Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

On the Fractal Beauty of Bin Packing

L. Epstein, S. Seiden, R. van Stee

Software Engineering (SEN)

SEN-R0104 February 28, 2001

Report SEN-R0104
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

On the Fractal Beauty of Bin Packing

Leah Epstein

School of Computer and Media Sciences, The Interdisciplinary Center, P.O. Box 167, 46150 Herzliya, Israel.

Epstein.Leah@idc.ac.il.

Steve Seiden

Department of Computer Science, 298 Coates Hall, Louisiana State University, Baton Rouge, LA 70803, U.S.A.

sseiden@acm.org.

Rob van Stee*

CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands.

Rob.van.Stee@cwi.nl.

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 the first unbounded space algorithms for this problem. We also show the first lower bounds on the asymptotic performance ratio. The case where 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.

2000 Mathematics Subject Classification: 68Q25, 68W25, 68W40

1998 ACM Computing Classification System: F.2.2

Keywords and Phrases: bin packing, variable-sized bins, on-line algorithms, competitive analysis

Note: Work carried out under theme SEN4 “Evolutionary Systems and Applied Algorithmics”.

1. Introduction

In this paper we investigate the bin packing problem, one of the oldest and most well studied problems in computer science [3, 5]. The influence and importance of this problem are witnessed by the fact that it has spawned off whole areas of research, including the fields of online algorithms and approximation algorithms. 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. The usage should be obvious from the context. 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.

*Research supported by the Netherlands Organization for Scientific Research (NWO), project number SION 612-30-002.

The *variable-sized bin packing* problem differs from the classical one in that 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 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.

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

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

The *optimal asymptotic performance ratio* is defined to be

$$R^\infty_{\mathrm{OPT}} = \inf_{\mathcal{A}} R^\infty_{\mathcal{A}}.$$

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 [8, 9]. He showed that the Next Fit algorithm has performance ratio 2. Subsequently, it was shown by Johnson, Demers, Ullman, Garey and Graham that the First Fit algorithm has performance ratio $\frac{17}{10}$[10]. Yao showed that Revised First Fit has performance ratio $\frac{5}{3}$, and further showed that no online algorithm has performance ratio less than $\frac{3}{2}$ [20]. Brown and Liang independently improved this lower bound to 1.53635 [1, 13]. This was subsequently improved by van Vliet to 1.54014 [18]. 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_∞ [12]. They further showed that no bounded space online algorithm achieves a performance ratio less than h_∞ [12]. A sequence of further results has brought the upper bound down to 1.58889 [12, 14, 15, 16].

The variable-sized bin packing problem was first investigated by Frieson and Langston [6, 7]. Kinnerly and Langston gave an online algorithm with performance ratio $\frac{7}{4}$ [11]. 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 [12]. Like Harmonic, it uses bounded space. Csirik also showed that if the algorithm has two bin sizes 1 and $\alpha < 1$, and that 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 [17].

The related problem of variable-sized bin covering has be solved by Woeginger and Zhang [19].

Our Results: In this paper, we present two algorithms for the variable-sized online bin packing problem. These algorithms have the best known performance for many sets of bin sizes. We also show the first lower bounds. 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. *Color versions of the figures can be found at* `http://www.csc.lsu.edu/~seiden/figs`.

2. Two Algorithms

To begin, we present two different unbounded space online algorithms for variable-sized bin packing.

We focus in on the case where 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. We present two algorithms, both of which are combinations of the Variable 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 only defined 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 multi-set. 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.

We differentiate between bins which are *open* and bins which are *closed*. An open bin is a non-empty bin into which the algorithm may potentially place one or more other pieces. The algorithm does not ever put a piece into a bin which is closed.

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.

Bins in group n all have capacity 1 and are packed using the NEXT FIT algorithm. I.e. there is 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 one, 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. Similar 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 1. The algorithm uses the sub-routine PUT(p, G),

```
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.
```

Figure 1: The VRH1(μ) algorithm and the PUT sub-routine.

where p is an item and G is a group.

We analyze VRH1 using the technique of *weighting systems* introduced in [16]. 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 [16].

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.

