Packing Small Vectors

Yossi Azar * Ilan Reuven Cohen* Amos Fiat* Alan Roytman*

Abstract

Online d-dimensional vector packing models many settings such as minimizing resources in data centers where jobs have multiple resource requirements (CPU, Memory, etc.). However, no online d-dimensional vector packing algorithm can achieve a competitive ratio better than d. Fortunately, in many natural applications, vectors are relatively small, and thus the lower bound does not hold. For sufficiently small vectors, an $O(\log d)$-competitive algorithm was known. We improve this to a constant competitive ratio, arbitrarily close to $e \approx 2.718$, given that vectors are sufficiently small.

We give improved results for the two dimensional case. For arbitrarily small vectors, the first fit algorithm for two dimensional vector packing is no better than 2-competitive. We present a natural family of first fit variants, and for optimized parameters get a competitive ratio ≈ 1.48 for sufficiently small vectors.

We improve upon the 1.48 competitive ratio – not via a first fit variant – and give a competitive ratio arbitrarily close to $4/3$ for packing small, two dimensional vectors. We show that no algorithm can achieve better than a $4/3$ competitive ratio for two dimensional vectors, even if one allows the algorithm to split vectors among arbitrarily many bins.

*School of Computer Science, Tel-Aviv University. E-mails: azar@tau.ac.il, ilanrcohen@gmail.com, fiat@tau.ac.il, alan.roytman@cs.tau.ac.il.

1 Introduction

As cloud computing and the use of large server farms have become more prevalent, the costs of providing power and cooling servers have skyrocketed, so much so that these costs now surpass the costs of hardware and servers [15]. Every year, billions of dollars are spent on data centers due to the costs of energy consumption alone [2]. Indeed, server utilization in data centers is surprisingly low, and is estimated to be between 5% and 15% on average [5]. Such underutilized servers result in energy waste and monetary cost. Even small improvements in power efficiency can result in substantial gains monetary gains and have a positive impact on the environment.

The work of [16] studied the impact of resource contention among jobs by measuring energy consumption on individual hardware components. They concluded that jobs which do not contend for the same set of resources can be parallelized well and consume significantly less power when compared to jobs which make heavy use of the same resource. These results substantiate the idea that, when assigning jobs to machines, it is important to represent jobs as vectors to capture the fact that resource requirements are multidimensional (e.g., CPU, memory, and I/O). Modeling jobs in this manner is important to understand how to design algorithms that minimize the number of active servers in a scenario where jobs make heavy use of many hardware components. For instance, assigning multidimensional jobs to machines has applications in implementing databases for shared-nothing environments [11], along with optimizing parallel queries in databases as such tasks typically involve resources such as the CPU or disk [10]. Indeed, jobs are increasingly becoming more parallel, and hence leave a large footprint on many CPU cores.

The guarantees on performance are quite pessimistic for the general vector packing problem, but this may not accurately model what happens in practice. In fact, the requirements of any single job are typically small across all dimensions (relative to the total machine capacity). In this paper we study this scenario and exploit the restriction that inputs consist only of small vectors.

Motivated by these reasons, we study the classic Bin Packing problem in an online, multidimensional setting, and call this problem the *Vector Bin Packing* problem. In the offline version of the problem, we are given a set of vectors $\{v_1, \ldots, v_n\}$ where $v_i = (v_{i1}, \ldots, v_{ik}) \in [0,1]^d$ for all $i \in [n]$, where $[n] = \{1, \ldots, n\}$, and we must find a partition of the set of vectors into feasible sets $B_1, \ldots, B_m$ such that, for each $1 \leq j \leq m$ and each coordinate k, we have $\sum_{i \in B_j} v_{ik} \leq 1$. We refer to each set B_j as a *bin*, each of which has capacity 1 for each coordinate $1 \leq k \leq d$. The objective function is to minimize m, the number of bins used to feasibly pack all vectors. In the online version of the problem, d-dimensional vectors arrive in an online manner and must be immediately assigned to an open bin, or to a new bin, so that the capacity constraints on each bin are satisfied along each dimension. We focus on the setting where all vectors have small values in each coordinate (e.g., at most ϵ) relative to the size of a bin. We sometimes refer to v_{ik} as the *load* of vector v_i on dimension k.

The benchmark we use to measure the performance of an online algorithm is the competitive ratio. In particular, we compare how many bins an online algorithm ALG opens relative an optimal solution OPT that is omniscient and knows which vectors will arrive in the future. More formally, for any input sequence x, let $\text{ALG}(x)$ denote the number of bins used by the online algorithm and $\text{OPT}(x)$ denote the number of bins used by an optimal solution that knows the entire sequence of vectors in advance. We say that ALG is c-competitive if $\text{ALG}(x) \leq c \cdot \text{OPT}(x) + a$ for any input sequence x (where we allow some additive constant a).

The online Vector Bin Packing problem is trivial in one dimension ($d = 1$) for small items. In particular, if all (single dimensional) values are at most ϵ then any algorithm that avoids opening a new bin unless necessary (e.g., First Fit, Best Fit, Any Fit) is $(1 + O(\epsilon))$-competitive. When applying such algorithms in higher dimensions, ($d \geq 2$), and even if one only considers input sequences of arbitrarily small d-dimensional vectors, the competitive ratio is at least d.

Contributions and Techniques

Azar et al. [1] showed that the competitive ratio for the Vector Bin Packing problem in d dimensions must depend on d, if input sequences consist of arbitrary vectors. However, prior to the algorithms presented herein, and even if one only considers input sequences of arbitrarily small vectors, the best competitive ratio was at least $\log d$ [1]. As our main contribution for arbitrary d, we close this gap and give an $O(1)$-competitive algorithm. Our contributions in this paper are as follows:

1. We give a randomized algorithm in Section 3.1 that is e-competitive in expectation for Vector Bin Packing, where d is arbitrarily large and input vectors are sufficiently small. More precisely, for any $\epsilon > 0$, if all vectors are smaller than $O\left(\frac{\epsilon^2}{\log d}\right)$, then the expected competitive ratio is at most $(1+\epsilon)e$. We then derandomize this algorithm in Section 3.2 and get the same guarantees in the deterministic setting as the randomized setting, except that vectors must be smaller by an additional factor of $\log \frac{1}{\epsilon}$.

2. For two dimensional vectors we give improved results:

   (a) In Section 2.1, we describe a family of restricted first fit algorithms for vector bin packing when vectors are sufficiently small. We show that optimizing over this family gives an algorithm with a competitive ratio of $\approx 1.48 + O(\epsilon)$ if all vectors have values at most ϵ.

   (b) In Section 13, we give an optimal algorithm for two dimensional vector packing, when vectors are small. That is, for any $\epsilon > 0$, if all vectors are smaller than ϵ^2, then the (deterministic) competitive ratio is $\frac{4}{3} + O(\epsilon)$.

   (c) In Section 2.3, we show that the $\frac{4}{3} + O(\epsilon)$ algorithm is almost tight, by giving a lower bound of $\frac{4}{3}$ on the competitive ratio of Vector Bin Packing, even when vectors are arbitrarily small.

To obtain our results, we consider an intermediate problem: *splittable* vector bin packing. In the splittable vector setting vectors can be split into arbitrarily many fractions, $v \cdot \alpha_1, v \cdot \alpha_2, \ldots, v \cdot \alpha_k$, $\sum_i \alpha_i = 1$, where every fraction $v \cdot \alpha_i$ can be packed in a different bin. In the splittable model, the assumption that vectors are small is irrelevant as any big vector can be split into many small fractions. We remark that the lower bound of $4/3$ holds even in the splittable vector setting. We then give a reduction from the problem of small and unsplittable vector bin packing, to the problem of splittable vector bin packing, while losing little in the competitive ratio (as a function of the upper bound on the vector values). We note that we use significantly different rounding techniques for each of our contributions when going from the splittable setting to the unsplittable setting.

We begin with our techniques for $d = 2$ dimensions regarding our restricted first fit algorithms for splittable vectors. The main idea here is that we restrict certain bin configurations that cause a large imbalance between the loads on both bin dimensions. Each time we reject a vector from a bin, this puts restrictions on vectors that can be assigned to future bins. To handle the case when vectors are small and unsplittable, we note that the load on some bins may not lie on the curve defined by our function f. However, we argue that we can map the load on each such bin to a corresponding load on the curve that approximately preserves our restriction property.

For our $(\frac{4}{3}+O(\epsilon))$-competitive algorithm, we note that in the splittable setting, our algorithm is actually $\frac{4}{3}$-competitive. We obtain our result for the splittable setting by partioning bins into virtual bins of two types. We carefully design the algorithm to ensure that the total load among all bins of the first type is equal on both coordinates. We maintain the invariant that these bins, in aggregate, are sufficiently packed. For the second type of bins, we leave enough space to accommodate future vectors in order to guarantee that our invariant continues to hold. All together, this yields a tight

bound. To obtain our $\frac{4}{3} + O(\epsilon)$ result for the unsplittable setting when vectors are small, we assign vectors to buckets according to the ratio between their loads. Within each bucket, we can closely mimic the behavior of the algorithm in the splittable setting.

For our main contribution for arbitrary d, namely the $e(1+\epsilon)$-competitive algorithm, we note that in the splittable setting, our algorithm is e-competitive. This result is obtained by defining a probability density function that fully allocates vectors among continuous bins in a way that is essentially oblivious to the incoming vector. We maintain an accurate estimate of OPT and open at most $e \cdot \text{OPT}$ bins. To obtain our randomized and deterministic $e(1+\epsilon)$-competitive results for the unsplittable setting, our algorithm is inspired by [12]. However, we must overcome several obstacles. In particular, our arguments must be extended to the cases when vectors are small, the number of bins can change dynamically over time, and the probabilities are not uniform.

