Streaming Algorithms for Bin Packing and Vector Scheduling

Graham Cormode and Pavel Veselý

Department of Computer Science, University of Warwick, Coventry, UK.
{G.Cormode,Pavel.Vesely}@warwick.ac.uk.

Abstract

Problems involving the efficient arrangement of simple objects, as captured by bin packing and makespan scheduling, are fundamental tasks in combinatorial optimization. These are well understood in the traditional online and offline cases, but have been less well-studied when the volume of the input is truly massive, and cannot even be read into memory. This is captured by the streaming model of computation, where the aim is to approximate the cost of the solution in one pass over the data, using small space. As a result, streaming algorithms produce concise input summaries that approximately preserve the optimum value.

We design the first efficient streaming algorithms for these fundamental problems in combinatorial optimization. For BIN PACKING, we provide a streaming asymptotic $1+\varepsilon$-approximation with $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon}\right)$ memory, where $\widetilde{\mathcal{O}}$ hides logarithmic factors. Moreover, such a space bound is essentially optimal. Our algorithm implies a streaming $d+\varepsilon$-approximation for VECTOR BIN PACKING in d dimensions, running in space $\widetilde{\mathcal{O}}\left(\frac{d}{\varepsilon}\right)$. For the related VECTOR SCHEDULING problem, we show how to construct an input summary in space $\widetilde{\mathcal{O}}(d^2 \cdot m/\varepsilon^2)$ that preserves the optimum value up to a factor of $2-\frac{1}{m}+\varepsilon$, where m is the number of identical machines.

1 Introduction

The streaming model captures many scenarios when we must process very large volumes of data, which cannot fit into the working memory. The algorithm makes one or more passes over the data with a limited memory, but does not have random access to the data. Thus, it needs to extract a concise summary of the huge input, which can be used to approximately answer the problem under consideration. The main aim is to provide a good trade-off between the space used for processing the input stream (and hence, the summary size) and the accuracy of the (best possible) answer computed from the summary. Other relevant parameters are the time and space needed to make the estimate, and the number of passes, ideally equal to one.

While there have been many effective streaming algorithms designed for a range of problems in statistics, optimization, and graph algorithms (see surveys by Muthukrishnan [38] and McGregor [37]), there has been little attention paid to the core problems of packing and scheduling. These are fundamental abstractions, which form the basis of many generalizations and extensions [14, 13]. In this work, we present the first efficient algorithms for packing and scheduling that work in the streaming model.

A first conceptual challenge is to resolve what form of answer is desirable in this setting. If items in the input are too many to store, then it is also unfeasible to require a streaming algorithm to provide an explicit description of how each item is to be handled. Rather, our objective is for the algorithm to provide the cost of the solution, in the form of the number of bins or the duration of the schedule. Moreover, many of our algorithms can provide a concise *description* of the solution, which describes in outline how the jobs are treated in the design.

A second issue is that the problems we consider, even in their simplest form, are NP-hard. The additional constraints of streaming computation do not erase the computational challenge. In some cases, our algorithms proceed by adopting and extending known polynomial-time approximation schemes for the offline versions of the problems, while in other cases, we come up with new approaches. The streaming model effectively emphasizes the question of how compactly can the input be summarized to allow subsequent approximation of the problem of interest. Our main results show that in fact the inputs for many of our problems of interest can be "compressed" to very small intermediate descriptions which suffice to extract near-optimal solutions for the original input. This implies that they can be solved in scenarios which are storage or communication constrained.

We proceed by formalizing the streaming model, after which we summarize our results. We continue by presenting related work, and contrast with the online setting.

1.1 Problems and Streaming Model

Bin packing. The BIN PACKING problem is defined as follows: The input consists of N items with sizes $s_1, \ldots, s_N$ (each between 0 and 1), which need to be packed into bins of unit capacity. That is, we seek a partition of the set of items $\{1, \ldots, N\}$ into subsets $B_1, \ldots, B_m$, called bins, such that for any bin B_i, it holds that $\sum_{j \in B_i} s_j \leq 1$. The goal is to minimize the number m of bins used.

We also consider the natural generalization to VECTOR BIN PACKING, where the input consists of d-dimensional vectors, with the value of each coordinate between 0 and 1 (i.e., the scalar items s_i are replaced with vectors $\mathbf{v^i}$). The vectors need to be packed into d-dimensional bins with unit capacity in each dimension, we thus require that $\| \sum_{\mathbf{v} \in B_i} \mathbf{v} \|_\infty \leq 1$ (where the infinity norm $\|\mathbf{v}\|_\infty = \max_i \mathbf{v}_i$).

Scheduling. The MAKESPAN SCHEDULING problem is closely related to BIN PACKING but, instead of filling bins with bounded capacity, we try to balance the loads assigned to a fixed number of bins. Now we refer to the input as comprising a set of *jobs*, with each job j defined by its processing time p_j. Our goal is to assign each job on one of m identical machines to minimize the *makespan*, which is the maximum load over all machines.

