-based bin packing algorithms*, Discrete Appl. Math., 34 (1991), pp. 203–227.
[17] S. S. Seiden, *On the online bin packing problem*, in Proceedings of the 28th International Colloquium on Automata, Languages and Programming, Crete, Greece, 2001, pp. 237–249.
[18] S. S. Seiden, *An optimal online algorithm for bounded space variable-sized bin packing*, SIAM J. Discrete Math., 14 (2001), pp. 458–470.
[19] A. van Vliet, *An improved lower bound for online bin packing algorithms*, Inform. Process. Lett., 43 (1992), pp. 277–284.
[20] G. Woeginger and G. Zhang, *Optimal on-line algorithms for variable-sized bin covering*, Oper. Res. Lett., 25 (1999), pp. 47–50.
[21] A. C. C. Yao, *New algorithms for bin packing*, J. ACM, 27 (1980), pp. 207–227.