Related Work

There is a large body of work for the Vector Bin Packing problem, since it has practical applications to cloud computing, along with the rise of virtual machine consolidation and migration [14]. The most closely related paper to ours is the work of Azar et al. [1]. They showed a lower bound of $\Omega\left(d^{\frac{1}{B}-\epsilon}\right)$ for any $\epsilon > 0$, and gave an online algorithm that is $O\left(d^{\frac{1}{B-1}}(\log d)^{\frac{B}{B-1}}\right)$-competitive for any integer $B \geq 2$ (where B is the ratio between the largest coordinate and the size of each bin). For $B = O(\log d)$, we close their gap by designing an $O(1)$-competitive algorithm for arbitrary d, and provide an essentially tight algorithm for $d = 2$.

In general, the single dimensional case of Vector Bin Packing (i.e., the classic Bin Packing problem) has a vast body of work. We only mention some of the more closely related papers to our models and results. For a broader overview of the literature, there are surveys available concerning the offline and online versions of the Bin Packing problem, along with some multidimensional results and other models [4, 8].

The work of [3] gave an algorithm that, for any $\epsilon > 0$, achieved an approximation ratio of $O\left(1 + \epsilon d + \ln\left(\frac{1}{\epsilon}\right)\right)$ in polynomial time for the offline setting when d is arbitrary. They also showed that, unless $\mathbf{NP} = \mathbf{ZPP}$, it is impossible to obtain a polynomial-time approximation algorithm for Vector Bin Packing with an approximation ratio of $d^{\frac{1}{2}-\epsilon}$ for any $\epsilon > 0$ (this was strengthened to $d^{1-\epsilon}$ in [1]). For the online setting, a $(d + .7)$-competitive algorithm was given in [9].

There is also a line of research in the online setting that considers variable sized bins, which was first studied by [7] and more recently by [17]. In this problem, a set of bin capacity profiles $\mathbb{B}$ is given, each element of which is a vector in $\mathbb{R}^d$. The multidimensional version of this problem was introduced in [6], where it was shown that, for any $\epsilon > 0$, there is a set of profiles such that the competitive ratio is $1 + \epsilon$. Moreover, they provided a negative result by arguing that there exists a set of bin profiles such that any randomized algorithm must have a competitive ratio of $\Omega(d)$.

A closely related problem to the online Vector Bin Packing problem is the online Vector Scheduling problem with identical machines. In this setting, we have a hard constraint on the number of bins (i.e., machines) that are open, and we must minimize the makespan (i.e., the largest load over all machines and all dimensions). The multidimensional version of this problem was studied in the offline setting by [3], in which an $O(\log^2(d))$-approximation algorithm was given. For the online setting, this result was later improved by [1] and [13], where $O(\log d)$-competitive algorithms were given for the problem. More recently, an optimal algorithm with a competitive ratio of $O\left(\frac{\log d}{\log \log d}\right)$ along with a matching lower bound were given in [12].

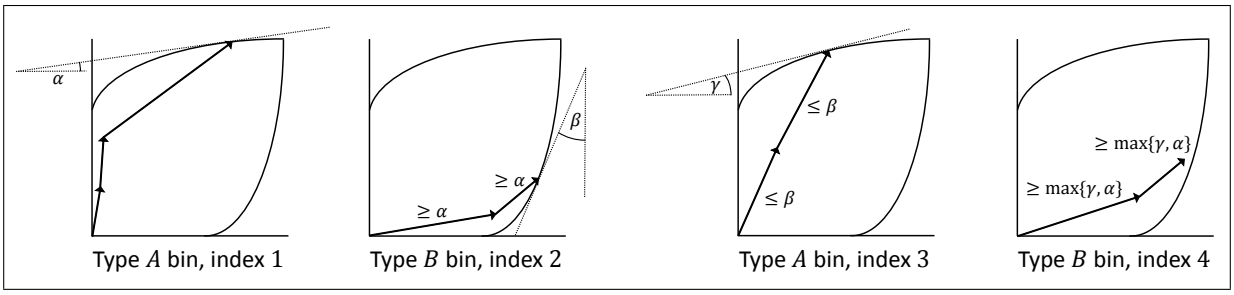

Figure 1: A snapshot during the execution of some f-restricted First Fit Algorithm. Note that subsequent to a Type A bin, where the vectors "touch" the forbidden curve above, the angle with the x-axis of the line tangent at this point (α at index 1, γ at index 3) is a lower bound on the angle of any vector (fragment) placed in a subsequent Type B bin. A similar claim holds for Type A bins subsequent to Type B bins.

2 Two Dimensional Vector Bin Packing

We first present our f-restricted First Fit Algorithm, which achieves a competitive ratio of ≈ 1.48 when vectors are splittable. If all vectors have values smaller than ϵ and are unsplittable, this incurs an additional factor of $1 + O(\epsilon)$ in the competitive ratio.

2.1 The f-restricted First Fit Algorithm

Splittable Vectors

Let $f : [0, 1] \to [0, 1]$ be a monotone non-decreasing concave function with $f(0) = c$ and $f(1) = 1$. We first define a class of f-restricted First Fit Algorithms when vectors are splittable (*i.e.*, when vectors v can be split into arbitrarily many fractions $\alpha_1 v$, $\alpha_2 v$, ..., $\alpha_k v$, where $\alpha_i \geq 0$, $\sum_i \alpha_i = 1$).

The f-restricted First Fit Algorithm is a variant of the first fit algorithm with the following twist:

- For an incoming vector v, add as large a fraction of v to the next bin in sequence such that the resulting load on the bin $\ell = (\ell_x, \ell_y)$ has $\ell_y \leq f(\ell_x)$ and $\ell_x \leq f(\ell_y)$.
- Continue assigning fractions of v to bins as above until the sum of fractions is equal to the original vector.

One can interpret the algorithm as a first fit algorithm for splittable vectors where every bin has "forbidden regions,": an "upper curve limit" $[t, f(t)]$ and a "right curve limit" $[f(t), t]$, for $0 \leq t \leq 1$. See Figure 1.

Observation 1. *At any point of the algorithm's run the load on every bin is either on the upper curve or the right curve, except at most one bin (the last active bin).*

Accordingly, we define a bin that has a load which lies on its upper curve to be of Type A, and similarly define a bin that has a load which lies on its right curve to be of Type B. Given a pair of points $a = (a_x, f(a_x))$ and $b = (f(b_y), b_y)$, we define $T(a, b)$ to be the competitive ratio assuming the load of all Type A bins is a and the load of all Type B bins is b. The following lemma is proved in Appendix A.

Lemma 1. $T(a, b) = \max\left\{\frac{a_y - a_x + b_x - b_y}{b_x a_y - a_x b_y}, \frac{1}{a_y}, \frac{1}{b_x}\right\}$.

We define a partial order on two dimensional vectors, $(v_x, v_y) \leq (\tilde{v}_x, \tilde{v}_y)$ if and only if $v_x \leq \tilde{v}_x$ and $v_y \leq \tilde{v}_y$. Note that this partial order defines a total order for vectors along the upper curve, and a total order for vectors on the right curve. This follows since f is monotone. By volume consideration it is easy to verify that:

Observation 2. *The function $T(a, b)$ is monotone. That is, if $a \leq \tilde{a}$, then $T(a, b) \geq T(\tilde{a}, b)$ for all b. Similarly, if $b \leq \tilde{b}$, then $T(a, b) \geq T(a, \tilde{b})$ for all a.*

For each point in $x \in [0, 1]$, we define $H(x) = (h_x, h_y)$ and $H^R(x) = (h_y, h_x)$ such that $h_y/h_x = f'(x)$, $h_x = f(h_y)$. The following lemma and theorem are proved in Appendix A.

Lemma 2. *Let $\tilde{a}, \tilde{b}$ be the smallest load among all Type A,B bin loads, respectively. Then, $\tilde{b} \geq H(a_x)$ or $\tilde{a} \geq H^R(b_y)$.*

Theorem 3. *Let $f : [0, 1] \mapsto [0, 1]$ be a monotone non-decreasing concave function. The competitive ratio of the f-restricted First Fit Algorithm is $\max_x T((x, f(x)), H(x))$.*

Unsplittable, Small Vectors

One can naturally define an f-restricted First Fit Algorithm in the context of unsplittable vectors. That is, assign a vector to a bin only if adding the vector in its entirety does not violate the restrictions. Now, we verify that this f-restricted First Fit Algorithm obtains almost the same competitive ratio as in the splittable case, subject to the condition that vectors are sufficiently small (i.e., vector coordinate values are at most ϵ). We prove the following in Appendix A, and give an accompanying figure.

Lemma 4. *Let f be a function such that our f-restricted First Fit Algorithm yields an α-competitive ratio for the splittable vector setting. Then, the f-restricted First Fit Algorithm, when applied to unsplittable vectors, achieves a competitive ratio of $\alpha(1+O(\epsilon))$ (assuming all coordinates have value at most ϵ).*

In Appendix B, we give the best competitive ratio attainable when f is a linear function, and describe a family of f-restricted First Fit algorithms when vectors are sufficiently small. We approximately optimize over this family, using piecewise linear functions. In particular, we prove the following.

Theorem 5. *Assume that all vectors are unsplittable and have value smaller than ϵ. If f is a linear function, the best attainable competitive ratio for f-restricted First Fit algorithms is $\phi(1 + O(\epsilon))$, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. If f can be piecewise linear, there is a function which achieves a competitive ratio of $\approx 1.48(1 + O(\epsilon))$.*

2.2 A $\frac{4}{3}$-competitive Algorithm

Splittable Vectors

We now give a $\frac{4}{3}$-competitive online algorithm for splittable vectors. We write our algorithm assuming its competitive ratio is some value $c > 1$, and eventually argue that $c = \frac{4}{3}$ is sufficient to carry out our proof. Our algorithm works by maintaining virtual bins, each of which has equal x and y dimensions, which can be some fraction ≤ 1, called the *size* of the virtual bin. An arriving online vector may be split into multiple parts, each of which is assigned to some virtual bin. Each real bin consists of some number of virtual bins. The sum of the sizes of the virtual bins assigned

to a real bin is exactly one, except for possibly the last real bin which may have some unallocated space.