In VECTOR SCHEDULING, a job is described not only by its processing time, but also by, say, memory or bandwidth requirements. The input is thus a set of jobs, each job j characterized by a vector $\mathbf{v^j}$. The goal is to assign each job into one of m identical machines such that the maximum load over all machines and dimensions is minimized.

Streaming model. In the streaming scenario, the algorithm receives the input as a sequence of items, called the input stream. We do not assume that the stream is ordered in any particular way (e.g., randomly or by item sizes), so our algorithms must work for arbitrarily ordered streams. The items arrive one by one and upon receiving each item, the algorithm updates its memory state. A streaming algorithm is required to use space sublinear in the length of the stream, ideally just $\mathrm{polylog}(N)$, while it processes the stream. After the last item arrives, the algorithm computes its estimate of the optimal value, and the space or time used during this final computation is not restricted.

For many natural optimization problems outputting some explicit solution of the problem is not possible owing to the memory restriction (as the algorithm can store only a small subset of items). Thus the goal is to find a good approximation of the *value* of an offline optimal solution. Since our model does not assume that item sizes are integers, we express the space complexity not in bits, but in words (or memory cells), where each word can store any number from the input; a linear combination of numbers from the input; or any integer with $\mathcal{O}(\log N)$ bits (for counters, pointers, etc.).

1.2 Our Results

Bin packing. In Section 3, we present a streaming algorithm for BIN PACKING, which outputs an asymptotic $1+\varepsilon$-approximation of OPT, the optimal number of bins, using $\mathcal{O}\left(\frac{1}{\varepsilon}\cdot\log\frac{1}{\varepsilon}\cdot\log\mathsf{OPT}\right)$ memory.[1] This means that the algorithm uses at most $(1+\varepsilon)\cdot\mathsf{OPT}+o(\mathsf{OPT})$ bins, and in our case, the additive $o(\mathsf{OPT})$ term is bounded by the space used. The novelty of our contribution is to combine a data structure that approximately tracks all quantiles in a numeric stream [26] with techniques for approximation schemes [18, 33]. We show that we can improve upon the $\log\mathsf{OPT}$ factor in the space complexity if randomization is allowed or if item sizes are drawn from a bounded-size set of real numbers. On the other hand, we argue that our result is close to optimal, up to a factor of $\mathcal{O}\left(\log\frac{1}{\varepsilon}\right)$, if item sizes are accessed only by comparisons (including comparisons with some fixed constants). Thus, one cannot get an estimate with at most $\mathsf{OPT}+o(\mathsf{OPT})$ bins by a streaming algorithm, unlike in the offline setting [28]. The hardness emerges from the space complexity of the quantiles problem in the streaming model.

For VECTOR BIN PACKING, we design a streaming asymptotic $d+\varepsilon$-approximation algorithm running in space $\mathcal{O}\left(\frac{d}{\varepsilon}\cdot\log\frac{d}{\varepsilon}\cdot\log\mathsf{OPT}\right)$; see Appendix B. We remark that if vectors are rounded into a sublinear number of types, then better than d-approximation is not possible [7].

Scheduling. For MAKESPAN SCHEDULING, one can obtain a straightforward streaming $1+\varepsilon$-approximation[2] with space of only $\mathcal{O}(\frac{1}{\varepsilon}\cdot\log\frac{1}{\varepsilon})$ by rounding sizes of suitably large jobs to powers of $1+\varepsilon$ and counting the total size of small jobs. In a higher dimension, it is also possible to get a streaming $1+\varepsilon$-approximation, by the rounding introduced by Bansal *et al.* [8]. However, the memory required for this algorithm is exponential in d, precisely of size $\mathcal{O}\left(\left(\frac{1}{\varepsilon}\log\frac{d}{\varepsilon}\right)^d\right)$, and thus only practical when d is a very small constant. Moreover, such a huge amount of memory is needed even if the number m of machines (and hence, of big jobs) is small as the algorithm rounds small jobs into exponentially many types. See Appendix D for more details.

In case m and d make this feasible, we design a new streaming $\left(2-\frac{1}{m}+\varepsilon\right)$-approximation with $\mathcal{O}\left(\frac{1}{\varepsilon^2}\cdot d^2\cdot m\cdot\log\frac{d}{\varepsilon}\right)$ memory, which implies a 2-approximation streaming algorithm running in space $\mathcal{O}(d^2\cdot m^3\cdot\log dm)$. We thus obtain a much better approximation than for VECTOR BIN PACKING with a reasonable amount of memory (although to compute the actual makespan from our input summary, it takes time doubly exponential in d [8]). Our algorithm is not based on rounding, as in the aforementioned algorithms, but on combining small jobs into containers, and the approximation guarantee of this approach is at least $2-\frac{1}{m}$, which we demonstrate by an example. We describe the algorithm in Section 4.