Consider first 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 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 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

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

■

From [16], we also have

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

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

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

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

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

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

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 [16, 17]. Define

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

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

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

The procedure is displayed in Figure 2. The heart of the procedure is the sub-routine 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 $z \leq x$, 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 bins sizes, and return x. We display the upper bound achieved by VRH1(μ) for several values of μ in Figure 3. By optimizing τ for each choice of α and μ, it is possible to improve the algorithm's performance. However, for simplicity's sake, we keep $\tau = \frac{1}{7}$ in this paper.

```
        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).
```

Figure 2: The algorithm for computing $\mathcal{P}$, along with sub-routine TRYALL.

Now we describe VRH2(μ). Redefine

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

Define n_1, n_2, ϵ and n as for VRH1. Again, rename the members of T as $t_1 = 1 > t_2 > t_3 > \cdots > t_n = \epsilon$. (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 performance ratio of VRH2(μ) for several values of μ in Figure 3.

3. Lower Bounds

We now consider the question of lower bounds. Prior to this work, no general lower bounds were known.

Our method follows along the lines laid down by Liang, Brown and van Vliet [1, 13, 18]. 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 $0 < s_1 < s_2 < \cdots < s_k \le 1$. 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 $p_i, 1 \le i \le k$ are non-negative 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

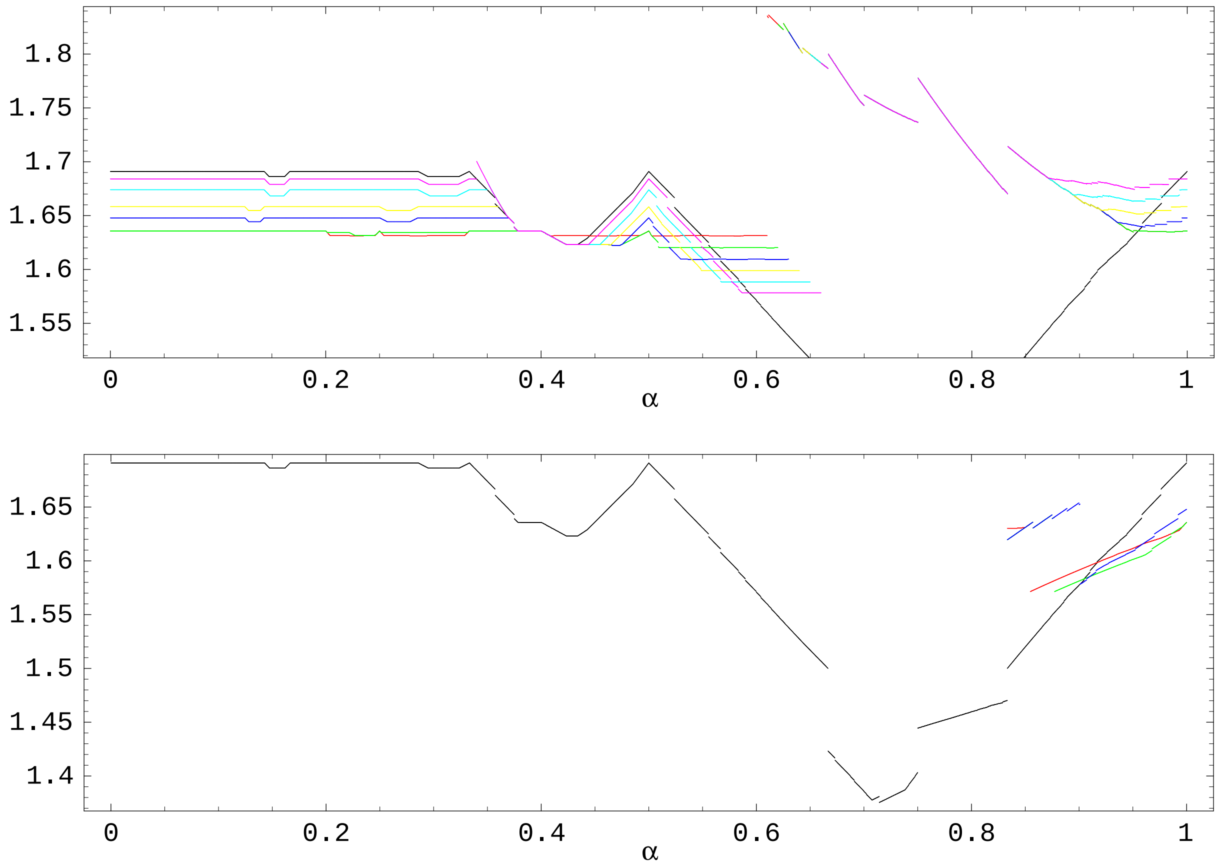

Figure 3: Upper bounds for variable sized bin packing. In both figures, VARIABLE HARMONIC is shown in black. In the top figure, we display VRH1(.61) in red, VRH1(.62) in blue, VRH1(.63) in green, VRH1(.64) in yellow, VRH1(.65) in light blue and VRH1(.66) in purple. In the bottom figure, we display VRH2(.60) in red, VRH2(.61) in blue and VRH2(.62) in green.