We distinguish two types of virtual bins, *open* and *closed* virtual bins. Over time, when new vectors arrive, new virtual bins may be created. The new virtual bins may be placed in the unallocated space of the last real bin, or a new real bin may be created to accommodate new virtual bins. When created, a new virtual bin may be open or closed. The size of the new virtual bin is determined when created and never changes. Subsequently, fractions of newly arriving vectors may be assigned to open virtual bins (but not to closed virtual bins). An open virtual bin may become a closed virtual bin.

Let V_1 denote the total load from vectors along the first coordinate, and let V_2 denote the total load from vectors along the second coordinate. Throughout the algorithm, we assume without loss of generality that $V_1 \geq V_2$. In fact, we assume that $V_1 \geq V_2$ even after the vector arrives: if this is not the case we can represent the incoming vector v as the sum of two vectors, $v = v' + v''$, $v' = \alpha v$, $v'' = (1-\alpha)v$, $0 \leq \alpha \leq 1$. Dealing with v' results in the volume on both coordinates being equal, subsequently – we "rename" the coordinates, with the leading coordinate (now coordinate 2) being renamed to be coordinate 1.

Each open virtual bin has zero load on its second coordinate, while the load on its first coordinate occupies exactly a $\frac{1}{c}$-fraction of the open virtual bin's size. The main purpose for leaving free space in open virtual bins is to accommodate future vectors.

```
1  while vector v = (x, y) arrives do
2      if x ≥ y then                                          // See Figure 2 Case (a)
3          Allocate a virtual bin of size c(x − y) + y, and assign v into it
4          Mark the segment [0, y] of the bin as closed, and [y, c(x − y) + y] as open
5      else
6          Get an open virtual bin b of size c(y − x)
7          if y ≥ c/(c−1) x then                              // See Figure 2 Case (b)
8              Assign v to b
9          else                                               // See Figure 2 Case (c)
10             Let f ← (c − 1)(y/x − 1), and assign the vector f · v to bin b
11             Allocate a virtual bin a of size (1 − f) · y
12             Assign the remaining vector (1 − f) · v into bin a, mark it as closed
13         Mark bin b as closed
```

Algorithm 1: The 4/3 competitive algorithm for splittable, two dimensional vectors.

In lines 3 and 11 of Algorithm 1 one needs to allocate new virtual bins. In fact, such bins may span more than one real bin. For ease of exposition, we assume that if a virtual bin is to span more than one real bin, the incoming vector v is split into two smaller vectors αv and $(1-\alpha)v$, so that the allocated virtual bin fits into one real bin.

We now argue that in line 6 of Algorithm 1, a suitable virtual bin (of size $c(y-x)$) must be available. In fact – as stated – this is simply false. However, we prove an equivalent claim that one can split the incoming vector v into multiple parts, and split existing open virtual bins, so that all fractional parts of v have an appropriately sized virtual bin available.

Let $v = (x, y)$ be the incoming vector. Let V_1, V_2 be the loads before the arrival of v, and $V_1' = V_1 + x$, $V_2' = V_2 + y$ be the sum of coordinate loads after the arrival of v. By assumption, $V_1' \geq V_2'$, and hence $V_1 - V_2 \geq y - x$.

We prove below that the algorithm preserves the four invariants given in Figure 3, in particular, invariant 2 in Figure 3 says that the sum of the sizes of the open virtual bins is $c(V_1 - V_2)$. By definition, the sum of the loads on the first coordinate is $V_1 - V_2 \geq y - x$. That is, in aggregate the sum of sizes of open virtual bins is $\geq c(y-x)$.

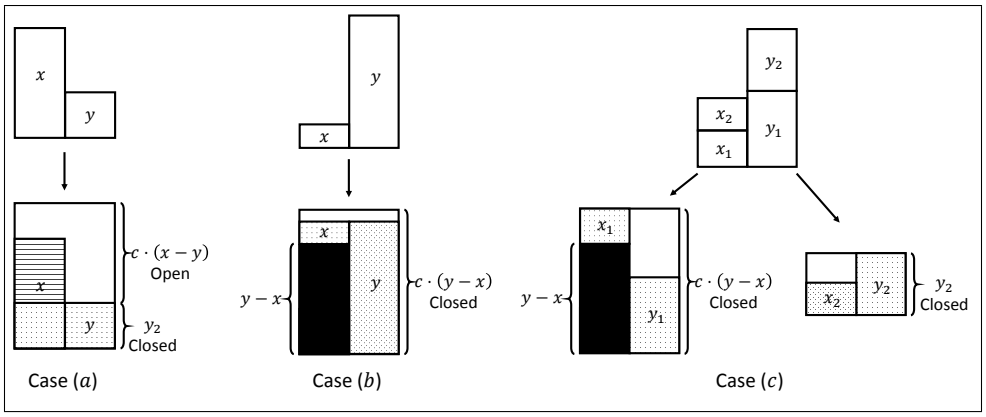

Figure 2: The various cases regarding allocation/use and reclassification of bins in the (optimal) 4/3 splittable vector bin packing algorithm. Striped regions denote the load on closed virtual bins, dotted regions denote the load on open virtual bins, and black regions denote the load on an existing virtual bin, which is reclassified as a closed virtual bin. Note that in Case (c), we have $x_1 + x_2 = x$, $y_1 + y_2 = y$, and moreover $\frac{y_1}{x_1} = \frac{y_2}{x_2}$.

If any open virtual bin has size equal to $c(y-x)$ we are done. If any such bin has size $> c(y-x)$, we can split this open virtual bin into two smaller open virtual bins, one of which has size exactly $c(y-x)$. Otherwise, if all open virtual bins are strictly smaller than $c(y-x)$, we split the vector into two smaller fractions, one of which can fit into one of these small open virtual bins, and reiterate the process. We prove the theorem given below in Appendix C. The proof is based on the idea that our algorithm maintains the invariants in Figure 3 at all times.

1. The total load on both coordinates among all closed virtual bins equals V_2,
2. The total load on the first coordinate among all open virtual bins equals $V_1 - V_2$,
3. Each open virtual bin has zero load on its second coordinate, while the load on its first coordinate occupies exactly a $\frac{1}{c}$-fraction of the open virtual bin's size,
4. The total load on the first coordinate among all closed virtual bins occupies at least a $\frac{1}{c}$-fraction of the total space allocated for closed virtual bins (we also maintain the same invariant for the second coordinate among all closed virtual bins).

Figure 3: Invariants maintained by the (optimal) $\frac{4}{3}$-competitive algorithm for splittable, two dimensional vectors. Proof that these invariants are preserved is given in Theorem 6. As discussed, one can assume without loss of generality that $V_1 \geq V_2$ up to a renaming of the coordinates.

Theorem 6. *For $d = 2$ dimensions, there is an algorithm for the online Vector Bin Packing problem which achieves a competitive ratio of $\frac{4}{3}$ when vectors are splittable.*

The Splittable Algorithm on Real Bins

We now describe how to implement Algorithm 1 with real bins. The algorithm maintains one real bin with partially unallocated space, which is the most recently opened bin. The previous bins are fully allocated and partitioned into closed virtual bins and open virtual bins. When the algorithm needs to assign a vector into open virtual bins, it does so in a first fit manner. For each real bin i,

we maintain a value O_i, which is the total size of open virtual bins on bin i, and C_i, which is the total size of closed virtual bins on bin i.

In general, the state of the algorithm is a sequence of bins of type A, B and at most one type C bin. Type A bins are fully allocated real bins where $C_i = 1$. Type B bins are fully allocated real bins where $O_i + C_i = 1, O_i > 0$ (i.e., such bins have at least one open virtual bin). A Type C bin is a partially allocated real bin, where $O_i + C_i < 1$. In Appendix C, we have a figure which shows the general state of the algorithm on real bins, and we explicitly give an algorithm on real bins that is similar in flavor to Algorithm 1.

Unsplittable, Small Vectors

First, we describe a slight modification to our $\frac{4}{3}$-competitive algorithm that operates on real bins for the splittable vector case which guarantees that a vector may be split into at most two bins. This constraint incurs a $1 + O(\epsilon)$ loss in the competitive ratio. We simulate the $\frac{4}{3}$-competitive algorithm on bins of size $1 - 2\epsilon$. Note that the algorithm for the original vector v iterates if $C_k + O_k > 1$ or $O_b < 0$, where k is the Type C bin and b is the first Type B bin. In this case, instead of iterating, we modify the algorithm by assigning the whole vector to a bin which violates a constraint. Every time a constraint is violated, the bin to which the vector is assigned changes its type. Since a bin can change its type at most twice and the load of v is at most ϵ, the assignment is feasible. By volume consideration, this algorithm is $\frac{c}{1-2\epsilon}$-competitive. Clearly, if the algorithm assigns the vector to one bin, we follow this assignment. After this modification, if a vector is split, it is split into a Type B bin and a Type C bin.

For the unsplittable case, we discretize all possible ratios by rounding each vector's ratio down to the nearest power of $(1 + \sqrt{\epsilon})$, and assign each vector to buckets according to its ratio. For example, for a vector (x, y) where $x < y < 4x$, we let $r = \frac{y}{x}$ and map the vector to the bucket corresponding to $(1 + \sqrt{\epsilon})^j$, where j is defined such that $(1 + \sqrt{\epsilon})^j \leq r < (1 + \sqrt{\epsilon})^{j+1}$. Based on these buckets, the algorithm for unsplittable vectors attempts to mimic the load of the splittable vector algorithm's load on each bucket. As mentioned, the splittable vector algorithm may assign a fraction to a Type B bin b and a fraction to the Type C bin k. We ensure that the rounded load on k is smaller than the load of the splittable vector algorithm on k, but the rounded load on bin b is just up to 2ϵ larger for each bucket, since each vector's load is at most ϵ. Hence, we lose an additive 2ϵ for each bucket, of which there are $O\left(1/\sqrt{\epsilon}\right)$. Therefore, the unsplittable algorithm will never have a bin that overflows relative to an algorithm for splittable vectors that is constrained to have bins of size $1 - O(\sqrt{\epsilon})$. Moreover, we lose a multiplicative factor of $1 + \sqrt{\epsilon}$ due to the fact that we discretize the vectors. Hence, we lose a $(1 + O(\sqrt{\epsilon}))$-factor in our competitive ratio. We get the following theorem:

Theorem 7. *There is a $\frac{4}{3}(1+\epsilon)$-competitive algorithm for the two dimensional Vector Bin Packing problem when vectors are unsplittable and have coordinate values at most $O(\epsilon^2)$.*

2.3 A Tight Lower Bound

In this section we show a lower bound of $\frac{4}{3}$ on the competitive ratio of any online splittable Vector Bin Packing algorithm. This trivially extends to bin packing of unsplittable vectors, even for arbitrarily small unsplittable vectors and for arbitrary dimension, since such settings are more general. The proof is in Appendix D.