2 Related Work

We give an overview of related work in offline, online, and sublinear algorithms, and highlight the differences between online and streaming algorithms. Recent surveys of Christensen *et al.* [13] and Coffman *et al.* [14] have a more comprehensive overview.

2.1 Bin Packing

Offline approximation algorithms. BIN PACKING is an NP-complete problem and indeed it is NP-hard even to decide whether two bins are sufficient or at least three bins are necessary. This follows by a simple

[1] We remark that some online algorithms can be implemented in the streaming model, as described in Section 2.1, but they give worse approximation guarantees.

[2] Unlike for BIN PACKING, an additive constant or even an additive $o(\mathsf{OPT})$ term does not help in the definition of the approximation ratio, since we can scale every number on input by any $\alpha>0$ and OPT scales by α as well.

reduction from the PARTITION problem and presents the strongest inapproximability to date. Most work in the offline model focused on providing *asymptotic* R-approximation algorithms, which use at most $R \cdot \mathsf{OPT} + o(\mathsf{OPT})$ bins. In the following, when we refer to an approximation for BIN PACKING we implicitly mean the asymptotic approximation. The first *polynomial-time approximation scheme* (PTAS), that is, a $1+\varepsilon$-approximation for any $\varepsilon > 0$, was given by Fernandez de la Vega and Lueker [18]. Karmarkar and Karp [33] provided an algorithm which returns a solution with $\mathsf{OPT} + \mathcal{O}(\log^2 \mathsf{OPT})$ bins. Recently, Hoberg and Rothvoß [28] proved it is possible to find a solution with $\mathsf{OPT} + \mathcal{O}(\log \mathsf{OPT})$ bins in polynomial time.

The input for BIN PACKING can be described by N numbers, corresponding to item sizes. While in general these sizes may be distinct, in some cases the input description can be compressed significantly by specifying the number of items of each size in the input. Namely, in the HIGH-MULTIPLICITY BIN PACKING problem, the input is a set of pairs $(a_1, s_1), \ldots, (a_\sigma, s_\sigma)$, where for $i = 1, \ldots, \sigma$, a_i is the number of items of size s_i (and all s_i's are distinct). Thus, σ encodes the number of item sizes, and hence the size of the description. The goal is again to pack these items into bins, using as few bins as possible. For constant number of sizes, σ, Goemans and Rothvoß [24] recently gave an exact algorithm for the case of rational item sizes running in time $(\log \Delta)^{2^{\mathcal{O}(\sigma)}}$, where Δ is the largest multiplicity of an item or the largest denominator of an item size, whichever is the greater.

While these algorithms provide satisfying theoretical guarantees, simple heuristics are often adopted in practice to provide a "good-enough" performance. FIRST FIT [32], which puts each incoming item into the first bin where it fits and opens a new bin only when the item does not fit anywhere else achieves 1.7-approximation [16]. For the high-multiplicity variant, using an LP-based Gilmore-Gomory cutting stock heuristic [22, 23] gives a good running time in practice [2] and produces a solution with at most $\mathsf{OPT} + \sigma$ bins. However, neither of these algorithms adapts well to the streaming setting with possibly distinct item sizes. For example, FIRST FIT has to remember the remaining capacity of each open bin, which in general can require space proportional to OPT.

VECTOR BIN PACKING proves to be substantially harder to approximate, even in a constant dimension. For fixed d, Bansal, Eliáš, and Khan [7] showed an approximation factor of $\approx 0.807 + \ln(d+1) + \varepsilon$. For general d, a relatively simple algorithm based on an LP relaxation, due to Chekuri and Khanna [11], remains the best known, with an approximation guarantee of $1 + \varepsilon d + \mathcal{O}(\log \frac{1}{\varepsilon})$. The problem is APX-hard even for $d = 2$ [40], and cannot be approximated within a factor better than $d^{1-\varepsilon}$ for any fixed $\varepsilon > 0$ [13] if d is arbitrarily large. Hence, our streaming $d + \varepsilon$-approximation for VECTOR BIN PACKING asymptotically achieves the offline lower bound.

Sampling-based algorithms. Sublinear-time approximation schemes constitute a model related to, but distinct from, streaming algorithms. Batu, Berenbrink, and Sohler [9] provide an algorithm that takes $\widetilde{\mathcal{O}}\left(\sqrt{N} \cdot \mathrm{poly}(\frac{1}{\varepsilon})\right)$ weighted samples, meaning that the probability of sampling an item is proportional to its size. It outputs an asymptotic $1+\varepsilon$-approximation of OPT. If uniform samples are also available, then sampling $\widetilde{\mathcal{O}}\left(N^{1/3} \cdot \mathrm{poly}(\frac{1}{\varepsilon})\right)$ items is sufficient. These results are tight, up to a $\mathrm{poly}(\frac{1}{\varepsilon}, \log N)$ factor. Later, Beigel and Fu [10] focused on uniform sampling of items, proving that $\widetilde{\Theta}(N/\mathsf{SIZE})$ samples are sufficient and necessary, where SIZE is the total size of all items. Their approach implies a streaming approximation scheme by uniform sampling of the substream of big items. However, the space complexity in terms of $\frac{1}{\varepsilon}$ is not stated in the paper, but we calculate this to be $\Omega\left(\varepsilon^{-c}\right)$ for a constant $c \geq 10$. Moreover, $\Omega(\frac{1}{\varepsilon^2})$ samples are clearly needed to estimate the number of items with size close to 1. Note that our approach is deterministic and substantially different than taking a random sample from the stream.