the set of all patterns p with respect to ϱ with $\text{size}(p) = \beta$. 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 $\mathcal{A}$ uses 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

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

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. I.e. 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 linear program: Minimize*

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

subject to

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

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

Proof Clearly, the LP 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 LP, and the objective value is exactly the cost incurred. Therefore the LP lower bounds the optimal offline cost. The LP 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*

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

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 pattern is potentially quite large, and we would like to reduce it 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 non-dominant 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 an LP solution by applying the following operation to each non-dominant 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 non-dominant 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 LP given in Lemma 3.1.
3. Compute and return the value of the LP given in Lemma 3.2.

Step one 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. What's in a sequence?

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 in on the case where there are two bin sizes, and examine properties of $L_2(\varrho, \alpha_1)$. We abbreviate α_1 using α and L_2 using L.

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

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

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

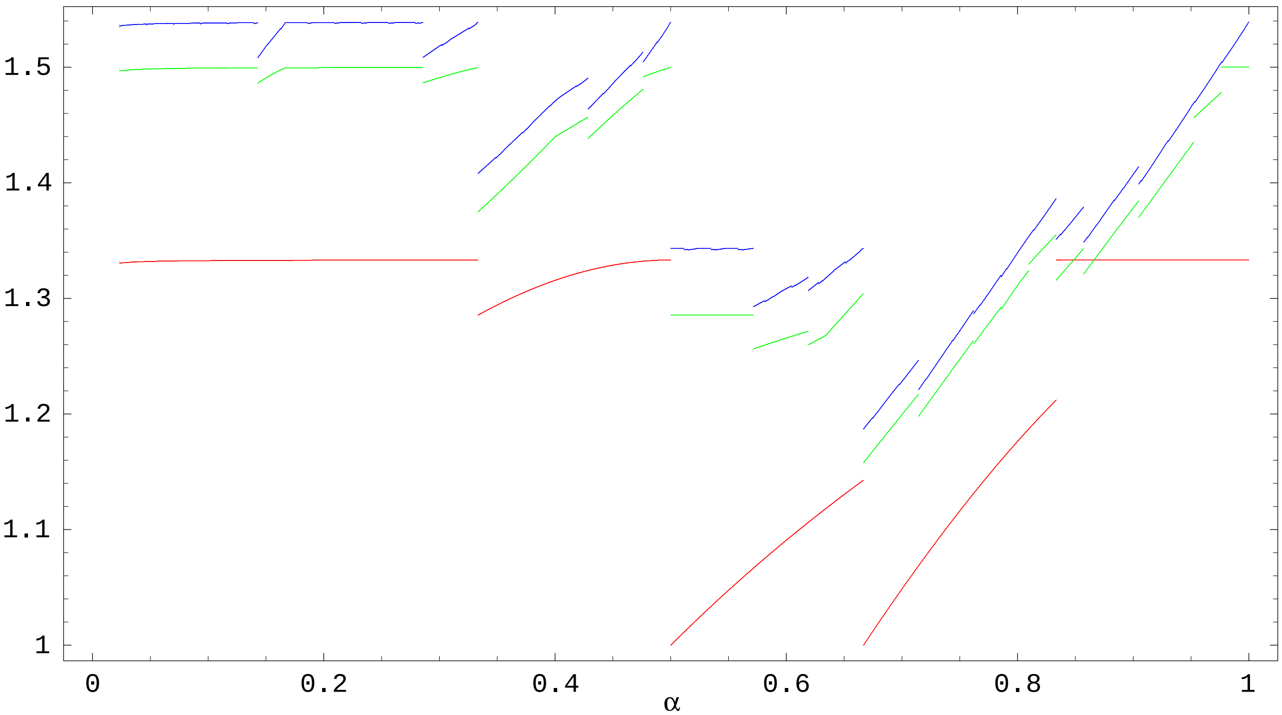

Figure 4: The evolution of the curves given by the greedy item sequence. In red is $\frac{1}{2}, \frac{1}{3}$; green is $\frac{1}{2}, \frac{1}{3}, \frac{1}{7}$; blue is $\frac{1}{2}, \frac{1}{3}, \frac{1}{7}, \frac{1}{43}$.