Theorem 8. *There is no online algorithm for the online, two dimensional Vector Bin Packing problem which achieves a competitive ratio better than $\frac{4}{3}$, even when vectors are splittable.*

3 Online Multidimensional Vector Bin Packing

Splittable Vectors

For arbitrarily large d, we first give an online, deterministic algorithm for the splittable vector setting which achieves a competitive ratio of e. Before describing the algorithm, we first imagine infinitesimally small bins consecutively along the real interval $[0, \infty)$ (at the limit each bin is a real number). Actual bins, however, are integer-aligned intervals of length 1 along $[0, \infty)$ (i.e., bin i corresponds to the interval $[i-1, i)$ for all integers $i \geq 1$).

Our algorithm utilizes a probability density function (to be defined). When receiving an input vector it, allocates fractions thereof among the infinitesimally small bins. That is, deterministically allocating (infintesimally small) fractions of the input vector among these infinitesimally small bins, proportional to the density function. Let V_k denote the total load that has arrived thus far along dimension k for $k \in \{1, \ldots, d\}$. Our algorithm maintains $V = \max\{V_1, \ldots, V_d\} \leq \text{OPT}$ at all times.

while *vector* v_i *arrives* **do**
  Split v_i into infinitesimally small fractions
  while *some infinitesimal fraction of* v_i, δv_i, *remains to be allocated* **do**
    Recompute $V_1, \ldots, V_d$, (every V_i is incremented by δv_i)
    Let $V = \max\{V_1, \ldots, V_d\}$
    Allocate δv_i to the infinitesimally small bins in the interval $I = [V, e \cdot V]$
    Bin $x \in I$ is allocated a fraction of δv_i according to the density function $f(x) = \frac{1}{x}$

Algorithm 2: Sliding Window Assignment.

We prove the following theorem in Appendix E.

Theorem 9. *Algorithm Sliding Window Assignment is a deterministic e-competitive algorithm for the online multidimensional Vector Bin Packing problem for any dimension d.*

The following algorithm is implementation of Algorithm 2 on real bins. Note that, the following algorithm assumes that upon of arrival of a vector v_i, $\lceil V \rceil$ does not increase (if it does we split the vector so that it does not change).

while *vector* v_i *arrives* **do**
  Let $V = \max\{V_1, \ldots, V_d\}$
  Allocate v_i to bin j according to the following fraction p_{ij}

$$p_{ij} = \begin{cases} \ln\left(\frac{j+1}{j}\right) & \lceil V \rceil \leq j \leq \lfloor \lceil V \rceil \cdot e \rfloor \\ \ln\left(\frac{\lceil V \rceil \cdot e}{j}\right) & j = \lceil \lceil V \rceil \cdot e \rceil \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 3: Sliding Window Assignment.

Proof. We show the following three things which give the theorem:

1. Each vector is fully allocated.
2. The load on each dimension of each bin is at most 1.
3. The number of bins opened by the algorithm is at most $e \cdot \text{OPT} + 1$.

We first prove the each vector is fully allocated, i.e. $\sum_j p_{ij} = 1$:

$$\sum_j p_{ij} = \ln\left(\frac{\lceil V\rceil+1}{\lceil V\rceil}\right)+\ln\left(\frac{\lceil V\rceil+2}{\lceil V\rceil+1}\right)+\cdots+\ln\left(\frac{\lceil\lceil V\rceil\cdot e\rceil}{\lceil\lceil V\rceil\cdot e\rceil-1}\right)+\ln\left(\frac{\lceil V\rceil\cdot e}{\lceil\lceil V\rceil\cdot e\rceil}\right) = \ln\left(\frac{\lceil V\rceil\cdot e}{\lceil V\rceil}\right) = 1$$

Second, for each bin j and each dimension k we prove that the total load fractionally assigned is at most 1. The total load assigned to bin j on dimension k is given by:

$$\sum_i v_{ik}\cdot p_{ij} \le j\cdot\sum_i p_{ij} \le j\cdot\ln(\frac{j+1}{j}) \le 1,$$

where the first inequality follows from $p_{i'j} = 0$ if $\lceil V\rceil > j$. In particular if $\sum_{i=1}^{i'-1} v_{ik} > j$ then $\lceil V\rceil > j$. The second inequality follows from $p_{ij} \le \ln(\frac{j+1}{j})$. The third inequality follows from $\ln(1+1/j) \le 1/j$.

Third, clearly the algorithm opens at most $\lceil\lceil V\rceil\cdot e\rceil$ bins. Since $\textsc{opt} = \lceil V\rceil$, we get that the algorithm opens at most $\lceil\textsc{opt}\cdot e\rceil \le e\cdot\textsc{opt}+1$ bins. This gives the theorem. □

In the following subsections, we simulate Algorithm Sliding Window Assignment and generalize the techniques from [12] to achieve a randomized and deterministic algorithm for the unsplittable, small vector case.

3.1 A Randomized Algorithm for Unsplittable, Small Vectors

Our randomized unsplittable vector algorithm simulates Algorithm Sliding Window Assignment on bins of smaller size, and uses its allocation as a probability function in order to randomly choose a bin. If the vector fits into the randomly chosen bin, it assigns the vector to the bin, and otherwise it passes the vector to a second stage First Fit algorithm that assigns the vector to spillover bins. Our main lemma bounds the total volume of overflowing vectors given to the First Fit algorithm.

```
1 while vector v_i arrives do
2     Simulate Algorithm 3 on bins of size 1 − ε
3     Let p_{i,j} denote the fraction of vector v_i assigned to bin j by Algorithm 3
4     Assign vector v_i to a random bin by interpreting the fractions p_{i,j} as probabilities
5     Let j* denote the randomly chosen bin
6     if vector v_i fits into bin j* then
7         Assign vector v_i to bin j*
8     else
9         Assign vector v_i using the First Fit algorithm to spillover bins
```

Algorithm 4: Randomized Sliding Window Assignment - Unsplittable.

Note that simulating Algorithm 3 on $(1-\epsilon)$-sized bins is equivalent to multiplying each vector by a $(1+\epsilon)$-factor. By volume consideration, since each bin is smaller by a $(1-\epsilon)$-factor, the competitive ratio increases by at most a $(1+\epsilon)$-factor. In addition, we have the property that $\sum_i p_{i,j}v_{ik} \le \frac{1}{1+\epsilon}$ since Algorithm 3 gives a feasible assignment to bins of size $(1-\epsilon)$. Let I^s denote the indices of vectors assigned by the First Fit algorithm (in line 9), and let V^s be the total volume of these vectors, namely $V^s = \sum_{i\in I^s}\sum_{k\in[d]} v_{ik}$.

Observation 3. *The number of bins opened by the First Fit algorithm is at most* $2V^s+1$.

This holds since the First Fit algorithm guarantees that there is at most one bin with total volume less than $\frac{1}{2}$. We first bound the probability that a vector does not fit in its randomly chosen bin, we prove the following lemma and theorem in Appendix E.

Lemma 10. *If the maximum coordinate on each vector is smaller than $\epsilon^2/(24 \log d)$ (for $\epsilon \leq 1/2$), then the probability that a vector does not fit in its randomly chosen bin is at most $1/d^3$.*

Theorem 11. *Algorithm 4 achieves an expected competitive ratio of $e(1+\epsilon)+o(1)$ for the Vector Bin Packing problem when d is arbitrary and all vectors are unsplittable and small (i.e., all values are smaller than $\epsilon^2/(24 \log d)$), where $o(1)$ is an arbitrarily small function of $1/d$.*

3.2 A Deterministic Algorithm for Unsplittable, Small Vectors

Finally, we introduce a deterministic algorithm that is $e(1+\epsilon)$-competitive when vectors are small and unsplittable. We fix the index i of vector v_i, dimension $k \in [d]$, and an open bin j. Let $f(x) = \alpha^x$ where $\alpha = e^{\epsilon/2}$, and let $L^i_{j,k} = \sum_{i \in V[j]} v_{ik}$, where $V[j]$ is the set of vectors that are (virtually) assigned to bin j. The algorithm uses the following potential function, where A is the current set of open bins (which may change and does not include the spillover bins opened by the First Fit algorithm):

$$\Phi^i_A = \sum_{j \in A} \sum_{k \in [d]} \Phi^i_{j,k}, \quad \Phi^i_{j,k} = f(L^i_{j,k} - \alpha \cdot \sum_{i' \leq i} p_{i',j} v_{ik}).$$

```
1 while vector v_i arrives do
2     Simulate Algorithm 3 on (1 − ε̃) sized bins
3     Let p_{i,j} be the fractional assignment of vector i to bin j
4     Assign vector v_i to the bin j* which minimizes the potential Φ^i_{{j|p_{i,j}>0}}
5     if vector i fits into bin j* then
6         Assign vector i to bin j*
7     else
8         Assign vector i using the First Fit algorithm to spillover bins
```

Algorithm 5: Sliding Window Assignment - Unsplittable Determinstic.

We prove the following theorem in Appendix E.