Online algorithms. Online and streaming algorithms are similar in the sense that they are required to process items one by one. However, an online algorithm must make all its decisions immediately — it must fix the placement of each incoming item on arrival.[3] A streaming algorithm can postpone such decisions to

[3]Relaxations which allow a limited amount of "repacking" have also been considered [17].

the very end, but is required to keep its memory small, whereas an online algorithm may remember all items that have arrived so far. Hence, online algorithms apply in the streaming setting only when they have small space cost, including the space needed to store the solution constructed so far. The approximation ratio of online algorithms is quantified by the *competitive ratio*.

For BIN PACKING, the best possible competitive ratio is substantially worse than what we can achieve offline or even in the streaming setting. Balogh *et al.* [5] designed an asymptotically 1.5783-competitive algorithm, while the current lower bound on the asymptotic competitive ratio is 1.5403 [6]. This (relatively complicated) online algorithm is based on the HARMONIC algorithm [35], which for some integer K classifies items into size groups $(0, \frac{1}{K}], (\frac{1}{K}, \frac{1}{K-1}], \ldots, (\frac{1}{2}, 1]$. It packs each group separately by NEXT FIT, keeping just one bin open, which is closed whenever the next item does not fit. Thus HARMONIC can run in memory of size K and be implemented in the streaming model, unlike most other online algorithms which require maintaining the levels of all bins opened so far. Its competitive ratio tends to approximately 1.691 as K goes to infinity. Surprisingly, this is also the best possible ratio if only a bounded number of bins is allowed to be open for an online algorithm [35], which can be seen as the intersection of online and streaming models.

For VECTOR BIN PACKING, the best known competitive ratio of $d + 0.7$ [20] is achieved by FIRST FIT. A lower bound of $\Omega(d^{1-\varepsilon})$ on the competitive ratio was shown by Azar *et al.* [3]. It is thus currently unknown whether or not online algorithms outperform streaming algorithms in the vector setting.

2.2 Scheduling

Offline approximation algorithms. MAKESPAN SCHEDULING is strongly NP-complete [21], which in particular rules out the possibility of a PTAS with time complexity $\mathrm{poly}(\frac{1}{\varepsilon}, n)$. After a sequence of improvements, Jansen, Klein, and Verschae [31] gave a PTAS with time complexity $2^{\widetilde{\mathcal{O}}(1/\varepsilon)} + \mathcal{O}(n \log n)$, which is essentially tight under the Exponential Time Hypothesis (ETH) [12].

For constant dimension d, VECTOR SCHEDULING also admits a PTAS, as shown by Chekuri and Khanna [11]. However, the running time is of order $n^{(1/\varepsilon)^{\widetilde{\mathcal{O}}(d)}}$. The approximation scheme for a fixed d was improved to an efficient PTAS, namely to an algorithm running in time $2^{(1/\varepsilon)^{\widetilde{\mathcal{O}}(d)}} + \mathcal{O}(dn)$, by Bansal *et al.* [8], who also showed that the running time cannot be significantly improved under ETH. In contrast our streaming $\mathrm{poly}(d, m)$-space algorithm computes an input summary maintaining 2-approximation of the original input. This respects the lower bound, since to compute the actual makespan from the summary, we still need to execute an offline algorithm, with running time doubly exponential in d. The best known approximation ratio for large d is $\mathcal{O}(\log d/(\log \log d))$ [27, 30], while α-approximation is not possible in polynomial time for any constant $\alpha > 1$ and arbitrary d, unless NP = ZPP.

Online algorithms. For the scalar problem, the optimal competitive ratio is known to lie in the interval $(1.88, 1.9201)$ [1, 25, 29, 19], which is substantially worse than what can be done by a simple streaming $1 + \varepsilon$-approximation in space $\mathcal{O}(\frac{1}{\varepsilon} \cdot \log \frac{1}{\varepsilon})$. Interestingly, for VECTOR SCHEDULING, the algorithm by Im *et al.* [30] with ratio $\mathcal{O}(\log d/(\log \log d))$ actually works in the online setting as well and needs space $\mathcal{O}(d \cdot m)$ only during its execution (if the solution itself is not stored), which makes it possible to implement it in the streaming setting. This online ratio cannot be improved as there is a lower bound of $\Omega(\log d/(\log \log d))$ [30, 4], whereas in the streaming setting we can achieve a 2-approximation with a reasonable memory (or even $1 + \varepsilon$-approximation for a fixed d). If all jobs have sufficiently small size, we improve the analysis in [30] and show that the online algorithm achieves $1 + \varepsilon$-approximation; see Section 4.