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 , as a shorthand for $,_0$.

The recurrence u_i described in Section 1, which is found in connection with bounded-space bin packing [12], 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 , (1). Somewhat surprisingly, it is also the sequence used by Brown, Liang and van Vliet in the construction of their lower bounds [1, 13, 18]. In essence, they analytically determine the value of $L_1(, _\tau(1))$. 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 items sizes in it, and we re-sort them so that the algorithm is confronted with items from smallest to largest. In general, this re-sorting seems to be 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.

Examining Figure 4, one immediately notices that that $L(, _\tau(1), \alpha)$ 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

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

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 lemmas:

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:

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 $1/\chi_i^* \sum_{p\in\mathcal{P}_i(\varrho)} \mathrm{size}(p)\phi(p)$, the factor $1/\chi_i^*$ decreases by at most $\alpha/(\alpha+\delta)$ and $\mathrm{size}(p)$ 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 at most $(\alpha-\delta)/\alpha$. ■

Considering Figure 4 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 , (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 $\alpha, , (1-\alpha)$. As expected, this sequence performs much better than , (1) in the areas described above.

It is possible to find further improvements for certain values of α. For instance, the sequence $\alpha/2, , (1-\alpha/2)$ also works well in some places, and we used other sequences as well. These are shown in Figure 5.

As a general guideline for finding sequences, items should not fit too well in either bin size. If an item has size x, then $\min\left\{1-\lfloor\frac{1}{x}\rfloor x, \alpha-\lfloor\frac{\alpha}{x}\rfloor x\right\}$ 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 using 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 fast or slow: 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.

Using Lemma 4.2 we obtain the main theorem of this section:

Theorem 4.1 *Any online algorithm for the variable sized bin packing problem with $m = 2$ has asymptotic performance ratio at least* $495176908800/370749511199 > 1.33561$.

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: The bin of size α is of no use.

We use the sequences described in the caption of Figure 5. 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 sub-interval. This constant sequence is fixed at the upper limit of the interval. I.e. 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 5 and 6. The lowest lower bound is 495176908800/370749511199, in the interval [0.7196,0.7197). ■