Theorem 12. *There exists a deterministic $(e(1+\epsilon)+o(1))$-competitive algorithm for the Vector Bin Packing problem for arbitrary d when vectors are unsplittable and smaller than $O(\frac{\epsilon^2}{\log d \log(\frac{1}{\epsilon})})$, where $o(1)$ is an arbitrarily small function of $1/d$.*

References

[1] Yossi Azar, Ilan Reuven Cohen, Seny Kamara, and Bruce Shepherd. Tight bounds for online vector bin packing. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing*, 2013.

[2] Anton Beloglazov and Rajkumar Buyya. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010.

[3] Chandra Chekuri and Sanjeev Khanna. On multi-dimensional packing problems. In *Proceedings of the 10th annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[4] Edward G. Coffman Jr., János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: survey and classification. In *Handbook of Combinatorial Optimization*, pages 455–531. Springer, 2013.

[5] U. S. Environmental Protection Agency. Report to congress on server and data center energy efficiency public law 109-431. Technical report, EPA ENERGY STAR Program, 2007.

[6] Leah Epstein. On variable sized vector packing. *Acta Cybernetica*, 16(1):47–56, 2003.

[7] Donald K. Friesen and Michael A. Langston. Variable sized bin packing. *SIAM Journal on Computing*, 15(1):222–230, 1986.

[8] Gabor Galambos and Gerhard J. Woeginger. On-line bin packing a restricted survey. *Zeitschrift für Operations Research*, 42(1):25–45, 1995.

[9] Michael R. Garey, Ronald L. Graham, David S. Johnson, and Andrew Chi-Chih Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257–298, 1976.

[10] Minos N. Garofalakis and Yannis E. Ioannidis. Scheduling issues in multimedia query optimization. *ACM Computing Surveys*, 1995.

[11] Minos N. Garofalakis and Yannis E. Ioannidis. Multi-dimensional resource scheduling for parallel queries. In *Proceedings of 1996 ACM SIGMOD International Conference on Management of Data*, 1996.

[12] Sungjin Im, Nathaniel Kell, Janardhan Kulkarni, and Debmalya Panigrahi. Tight bounds for online vector scheduling. *CoRR*, abs/1411.3887, 2014.

[13] Adam Meyerson, Alan Roytman, and Brian Tagiku. Online multidimensional load balancing. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*. Springer Berlin Heidelberg, 2013.

[14] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. *Microsoft Research T.R.*, 2011.

[15] Meikel Poess and Raghunath Othayoth Nambiar. Energy cost, the key challenge of today's data centers: a power consumption analysis of tpc-c results. *Proceedings of the VLDB Endowment*, 2008.

[16] Sebi Ryffel, Thanos Stathopoulos, Dustin McIntire, William Kaiser, and Lothar Thiele. Accurate energy attribution and accounting for multi-core systems. In *Technical Report 67, Center for Embedded Network Sensing*, 2009.

[17] Steven S. Seiden, Rob Van Stee, and Leah Epstein. New bounds for variable-sized online bin packing. *SIAM Journal on Computing*, 32(2):455–469, 2003.

A Restricted Fit Proofs

Proof of Lemma 1

Proof. Let n_A, n_B be the number of Type A bins with a load of a and the number of Type B bins with a load of b, respectively. By definition (we omit the additive 1 in the numerator because there is at most one bin which is neither Type A nor Type B):

$$T(a,b) = \max_{n_A \geq 0, n_B \geq 0} \frac{n_A + n_B}{\max\{n_A a_x + n_B b_x, n_A a_y + n_B b_y\}}.$$

Let $k = n_A/n_B$, then we have:

$$T(a,b) = \sup_{k\geq 0} \frac{1+k}{\max\{a_x + kb_x, a_y + kb_y\}} = \sup_{k\geq 0} \min\left\{\frac{1+k}{a_x+kb_x}, \frac{1+k}{a_y+kb_y}\right\}.$$

Let $f_1(k) = \frac{1+k}{a_x+kb_x}$, $f_2(k) = \frac{1+k}{a_y+kb_y}$. Taking the derivative of f_1, f_2, one observes that both f_1 and f_2 are monotone functions. Note also that $f_1(0) = 1/a_x$, $f_2(0) = 1/a_y$, while $f_1(k) \to 1/b_x$ and $f_2(k) \to 1/b_y$ as $k \to \infty$. These functions intersect at most once for k' such that $a_x + k'b_x = a_y + k'b_y$ (if $k' = (a_y - a_x)/(b_x - b_y) > 0$). If the two functions do not intersect, then the maximum is at $k = 0$ or at $k \to \infty$. If these functions do intersect, then the minimum function is the composition of two monotone functions from $[0, k']$ and $[k', \infty)$. Therefore, the maximum is at $k = 0$, $k \to \infty$, or $k = k'$. If the maximum is at $k = 0$, then $\min\{f_1(0), f_2(0)\} \leq 1/a_y$, if the maximum is at $k \to \infty$, then $\lim_{k\to\infty} \min\{f_1(k), f_2(k)\} \leq 1/b_x$, if the maximum is at $k = k'$, then

$$f_1(k') = f_2(k') = \frac{a_y - a_x + b_x - b_y}{b_x a_y - a_x b_y}.$$

□

Proof of Lemma 2

Proof. Let $i_{\tilde{a}}$ be the index of the first Type A bin with a load of $\tilde{a}$, similarly let $i_{\tilde{b}}$ be the index of the first Type B bin with load $\tilde{b}$. Assume without loss of generality that $i_{\tilde{a}} < i_{\tilde{b}}$. Let v be a vector that is packed into bin $i_{\tilde{b}}$. Let $a' = (a'_x, a'_y)$ be the load of bin $i_{\tilde{a}}$ when v arrives. Since v did not fit into bin $i_{\tilde{a}}$ (by the First Fit property), we have $v_y/v_x \geq f'(a'_x) \geq f'(\tilde{a}_x)$ since f is concave. As all derivatives are at least $f'(\tilde{a}_x)$, then $\tilde{b}_y \geq \tilde{b}_x \cdot f'(\tilde{a}_x)$, and therefore $\tilde{b} \geq H(\tilde{a}_x)$. □

Proof of Theorem 3

Proof. Let U be the set of loads of Type A bins and R be the set of loads of Type B bins, and $n_A = |U|, n_B = |R|$. By Observation 1, the number of total bins is at most $\text{ALG} \leq n_A + n_B + 1$. Let $\tilde{a} = (\tilde{a}_x, \tilde{a}_y) \in U$ be a point such that $\tilde{a} \leq a$ for all $a \in U$, and let $\tilde{b} = (\tilde{b}_x, \tilde{b}_y) \in R$ be a point such that $\tilde{b} \leq b$ for all $b \in R$. By volume consideration:

$$\text{OPT} \geq \max\left\{\sum_{a\in A} a_x + \sum_{b\in B} b_x, \sum_{a\in A} a_y + \sum_{b\in B} b_y\right\} \geq \max\left\{n_A\tilde{a}_x + n_B\tilde{b}_x, n_A\tilde{a}_y + n_B\tilde{b}_y\right\}.$$

Therefore, we have

$$\frac{\text{ALG} - 1}{\text{OPT}} \leq T\left(\tilde{a}, \tilde{b}\right).$$

By applying Observation 2 and Lemma 2 (we assume without loss of generality that $\tilde{b} \geq H(a_x)$), we have:

$$T(\tilde{a}, \tilde{b}) \leq T\left(\tilde{a}, H(a_x)\right) \leq \max_x T\left((x, f(x)), H(x)\right).$$

It is easy to verify that the analysis is tight. □

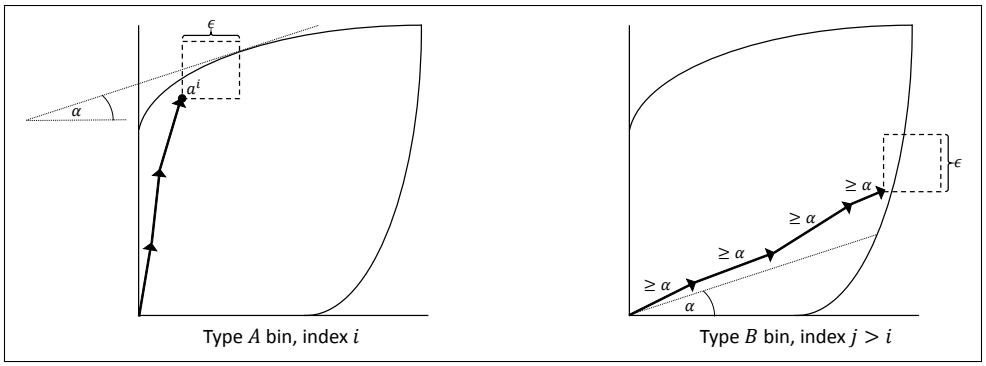

Figure 4: Applying an f-restricted First Fit Algorithm to packing unsplittable, small vectors. The analysis follows similarly to that of splittable vectors.

A.1 Proof of Lemma 4

In order to bound the competitive ratio of the f-restricted First Fit Algorithm for unsplittable, small vectors, we proceed in a manner similar to splittable vectors, and compute the worst possible pair of bins, one of Type A and the other of Type B. That is, we map the loads on bins in the unsplittable case to points on the curve and show that it is $(1+\epsilon)$ "close" to a corresponding pair of points on the curve that have a good competitive ratio.