3 Bin Packing

Notation. For an instance I, let $N(I)$ be the number of items in I, let $\mathsf{SIZE}(I)$ be the total size of all items in I, and let $\mathsf{OPT}(I)$ be the number of bins used in an optimal solution for I. Clearly, $\mathsf{SIZE}(I) \leq \mathsf{OPT}(I)$. For a bin B, let $s(B)$ be the total size of items in B. For a given $\varepsilon > 0$, we use $\widetilde{\mathcal{O}}(f(\frac{1}{\varepsilon}))$ to hide factors logarithmic in $\frac{1}{\varepsilon}$ and $\mathsf{OPT}(I)$, i.e., to denote $\mathcal{O}\big(f(\frac{1}{\varepsilon}) \cdot \operatorname{polylog} \frac{1}{\varepsilon} \cdot \operatorname{polylog} \mathsf{OPT}(I)\big)$.

Overview. We first briefly describe the approximation scheme of Fernandez de la Vega and Lueker [18], whose structure we follow in outline. Let I be an instance of BIN PACKING. Given a precision requirement $\varepsilon > 0$, we say that an item is *small* if its size is at most ε; otherwise, it is *big*. Note that there are at most $\frac{1}{\varepsilon}\mathsf{SIZE}(I)$ big items. The rounding scheme in [18], called "linear grouping", works as follows: We sort the big items by size non-increasingly and divide them into groups of $k = \lfloor \varepsilon \cdot \mathsf{SIZE}(I) \rfloor$ items (the first group thus contains the k biggest items). In each group, we round up the sizes of all items to the size of the biggest item in that group. It follows that the number of groups and thus the number of distinct item sizes (after rounding) is bounded by $\lceil \frac{1}{\varepsilon^2} \rceil$. Let I_{R} be the instance of HIGH-MULTIPLICITY BIN PACKING consisting of the big items with rounded sizes. It can be shown that $\mathsf{OPT}(I_{\mathrm{B}}) \leq \mathsf{OPT}(I_{\mathrm{R}}) \leq (1+\varepsilon) \cdot \mathsf{OPT}(I_{\mathrm{B}})$, where I_{B} is the set of big items in I (we detail a similar argument in Section 3.1). Due to the bounded number of distinct item sizes, we can find a close-to-optimal solution for I_{R} efficiently. We then translate this solution into a packing for I_{B} in the natural way. Finally, small items are filled greedily (e.g., by First Fit) and it can be shown that the resulting complete solution for I is a $1 + \mathcal{O}(\varepsilon)$-approximation.

Karmarkar and Karp [33] proposed an improved rounding scheme, called "geometric grouping". It is based on the observation that item sizes close to 1 should be approximated substantially better than item sizes close to ε. We present a version of such a rounding scheme in Section 3.1.

Our algorithm follows a similar outline with two stages (rounding and finding a solution for the rounded instance), but working in the streaming model brings two challenges: First, in the rounding stage, we need to process the stream of items and output a rounded high-multiplicity instance with few item sizes that are not too small, while keeping only a small number of items in the memory. Second, the rounding of big items needs to be done carefully so that not much space is "wasted", since in the case when the total size of small items is relatively large, we argue that our solution is close to optimal by showing that the bins are nearly full on average.

Input summary properties. More precisely, we fix some $\varepsilon > 0$ that is used to control the approximation guarantee. During the first stage, our algorithm has one variable which accumulates the total size of all small items in the input stream, i.e., those of size at most ε. Let I_{B} be the substream consisting of all big items. We process I_{B} and output a rounded high-multiplicity instance I_{R} with the following properties:

- (P1) There are at most σ item sizes in I_{R}, all of them larger than ε, and the memory required for processing I_{B} is $\mathcal{O}(\sigma)$.
- (P2) The i-th biggest item in I_{R} is at least as large as the i-th biggest item in I_{B} (and the number of items in I_{R} is the same as in I_{B}). This immediately implies that any packing of I_{R} can be used as a packing of I_{B} (in the same number of bins), so $\mathsf{OPT}(I_{\mathrm{B}}) \leq \mathsf{OPT}(I_{\mathrm{R}})$, and moreover, $\mathsf{SIZE}(I_{\mathrm{B}}) \leq \mathsf{SIZE}(I_{\mathrm{R}})$.
- (P3) $\mathsf{OPT}(I_{\mathrm{R}}) \leq (1+\varepsilon) \cdot \mathsf{OPT}(I_{\mathrm{B}}) + \mathcal{O}(\log \frac{1}{\varepsilon})$.
- (P4) $\mathsf{SIZE}(I_{\mathrm{R}}) \leq (1+\varepsilon) \cdot \mathsf{SIZE}(I_{\mathrm{B}})$.