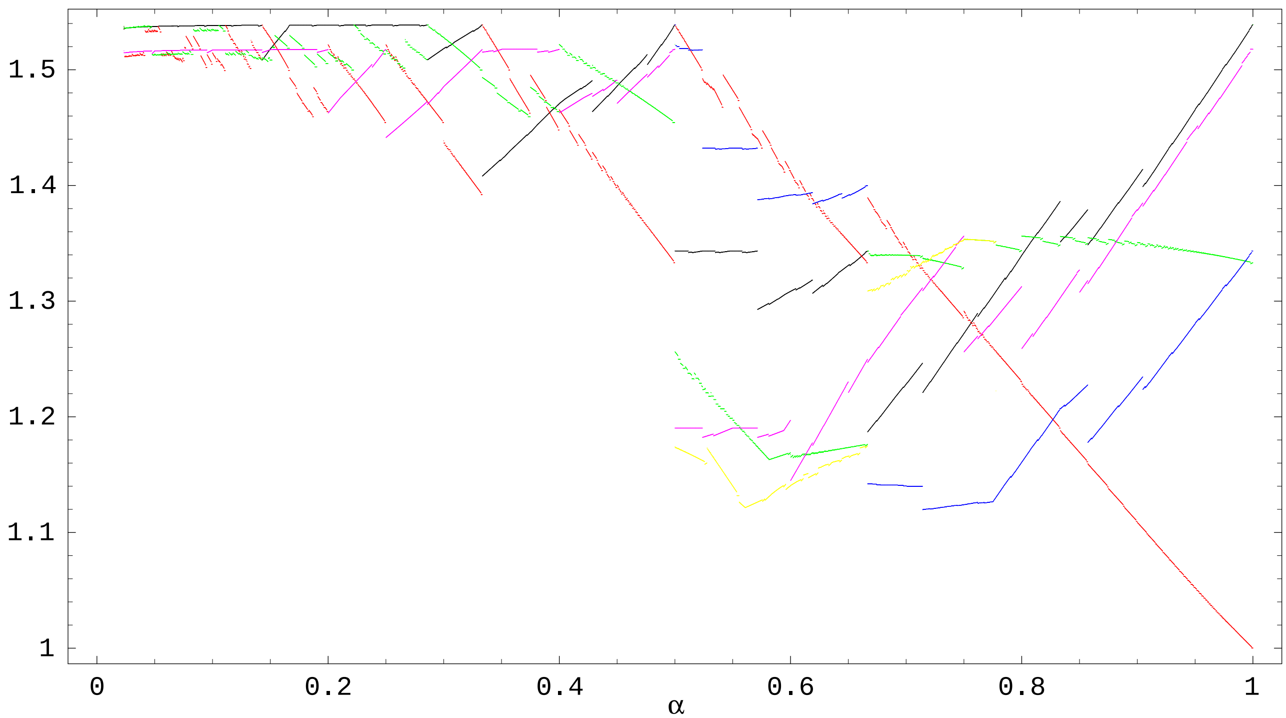

Figure 5: A variety of sequences for $\tau = 1/1000$: black is $\frac{1}{2}, \frac{1}{3}, \frac{1}{7}, \frac{1}{43}$; red is $\alpha, , \tau(1-\alpha)$; green is $\frac{\alpha}{2}, , \tau(1-\frac{\alpha}{2})$; blue is $\alpha, \frac{1}{3}, \frac{1}{7}, \frac{1}{43}$; purple is $\frac{1}{2}, \frac{1}{4}, \frac{1}{5}, \frac{1}{21}$; yellow is $\frac{1}{2}, \frac{\alpha}{2}, \frac{1}{9}, , \tau(\frac{7}{18} - \frac{\alpha}{2})$.

5. Conclusions

We have shown new algorithms and lower bounds for variable-sized on-line bin packing with two bin sizes. The largest gap between the performance of the algorithm and the lower bound is 0.18193, achieved for a second bin of size $\alpha = 0.9071$. The smallest gap is 0.03371 achieved for $\alpha = 0.6667$. Note that for $\alpha \le \frac{1}{2}$, there is not much difference with 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.