Let (a^i, b^i) be the loads on the worst Type A bin and Type B bin, respectively, where $a^i = (a^i_x, a^i_y)$ and $b^i = (b^i_x, b^i_y)$. We assume without loss of generality that the bin with load a^i has smaller index (as before). Consider the square of side length ϵ with lower left point a^i (see Figure 4). The top curve must intersect either the top or right side of the square (or both), otherwise any vector would fit into this bin. Consider the right most intersection point $a^f = (a^f_x, a^f_y)$. Since f is concave, the derivative at a^f_x must be smaller than the slope of the line connecting a^i to a^f. Therefore, any vector that does not fit into this bin must have derivative at least $f'(a^f_x)$, and hence $b^i_y/b^i_x \geq f'(a^f_x)$. Let $b^f = H(a^f_x)$, and $v = (v_x, v_y)$ be a vector that was rejected from bin b^i. Let $\tilde{b} = b^i + \gamma v$ for $\gamma \in [0,1]$ be a point on the right curve. Note that $v_y/v_x \geq f'(a^f_x)$ since this vector was also rejected from bin a^i (by the First Fit property), and $b^i_y/b^i_x \geq f'(a^f_x)$. We get that $\tilde{b}_y/\tilde{b}_x \geq f'(a^f_x)$, and therefore $\tilde{b} \geq b^f$, which implies $b^i_x \geq b^f_x - \epsilon$ and $b^i_y \geq b^f_y - \epsilon$. Therefore, by volume consideration we have $T(a^i, b^i) \leq T(a^f, b^f)(1 + O(\epsilon))$.

B Linear and Piecewise Linear Functions for f-Restricted First Fit Algorithms

The f-restricted First Fit Algorithm with Linear Constraints

We first focus on f-restricted First Fit algorithms when f is a linear function. That is, $f(x) = \min\{c + a \cdot x, 1\}$. Note that the competitive ratio must be at least $\frac{1}{c}$, and a is some constant which depends on c. We then seek to optimize our choice of f among all linear functions. We describe the linear function as a pair of points $(0, c)$, $(d, 1)$. By volume consideration and by Theorem 3, it is enough to consider the points $a_1 = (0, c)$, $a_2 = (d, 1)$, and their corresponding mapped pairs $b_1 = H(0)$ $b_2 = H(d)$. The competitive ratio is given by $\max\left\{\frac{1}{c}, T((0,c), b_1), T((d,1), b_2)\right\}$. The derivative at a_2 is 0, and hence we have $H(d) = b_2 = (c, 0)$. Therefore, first we will demand that $T((d,1),(c,0)) = \frac{1}{c}$, which implies $\frac{1-d+c}{c} = \frac{1}{c}$, and hence $d = c$. Next we demand that the second

pair also satisfy $T((0,c),(x_1,y_1)) = \frac{1}{c}$, where $b_1 = (x_1, y_1)$. This implies $\frac{x_1 - y_1 + c}{x_1 \cdot c} = \frac{1}{c}$, and hence $y_1 = c$. Note that, the expression $T((0,c),(x_1,c))$ is minimized for $x_1 = 1$, and satisfies $y_1 = \frac{(1-c)}{c} \cdot x_1$ (since $(x_1, y_1) = H(0)$). Thus we get $c = \frac{(1-c)}{c}$, which implies $\frac{1}{c} = \phi = (1+\sqrt{5})/2 \approx 1.618$ is the best approximation ratio for linear constraints.

Improving the Constraint Function

The linear function that we presented contains two lines: the line that connects $(0,c)$ to $(c,1)$ and the line that connects $(c,1)$ to $(1,1)$. Instead of finding the direct representation of the optimal function, we expand the previous function to a sequence of lines that linearly approximate the optimal function in the following manner: for any sample points $P^0, P^1, ..., P^{n+1}$ such that $P^0 = (0,c)$, $P^{n+1} = (1,1)$, and $P^i \leq P^{i+1}$ for each i, we define $f(x)$ as

$$f(x) = P^i_y + \frac{P^{i+1}_y - P^i_y}{P^{i+1}_x - P^i_x}(x - P^i_x), \text{ for } P^i_x \leq x \leq P^{i+1}_x.$$

It is easy to verify that f is a monotone non-decreasing concave function, moreover the competitive ratio of the f-restricted First Fit Algorithm for this choice of f is

$$\max_x T((x, f(x)), H(x)) = \max_i T(P^i, H(P^i_x)),$$

since for $P^i_x < x < P^{i+1}_x$, $H(x) = H(P^i_x)$ and $P^i_x \leq (x, f(x))$. Define the i^{th} slope as

$$M^i = \frac{P^{i+1}_y - P^i_y}{P^{i+1}_x - P^i_x}.$$

Approximately Optimizing the Function

As mentioned, we find n sample points that approximate the optimal function f which minimizes the competitive ratio. Clearly, we should demand that the competitive ratio be equal at all points, so that for all i we have $T(P^i, H(P^i)) = \frac{1}{c}$. Therefore, we can demand that $H^R(P^i)$ be mapped to one of the sample points, namely P^{n-i}. We will have that for any i, $H^R(P^i_x) = P^{n-i}$. Therefore,

$$M^i = \frac{P^{n-i}_x}{P^{n-i}_y}. \tag{1}$$

For a given c, the question is whether we can find such a set of points such that these conditions hold. As in the linear case, we have $P^0 = (0,c)$ and $P^n = (c,1)$, and hence we can compute M^0 using Equation (1). Since $M^0 = c$, we have that $P^1 = (\epsilon, c + \epsilon \cdot M^0)$, since we are linearly approximating the function. Now we can compute P^{n-1}, since we demand that the following hold for $k = 0$:

$$T(P^{k+1}, P^{n-k-1}) = \frac{1}{c}, \tag{2}$$

$$\frac{P^{n-k}_y - P^{n-k-1}_y}{P^{n-k}_x - P^{n-k-1}_x} = M^{n-k-1} = \frac{P^{k+1}_x}{P^{k+1}_y}. \tag{3}$$

These two equations uniquely define P^{n-1} (given P^{k+1} and c, the equations are linear). We can repeat this method iteratively for any k to compute P^{k+1} and P^{n-k-1} under the assumption that we know P^k and P^{n-k}. First, we use Equation (1) to compute M^k and get $M^k = \frac{P^{n-k}_x}{P^{n-k}_y}$, and then

we compute $P^{k+1} = P^k + (\epsilon, M^k \cdot \epsilon)$. Finally, we use Equation (2) and Equation (3) to compute P^{n-k-1}. This process stops for some k' such that $P_x^{k'+1} \geq P_x^{n-k'-1}$, in which case $n = 2k'+1$. Note that we do not determine $M^{k'}$, but if $M^{k'} \geq \frac{P_x^{n-k'}}{P_y^{n-k'}} = \frac{P_x^{k'+1}}{P_y^{k'+1}}$, then the competitive ratio of the f-restricted First Fit Algorithm that uses the sample points $P^0, \ldots, P^{n+1}$ is $\frac{1}{c}$. We numerically found an approximation to the best such function (using piecewise linear functions), giving a competitive ratio of $\frac{1}{c} \approx 1.48$.

C The $\frac{4}{3}$-competitive Algorithm

C.1 Proof of Theorem 6

Our online algorithm maintains four invariants regarding the virtual bins, as given in Figure 3. Recall that we may assume, without loss of generality, that $V_1 \geq V_2$:

It is clear from invariants 3 and 4 that our algorithm is c-competitive if it is able to maintain these invariants at all times. By proceeding via an inductive argument, we assume these invariants hold, and then show that they still hold after an arbitrary vector arrives.

Case 1, i.e. $x \geq y$. As specified by our algorithm, we allocate two virtual bins, the first of which is a closed virtual bin of size y and the second of which is an open virtual bin of size $c(x-y)$. After the assignment closed virtual bin is completely occupied on both coordinates (hence, the total load among closed virtual bins on both coordinates increases by y) while the open virtual bin has zero load on the second coordinate and $x-y$ load on the first coordinate and total size of $c(x-y)$. It is easy to verify that all invariants continue to hold.

Case 2, $y \geq \frac{c}{c-1}x > x$, i.e. $x \leq (c-1)(y-x)$. In this case, since we clearly have $y > x$, the gap between V_1 and V_2 closes, and hence the main idea is to reclassify an open virtual bin of appropriate size as a closed virtual bin. Since we assume $V_1' \geq V_2'$, via our second invariant, we know that there exist an open virtual bin with a load of $y-x$ (which is occupied by a $\frac{1}{c}$-fraction of load on the first coordinate, zero on the second coordinate). This open virtual bin has a total size of $c(y-x)$, and hence has $(c-1)(y-x)$ free space (note that the second coordinate is completely empty). We put the vector (x,y) in this virtual bin (the vector (x,y) entirely fits due to our assumption that $x \leq (c-1)(y-x)$) and declare the open virtual bin as a closed virtual bin. Note that the load on both coordinates of the closed virtual bin is y, and hence the load on each coordinate occupies at least a $\frac{1}{c}$-fraction of the bin's size. Moreover, the total load among all open virtual bins decreases by $y-x$, while the total load among all closed virtual bins increases by y. Hence, all invariants continue to hold in this case.

Case 3, $x < y < \frac{c}{c-1}x$, i.e. $x > (c-1)(y-x)$. The algorithm proceeds in a manner similar to the previous case, except that we no longer have the property that the vector (x,y) entirely fits into the large closed virtual bin. To take care of this issue, the algorithm allocates an additional closed virtual bin and splits the vector (x,y) across these two closed virtual bins.