In words, (P2) means that we are rounding item sizes up and, together with (P3), it implies that the optimal solution for the rounded instance approximates $\mathsf{OPT}(I_{\mathrm{B}})$ well. The last property is used in the case when the total size of small items constitutes a large fraction of the total size of all items. Note that $\mathsf{SIZE}(I_{\mathrm{R}}) - \mathsf{SIZE}(I_{\mathrm{B}})$ can be thought of as bin space "wasted" by rounding.

Observe that the succinctness of the rounded instance depends on σ. First, we show a streaming algorithm for rounding with $\sigma = \widetilde{\mathcal{O}}(\frac{1}{\varepsilon^2})$. Then we improve upon it and give an algorithm with $\sigma = \widetilde{\mathcal{O}}(\frac{1}{\varepsilon})$, which is essentially the best possible, while guaranteeing an error of $\varepsilon \cdot \mathsf{OPT}(I_{\mathrm{B}})$ introduced by rounding (elaborated

on in Section 3.2). More precisely, we show the following:

Lemma 1. *Given a steam I_{B} of big items, there is a deterministic streaming algorithm that outputs a* HIGH-MULTIPLICITY BIN PACKING *instance satisfying (P1)-(P4) with $\sigma = \mathcal{O}\left(\frac{1}{\varepsilon} \cdot \log \frac{1}{\varepsilon} \cdot \log \mathsf{OPT}(I_{\mathrm{B}})\right)$.*

Before describing the rounding itself and proving Lemma 1, we explain how to use it to calculate an accurate estimate of the number of bins.

Calculating a bound on the number of bins after rounding. First, we obtain a solution $\mathcal{S}$ of the rounded instance I_{R}. For instance, we may round the solution of the linear program introduced by Gilmore and Gomory [22, 23], and get a solution with at most $\mathsf{OPT}(I_{\mathrm{R}}) + \sigma$ bins. Or, if item sizes are rational numbers, we may compute an optimal solution for I_{R} by the algorithm of Goemans and Rothvoß [24]; however, the former approach appears to be more efficient and more general. In the following, we thus assume that $\mathcal{S}$ uses at most $\mathsf{OPT}(I_{\mathrm{R}}) + \sigma$ bins.

We now calculate a bound on the number of bins in the original instance. Let W be the total free space in the bins of $\mathcal{S}$ that can be used for small items. To be precise, W equals the sum over all bins B in $\mathcal{S}$ of $\max(0, 1 - \varepsilon - s(B))$. Note that the capacity of bins is capped at $1 - \varepsilon$, because it may happen that all small items are of size ε while the packing leaves space of just under ε in any bin. Then we would not be able to pack small items into these bins. Reducing the capacity by ε removes this issue. On the other hand, if a small item does not fit into a bin, then the remaining space in the bin is smaller than ε.

Let s be the total size of all small items in the input stream. If $s \leq W$, then all small items surely fit into the free space of bins in $\mathcal{S}$ (and can be assigned there greedily by FIRST FIT). Consequently, we output that the number of bins needed for the stream of items is at most $|\mathcal{S}|$, i.e., the number of bins in solution $\mathcal{S}$ for I_{R}. Otherwise, we need to place small items of total size at most $s' = s - W$ into new bins and it is easy to see that opening at most $\lceil s'/(1-\varepsilon) \rceil \leq (1 + \mathcal{O}(\varepsilon)) \cdot s' + 1$ bins for these small items suffices. Hence, in the case $s > W$, we output that $|\mathcal{S}| + \lceil s'/(1-\varepsilon) \rceil$ bins are sufficient to pack all items in the stream.

We prove that the number of bins that we output in either case is a good approximation of the optimal number of bins, provided that $\mathcal{S}$ is a good solution for I_{R} (proof deferred to Appendix A.2).

Lemma 2. *Let I be given as a stream of items. Suppose that $0 < \varepsilon \leq \frac{1}{3}$, that the rounded instance I_{R}, created from I, satisfies properties (P1)-(P4), and that the solution $\mathcal{S}$ of I_{R} uses at most $\mathsf{OPT}(I_{\mathrm{R}}) + \sigma$ bins. Let $\mathsf{ALG}(I)$ be the number of bins that our algorithm outputs. Then, it holds that $\mathsf{OPT}(I) \leq \mathsf{ALG}(I) \leq (1 + 3\varepsilon) \cdot \mathsf{OPT}(I) + \sigma + \mathcal{O}\left(\log \frac{1}{\varepsilon}\right)$.*