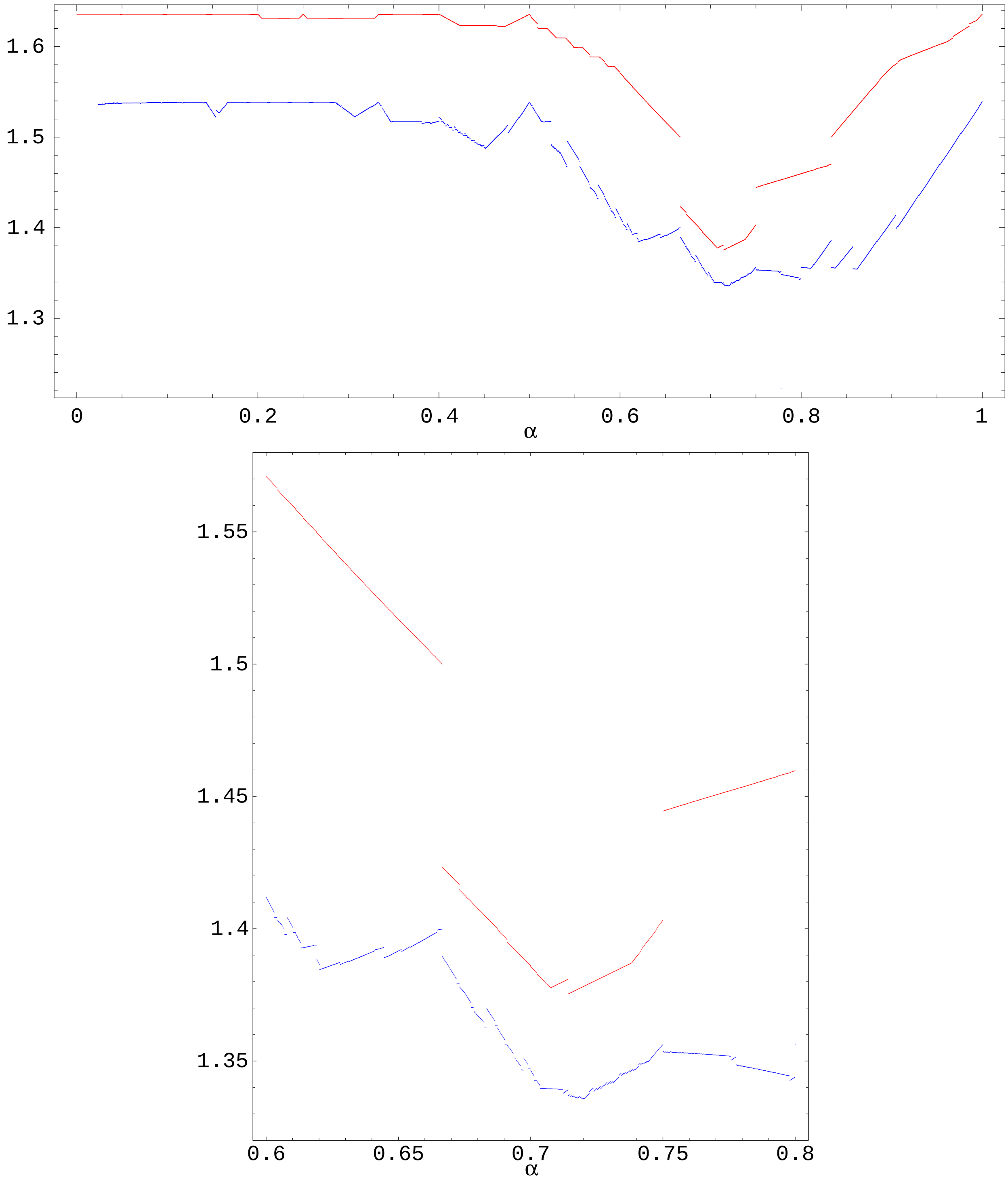

Figure 6: The best upper and lower bounds for variable sized online bin packing. The bottom figure is a closeup of [.6, .8]. The upper bound is best of the VRH1, VRH2 and VARIABLE HARMONIC algorithms.

References

1. Brown, D. J. A lower bound for on-line one-dimensional bin packing algorithms. Tech. Rep. R-864, Coordinated Sci. Lab., University of Illinois at Urbana-Champaign, 1979.
2. Chandra, B. Does randomization help in on-line bin packing? *Information Processing Letters 43*, 1 (Aug 1992), 15–19.
3. Coffman, E. G., Garey, M. R., and Johnson, D. S. Approximation algorithms for bin packing: A survey. In *Approximation Algorithms for NP-hard Problems*, D. Hochbuam, Ed. PWS Publishing Company, 1997, ch. 2.
4. Csirik, J. An on-line algorithm for variable-sized bin packing. *Acta Informatica 26*, 8 (1989), 697–709.
5. Csirik, J., and Woeginger, G. On-line packing and covering problems. In *On-Line Algorithms—The State of the Art*, A. Fiat and G. Woeginger, Eds., Lecture Notes in Computer Science. Springer-Verlag, 1998, ch. 7.
6. Friesen, D. K., and Langston, M. A. A storage-size selection problem. *Information Processing Letters 18* (1984), 295–296.
7. Friesen, D. K., and Langston, M. A. Variable sized bin packing. *SIAM Journal on Computing 15* (1986), 222–230.
8. Johnson, D. S. *Near-optimal bin packing algorithms.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1973.
9. Johnson, D. S. Fast algorithms for bin packing. *Journal Computer Systems Science 8* (1974), 272–314.
10. Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., and Graham, R. L. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing 3* (1974), 256–278.
11. Kinnersley, N., and Langston, M. Online variable-sized bin packing. *Discrete Applied Mathematics 22*, 2 (Feb 1989), 143–148.
12. Lee, C., and Lee, D. A simple on-line bin-packing algorithm. *Journal of the ACM 32*, 3 (Jul 1985), 562–572.

13. Liang, F. M. A lower bound for online bin packing. *Information Processing Letters 10* (1980), 76–79.

14. Ramanan, P., Brown, D., Lee, C., and Lee, D. On-line bin packing in linear time. *Journal of Algorithms 10*, 3 (Sep 1989), 305–326.

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

16. Seiden, S. On the online bin packing problem. Manuscript, 2000.

17. Seiden, S. S. An optimal online algorithm for bounded space variable-sized bin packing. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming* (Jul 2000), pp. 283–295.

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

19. Woeginger, G., and Zhang, G. Optimal on-line algorithms for variable-sized bin covering. Tech. Rep. Woe-22, TU Graz, Institut für Mathematik B, Feb 1998.

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