Let $f = (c-1)(\frac{y}{x}-1)$ be the fraction according to the algorithm, $x_1 = f \cdot x$, $x_2 = (1-f) \cdot x$, $y_1 = f \cdot y$, $y_2 = (1-f) \cdot y$. Note that the size of the first bin is $c(y-x)$ and the size of the second bin is y_2. Next, we show that the assignment is feasible. The open bin's load on the first coordinate after the assignment is $x_1 + (y-x) = (c-1)(\frac{y}{x}-1)x + (y-x) = c(y-x)$, and the bin's second coordinate load is $y_1 = y(c-1)(\frac{y}{x}-1) \leq c(y-x)$, since $y < \frac{c}{c-1}x$. The second bin's first coordinate load is $x_2 = (1-f)x \leq y_2$, since $x \leq y$, and finally the second bin's second coordinate load is y_2. The total size is $y_2 + c(y-x)$ and the total load across both bins on each coordinate is y. So, to maintain our invariants, we need to guarantee that y is at least a $\frac{1}{c}$-fraction of the total size. In

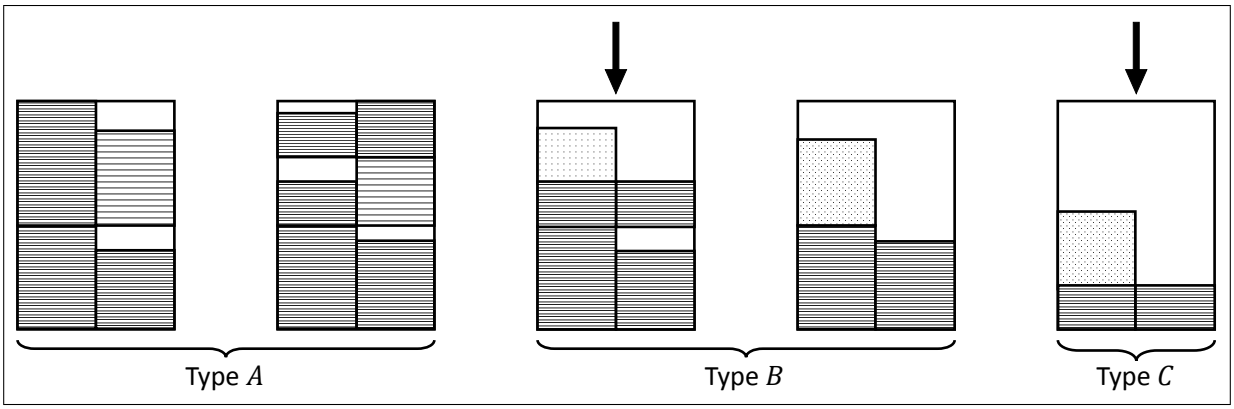

Figure 5: A snapshot during the execution of the $\frac{4}{3}$-competitive algorithm. We maintain two pointers: one for the first Type B bin that was opened, and one for the Type C bin. The striped regions denote the load on closed virtual bins, while the dotted regions represent the load on open virtual bins.

particular, we need:

$$\frac{1}{c}[c(y-x)+y_2] = \frac{1}{c}[c(y-x)+y-\frac{y}{x}(c-1)(y-x)] \leq y \Longrightarrow c \geq \frac{y^2}{y^2-yx+x^2}.$$

As long as c satisfies this constraint in this case, all invariants continue to hold. Noting that the expression is maximized for $\frac{y}{x}=2$ and attains a value of $\frac{4}{3}$ at the maximum point gives the theorem.

C.2 Implementing the $\frac{4}{3}$-competitive Algorithm on Real Bins

Assume without loss of generality that $V_1 \geq V_2$. In general, the state of the algorithm can be viewed as a sequence of various types of bins, while always maintaining two pointers to the most recently opened bin and the first available Type B bin (see Figure 5).

Algorithm 6 shows how to implement the splittable $\frac{4}{3}$-competitive algorithm that operates on virtual bins in terms of real bins.

```
1  while vector v = (x, y) arrives do
2      Let k be the index of the Type C bin
3      Let b be the index of the first Type B bin (b = k if such a bin does not exist)
4      if x ≥ y then
5          Assign v into bin k
6          C_k ← C_k + y, O_k ← O_k + c(y − x)
7      else
8          O_b ← O_b − c(y − x), C_b ← C_b + c(y − x)
9          if y ≥ c/(c−1) x then
10             Assign v into bin b
11         if x < y < c/(c−1) x then
12             Let f ← (c − 1)(y/x − 1)
13             Assign the vector f · v into bin b
14             Assign the remaining vector (1 − f) · v into bin k
15             C_k ← C_k + y(1 − f)
```

Algorithm 6: Fractional Assignment.

The algorithm ensures $0 \leq O_i + C_i \leq 1$, $O_i \geq 0$, $C_i \geq 0$, by splitting the vector appropriately and iterating. It easy to verify that this algorithm mimics Algorithm 1 and hence has the same competitive ratio, since for each bin i the values C_i, O_i exactly capture the amount of space of the corresponding virtual bins.

D Lower Bound Proof

Proof. Our lower bound will consist of two phases. In the first phase, a larger number of vectors A arrive of the form $[1, 0]$, and in the second phase $2A$ vectors arrive of the form $[\frac{1}{2}, 1]$. Suppose there is an online algorithm with competitive ratio c. We will show that it must be the case that $c \geq \frac{4}{3}$.

After the first phase, the value of OPT is precisely A, and since the algorithm is c-competitive, it cannot have more than cA bins open at the end of the first phase. In fact, we assume without loss of generality that the algorithm opens cA bins after the first phase, since if it opens less we can imagine opening exactly cA bins, some of which remain unused and the proof still goes through. Hence, among these cA bins, we only have $cA - A = A(c-1)$ free space left in the first coordinate (since in total the amount of space taken up by the vectors which arrive in the first phase is A on the first coordinate). Now consider what happens by the end of the second phase, at which point in time the value of OPT is precisely $2A$. Hence, at this point in time, the online algorithm is only allowed to open an additional cA bins. Let us consider the amount of vectors from the second phase that can fit among the first cA bins. Observe that the total space on the first coordinate among the second-phase vectors is A, and since there is at most $A(c-1)$ free space left among the first cA bins on the first coordinate, there must be at least $A - A(c-1) = A(2-c)$ space from the second-phase vectors along the first coordinate which must go into the set of bins opened by the algorithm during the second phase. Since the second-phase vectors are of the form $[\frac{1}{2}, 1]$, the second coordinate fills up at twice the rate of the first coordinate, which implies the second cA bins must be able to accommodate $2A(2-c)$ space. Hence, we have $2A(2-c) \leq cA$, which implies $c \geq \frac{4}{3}$. □

E Proofs of Section 3

E.1 Proof of Theorem 9

First, we must show that every vector that arrives is actually fully allocated (i.e., the algorithm feasibly allocates vectors). Hence, we need to argue that the density function with which we spread each infinitesimal vector δv among the infinitesimal bins within the interval $I = [V, e \cdot V]$ is a valid probability density function, where $V = \max\{V_1, \ldots, V_d\}$. This holds, since we have: $\int_V^{e \cdot V} \frac{1}{x} dx = \ln(x)|_V^{e \cdot V} = \ln\left(\frac{e \cdot V}{V}\right) = 1$.

The next thing we must ensure is that, for each dimension k and each bin b that lies on the interval $[0, \infty)$, the total load does not exceed 1 (i.e., our algorithm obeys each bin's capacity constraint). To this end, fix a dimension k and a bin $b \in [0, \infty)$. Observe that, once $V > b$, we will never allocate any fraction of an arriving vector v to bin b. Therefore, we need only concern ourselves with the total load we place on dimension k of bin b for all values $V \leq b$. We consider how much load we place on bin b when a small load of dv arrives on dimension k. Since the density at bin b is at most $\frac{1}{b}$ (as we also do not fractionally allocate a vector to b if $e \cdot V < b$), we place at most $\frac{dv}{b}$ load. Hence, the total load on dimension k of bin b is at most: $\int_0^b \frac{1}{b} dv = \frac{1}{b} \cdot b = 1$. It is easy to verify that the algorithm is e-competitive, since it never places a load on bins beyond $e \cdot V$, and hence it opens at most $e \cdot V \leq e \cdot \text{OPT}$ bins (note that V is potentially not an integer, so the number of actual bins opened will be within an additive constant of $e \cdot \text{OPT}$).

E.2 Proof of Lemma 10

To prove this lemma, we consider a variant of the algorithm above with no overflow bins. For this variant, bins will overflow on occasion. We seek to give an upper bound on the probability that a specific bin overflows. This will give a bound on the probability that a new vector v_i results in an overflow.

Suppose vector v_i is randomly assigned to a specific bin b. For a fixed coordinate k, let $a_i = v_{ik} \cdot \frac{24 \log d}{\epsilon^2}$. Let X_ℓ be the indicator random variable that is 1 if and only if vector v_ℓ is assigned to bin b. Clearly, if vector v_i does not fit into bin b due to dimension k, then we must have $X = \sum_{\ell<i} a_\ell X_\ell \geq \frac{24 \log d}{\epsilon^2} - 1$ (otherwise, vector v_i would not overflow due to dimension k). Hence, upper bounding the probability that this event occurs is sufficient. We let $\mu = \frac{24 \log(d)}{\epsilon^2 \cdot (1+\epsilon)}$ and note that $\mathbb{E}[X] = \sum_{\ell<i} a_\ell \mathbb{E}[X_\ell] = \frac{24 \log(d)}{\epsilon^2} \sum_\ell p_{\ell,b} v_{\ell k} \leq \frac{24 \log(d)}{\epsilon^2 \cdot (1+\epsilon)} = \mu$ (where the inequality follows since we simulate the splittable algorithm on bins of size $1-\epsilon$). Moreover, since $a_\ell \in [0,1]$, we can apply the Chernoff bound with $\delta = \frac{11\epsilon}{12} < 1$ (note that $(1+\delta)\mu \leq \frac{24 \log d}{\epsilon^2} - 1$) to get:

$$\Pr\left[X \geq \frac{24 \log d}{\epsilon^2} - 1\right] \leq \Pr[X \geq (1+\delta) \cdot \mu] \leq e^{-\frac{\delta^2 \mu}{3}} = e^{-\frac{\epsilon^2 \cdot 11^2 \cdot 24 \log d}{12^2 \cdot 3\epsilon^2 (1+\epsilon)}} \leq \frac{1}{d^4}.$$

Using the union bound over all dimensions bounds the probability that v_i overflows by $\frac{1}{d^3}$.

E.3 Proof of Theorem 11

By Theorem 9 and by volume consideration, the first stage of the algorithms uses $e \cdot (1+\epsilon) \cdot \text{OPT}$ bins. Using Lemma 10, we can bound the expected total load of vectors given to the First Fit algorithm. This is at most the total load times the probability that a vector overflows, which is at most $\frac{d \cdot \text{OPT}}{d^3} = \frac{\text{OPT}}{d^2}$. By Observation 3, we use at most an additional $O\left(\frac{\text{OPT}}{d^2}\right)$ bins.