3.1 Processing the Stream and Rounding

The streaming algorithm of the rounding stage makes use of the deterministic quantile summary of Greenwald and Khanna [26]. Given a precision $\delta > 0$ and an input stream of numbers $s_1, \ldots, s_N$, their algorithm computes a data structure $Q(\delta)$ which is able to answer a quantile query with precision δN. Namely, for any $0 \leq \phi \leq 1$, it returns an element s of the input stream such that the rank of s is $[(\phi - \delta)N, (\phi + \delta)N]$, where the rank of s is the position of s in the non-increasing ordering of the input stream.[4] The data structure stores an ordered sequence of tuples, each consisting of an input number s_i and valid lower and upper bounds on the true rank of s_i in the input sequence.[5] The first and last stored items correspond to the maximum and minimum numbers in the stream, respectively. Note that the lower and upper bounds on the rank of any stored number differ by at most $\lfloor 2\delta N \rfloor$ and upper (or lower) bounds on the rank of two consecutive stored numbers differ by at most $\lfloor 2\delta N \rfloor$ as well. The space requirement of $Q(\delta)$ is $\mathcal{O}(\frac{1}{\delta} \cdot \log \delta N)$, however, in

[4]Note that if s appears more times in the stream, its rank is an interval rather than a single number. Also, unlike in [26], we order numbers non-increasingly, which is more convenient for BIN PACKING.

[5]More precisely, valid lower and upper bounds on the rank of s_i can be computed easily from the set of tuples.

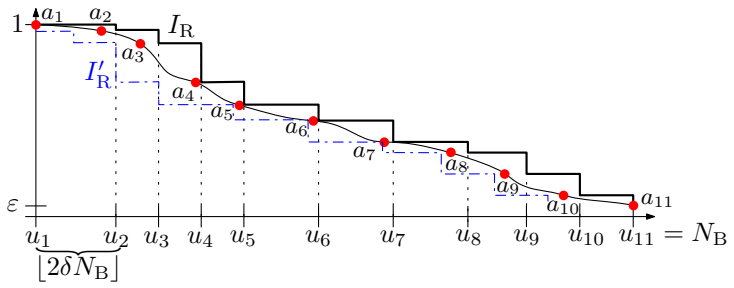

Figure 1: An illustration of the original distribution of sizes of big items in I_{B}, depicted by a smooth curve, and the distribution of item sizes in the rounded instance I_{R}, depicted by a bold "staircase" function. The distribution of I'_{R} (which is I_{R} without the $\lfloor 4\delta N_{\mathrm{B}} \rfloor$ biggest items) is depicted a (blue) dash dotted line. Selected items $a_i, \ldots, a_q$, with $q = 11$, are illustrated by (red) dots, and the upper bounds $u_1, \ldots, u_q$ on the ranks appear on the x axis.

practice the space used is observed to scale linearly with $\frac{1}{\delta}$ [36]. (Note that an offline optimal data structure for δ-approximate quantiles uses space $\mathcal{O}\left(\frac{1}{\delta}\right)$.) We use data structure $Q(\delta)$ to construct our algorithm for processing the stream I_{B} of big items.

Simple rounding algorithm. We begin by describing a simpler solution with $\delta = \frac{1}{4}\varepsilon^2$, resulting in a rounded instance with $\widetilde{\mathcal{O}}(\frac{1}{\varepsilon^2})$ item sizes. Subsequently, we introduce a more involved solution with smaller space cost. The algorithm uses a quantile summary structure to determine the rounding scheme. Given a (big) item s_i from the input, we insert it into $Q(\delta)$. After processing all items, we extract from $Q(\delta)$ the set of stored input items (i.e., their sizes) together with upper bounds on their rank (where the largest size has highest rank 1, and the smallest size has least rank N). Note that the number N_{B} of big items in I_{B} is less than $\frac{1}{\varepsilon}\mathsf{SIZE}(I_{\mathrm{B}}) \leq \frac{1}{\varepsilon}\mathsf{OPT}(I_{\mathrm{B}})$ as each is of size more than ε. Let q be the number of items (or tuples) extracted from $Q(\delta)$; we get that $q = \mathcal{O}(\frac{1}{\delta} \cdot \log \delta N_{\mathrm{B}}) = \mathcal{O}(\frac{1}{\varepsilon^2} \cdot \log(\varepsilon \cdot \mathsf{OPT}(I_{\mathrm{B}})))$. Let $(a_1, u_1 = 1), (a_2, u_2), \ldots, (a_q, u_q = N_{\mathrm{B}})$ be the output pairs of an item size and the bound on its rank, sorted so that $a_1 \geq a_2 \geq \cdots \geq a_q$. We define the rounded instance I_{R} with at most q item sizes as follows: I_{R} contains $(u_{j+1} - u_j)$ items of size a_j for each $j = 1, \ldots, q-1$, plus one item of size a_q. (See Figure 1.)