E.4 Proof of Theorem 12

If all vectors are smaller than $\frac{\epsilon^2}{60 \log d}$, we show a $(1+\tilde{\epsilon})e$-competitive algorithm where $\tilde{\epsilon} = \epsilon \cdot (1 + \log(\frac{1}{\epsilon}))$. We multiply each coordinate by $Q = \frac{60 \log(d)}{\epsilon^2}$ ($\epsilon \leq 0.5$). For ease of presentation we still use v as the transformed vector (i.e., originally, all entries in v were small, but after this transformation, all entries are in $[0,1]$). We assume that for all vectors i and dimensions $k \in [d]$, if $v_{ik} > 0$ then $v_{ik} \geq \max_k \frac{v_{ik}}{d^2}$ (we show how to omit this assumption later).

We fix the index i of vector v_i, dimension $k \in [d]$, and an open bin j. Let $f(x) = \alpha^x$ where $\alpha = e^{\epsilon/2}$, and let $L^i_{j,k} = \sum_{i \in V[j]} v_{ik}$, where $V[j]$ is the set of vectors that are (virtually) assigned to bin j. By virtual, we mean that each vector is assigned to a bin, even if it overflows. Note that $L^i_{j,k}$ is the load on bin j of dimension k in the virtual setting after vectors $v_1, \ldots, v_i$ have arrived. The algorithm uses the following potential function, where A is the current set of open bins (which may change and does not include the spillover bins opened by the First Fit algorithm):

$$\Phi^i_A = \sum_{j \in A} \sum_{k \in [d]} \Phi^i_{j,k}$$

$$\Phi^i_{j,k} = f(L^i_{j,k} - \alpha \cdot \sum_{i' \leq i} p_{i',j} v_{ik}).$$

```
while vector v_i arrives do
    Simulate Algorithm 3 on (1 − ε̃) sized bins
    Let p_{i,j} be the fractional assignment of vector i to bin j
    Assign vector v_i to the bin j* which minimizes the potential Φ^i_{{j|p_{i,j}>0}}
    if vector i fits into bin j* then
        Assign vector i to bin j*
    else
        Assign vector i using the First Fit algorithm to spillover bins
```

Algorithm 7: Sliding Window Assignment - Unsplittable Determinstic.

Observe that a vector v_i fits into bin j if $L^i_{j,k} + v_{ik} \leq Q$ for every $k \in [d]$, and $\sum_i p_{i,j} v_{ik} \leq \frac{Q}{1+\tilde{\epsilon}}$ since Algorithm 3 gives a feasible assignment to bins of size $(1-\tilde{\epsilon})$. Our main lemma bounds the total volume that is passed to the second phase First Fit algorithm. First, we give the following lemma.

Lemma 13. *Φ^i_A is non-decreasing in i, where $A = \{j | p_{i,j} > 0\}$.*

Proof. We use a probabilistic argument to prove that there exists a bin j which does not increase the potential function. Assume that we fix the value of Φ^{i-1}_A and assign vector v_i to bin j according to the probability $p_{i,j}$. We will show that $\mathbb{E}[\Phi^i_A] \leq \Phi^{i-1}_A$. This implies that there exists a bin j which minimizes the potential. For any fixed i, j, k we get:

$$\begin{aligned}
\mathbb{E}[\Phi^i_{j,k}] &= p_{i,j} f(L^{i-1}_{j,k} + v_{ik} - \alpha \cdot \sum_{i' \leq i-1} p_{i',j} v_{ik} - \alpha p_{i,j} v_{ik}) \\
&+ (1 - p_{i,j}) f(L^{i-1}_{j,k} - \alpha \cdot \sum_{i' \leq i-1} p_{i',j} v_{ik} - \alpha p_{i',j} v_{ik}) \\
&= \Phi^{i-1}_{j,k} \cdot \alpha^{-\alpha p_{i,j} v_{ik}} \cdot (p_{i,j}(\alpha^{v_{ik}} - 1) + 1) \\
&\leq \Phi^{i-1}_{j,k} \cdot \alpha^{-\alpha p_{i,j} v_{ik}} \cdot (p_{i,j} v_{ik}(\alpha - 1) + 1) \\
&\leq \Phi^{i-1}_{j,k} \cdot \exp\left(-\alpha \log \alpha \cdot p_{i,j} v_{ik}\right) \cdot \exp\left(p_{i,j} v_{ik}(\alpha - 1)\right) \\
&\leq \Phi^{i-1}_{j,k},
\end{aligned}$$

where the first inequality follows from $a^x - 1 \leq x(a-1)$ for any $a \geq 1$ and $x \in [0,1]$, the second inequality follows from $1 + x \leq e^x$, and the last inequality follows from $\alpha \log \alpha \geq \alpha - 1$ for $\alpha \geq 1$. □

Let I^s denotes the indices of vectors assigned by the First Fit algorithm, and let V^s be the total volume of these vectors, namely $V^s = \sum_{i \in I^s} \sum_{k \in [d]} v_{ik}$.

Lemma 14. *At any point in the execution of Algorithm 7, we have $V^s \leq \frac{2 \cdot e \cdot \mathrm{OPT}}{d^2}$.*

Proof. Note that, when vector v_i arrives, if $p_{i,j} = 0$ then $\Phi^i_{j,k}$ does not change. Using Lemma 13 and the observation that $\Phi^0_{j,k} = 1$, we conclude that

$$\Phi^i_A \leq e \cdot d \cdot \mathrm{OPT}. \tag{4}$$

Next, as in the randomized algorithm, we bound the total volume of vectors given to the First Fit algorithm, namely V^s. For each $i \in I^s$, define $k = k(i)$ to be an arbitrary dimension k with $L^i_{j,k} > Q$ (such a dimension k exists since v_i was added to I^s).

$$\begin{aligned} V^s &= \sum_{i\in I^s}\sum_{k'\in[d]} v_{ik'} \\ &= \sum_{j\in A}\sum_{k\in[d]}\sum_{i\in V[j]\cap I^s}\sum_{k'\in[d]} v_{ik'} \\ &\le \sum_{j\in A}\sum_{k\in[d]}\sum_{i\in V[j]\cap I^s} d^3 v_{ik} \\ &\le d^3\sum_{j\in A}\sum_{k\in[d]}(L^i_{j,k}-(Q-1))_+ \end{aligned}$$

where the first inequality follows from the assumption that $v_{ik} \ge \frac{\max_k v_{ik}}{d^2}$, and the second inequality follows since $v_{ik} \le 1$. Note that we define $(x)_+ = \max\{x, 0\}$. Let $Q' = \frac{Q}{1+\tilde{\epsilon}}$, we claim that for all i, j, k:

$$\Phi^i_{j,k} \ge 0.5 \cdot d^6 (L_{j,k} - Q + 1)_+ \tag{5}$$

This inequality trivially holds if $L^i_{j,k} \le Q-1$ since $\Phi^i_{j,k}$ is always non-negative. Otherwise, we have

$$\begin{aligned} \Phi^i_{j,k} &\ge \alpha^{L^i_{j,k}-\alpha Q'} \\ &\ge \alpha^{L^i_{j,k}-(1+\epsilon)Q'+0.4\epsilon Q'} \\ &\ge \alpha^{L^i_{j,k}-(1+\epsilon)Q'+0.2\epsilon Q} \\ &= d^6\alpha^{L^i_{j,k}-(1+\epsilon)Q'} \\ &\ge 0.5\cdot d^6(L^i_{j,k}-(1+\epsilon)Q')_+ \\ &\ge 0.5\cdot d^6(L^i_{j,k}-Q+1)_+ \end{aligned}$$

where the first inequality follows from $\sum_i p_{i,j} v_{ik} \le \frac{Q}{1+\tilde{\epsilon}} = Q'$ for every k, j, the second inequality follows from $e^{\epsilon/2} \le 1+0.6\epsilon$ for $\epsilon \le 0.5$, the third inequality follows by definition of Q' and $1+\tilde{\epsilon} \le 2$, the fourth inequality follows since $L^i_{j,k} - (1+\epsilon)Q' \ge \frac{2\log(\frac{1}{\epsilon})}{\epsilon}$ and $e^{\frac{\epsilon x}{2}} \ge \frac{x}{2}$ for $x \ge \frac{4\log(\frac{1}{\epsilon})}{\epsilon}$.

Finally we bound V^s using Equation (4) and Equation (5):

$$V^s \le d^3\sum_{j\in A}\sum_{k\in[d]}(L_{j,k}-Q+1)_+ \le \frac{2d^3}{d^6}\sum_{j\in[m]}\sum_{k\in[d]}\Phi_{j,k} \le \frac{2\cdot e\cdot \mathrm{OPT}}{d^2}$$

□

We now conclude the proof of Theorem 12. Similarly to the randomized case, by Theorem 9 and by volume consideration, the first stage of the algorithm uses $e\cdot(1+\tilde{\epsilon})\cdot \mathrm{OPT}$ bins. Using Lemma 14 and Observation 3, we conclude that we open at most an additional factor of $(1+o(1))$ bins due to the First Fit algorithm. Finally, we may omit the assumption that for all i and for all $k \in [d]$ such that $v_{ik} > 0$ we have $v_{ik} \ge \frac{\max_k v_{ik}}{d^2}$. We do this by imagining all values v_{ik} that are strictly less than $\frac{\max_k v_{ik}}{d^2}$ are "set" to zero. This incurs an additional factor of $\left(1+\frac{1}{d}\right)$ in the competitive ratio, since we need to resize bins in the simulation of Algorithm 3 in order to guarantee a feasible assignment.

Using these techniques above we conclude:

Remark 1. *There exists a $1+O(\epsilon)$ deterministic algorithm for the online d-dimensional Load Balancing problem when vectors are small.*