We show that the desired properties (P1)-(P4) hold with $\sigma = q$. Property (P1) follows easily from the definition of I_{R} and the design of data structure $Q(\delta)$. Note that the number of items is preserved. To show (P2), suppose for a contradiction that the i-th biggest item in I_{B} is bigger than the i-th biggest item in I_{R}, whose size is a_j for $j = 1, \ldots, q-1$, i.e., $i \in [u_j, u_{j+1})$ (note that $j < q$ as a_q is the smallest item in I_{B} and is present only once in I_{R}). We get that the rank of item a_j in I_{B} is strictly more than i, and as $i \geq u_j$, we get a contradiction with the fact that u_j is a valid upper bound on the rank of a_j in I_{B}.

Next, we give bounds for $\mathsf{OPT}(I_{\mathrm{R}})$ and $\mathsf{SIZE}(I_{\mathrm{R}})$, which are required by properties (P3) and (P4). We pack the $\lfloor 4\delta N_{\mathrm{B}} \rfloor$ biggest items in I_{R} separately into "extra" bins. Using the choice of $\delta = \frac{1}{4}\varepsilon^2$ and $N_{\mathrm{B}} \leq \frac{1}{\varepsilon}\mathsf{SIZE}(I_{\mathrm{B}})$, we bound the number of these items and thus extra bins by $4\delta N_{\mathrm{B}} \leq \varepsilon \cdot \mathsf{SIZE}(I_{\mathrm{B}}) \leq \varepsilon \cdot \mathsf{OPT}(I_{\mathrm{B}})$. Let I'_{R} be the remaining items in I_{R}. We claim that that the i-th biggest item b_i in I_{B} is bigger than the i-th biggest item in I'_{R} with size equal to a_j for $j = 1, \ldots, q$. For a contradiction, suppose that $b_i < a_j$, which implies that the rank r_j of a_j in I_{B} is less than i. Note that $j < q$ as a_q is the smallest item in I_{B}. Since we packed the $\lfloor 4\delta N_{\mathrm{B}} \rfloor$ biggest items from I_{R} separately, one of the positions of a_j in the ordering of I_{R} is $i + \lfloor 4\delta N_{\mathrm{B}} \rfloor$ and so we have $i + \lfloor 4\delta N_{\mathrm{B}} \rfloor < u_{j+1} \leq u_j + \lfloor 2\delta N_{\mathrm{B}} \rfloor$, where the first inequality holds by the construction of I_{R} and the second inequality is by the design of data structure $Q(\delta)$. It follows that $i < u_j - \lfloor 2\delta N_{\mathrm{B}} \rfloor$. Combining this with $r_j < i$, we obtain that the rank of a_j in I_{B} is less than $u_j - \lfloor 2\delta N_{\mathrm{B}} \rfloor$, which contradicts that $u_j - \lfloor 2\delta N_{\mathrm{B}} \rfloor$ is a valid lower bound on the rank of a_j.

The claim implies $\mathsf{OPT}(I'_{\mathrm{R}}) \leq \mathsf{OPT}(I_{\mathrm{B}})$ and $\mathsf{SIZE}(I'_{\mathrm{R}}) \leq \mathsf{SIZE}(I_{\mathrm{B}})$. We thus get that $\mathsf{OPT}(I_{\mathrm{R}}) \leq$

$\mathsf{OPT}(I'_{\mathrm{R}}) + \lfloor 4\delta N_{\mathrm{B}} \rfloor \leq \mathsf{OPT}(I_{\mathrm{B}}) + \varepsilon \cdot \mathsf{OPT}(I_{\mathrm{B}})$, proving property (P3). Similarly, $\mathsf{SIZE}(I_{\mathrm{R}}) \leq \mathsf{SIZE}(I'_{\mathrm{R}}) + \lfloor 4\delta N_{\mathrm{B}} \rfloor \leq \mathsf{SIZE}(I_{\mathrm{B}}) + \varepsilon \cdot \mathsf{SIZE}(I_{\mathrm{B}})$, showing (P4).

Better rounding algorithm. Our improved rounding algorithm reduces the number of sizes in the rounded instance (and also the memory requirement) from $\widetilde{\mathcal{O}}(\frac{1}{\varepsilon^2})$ to $\widetilde{\mathcal{O}}(\frac{1}{\varepsilon})$. It is based on the observation that the number of items of sizes close to ε can be approximated with much lower accuracy than the number of items with sizes close to 1, without affecting the quality of the overall approximation. This was observed already by Karmarkar and Karp [33].

The rounding and its analysis is fully described in Appendix A.1 which also gives the proof of Lemma 1. Here, we give a brief overview. Big items are split into groups based on size such that for an integer $j \geq 1$, the j-th group contains items with sizes in $(2^{-j-1}, 2^{-j}]$. Thus, there are $\lceil \log_2 \frac{1}{\varepsilon} \rceil$ groups. For each group j, we use a separate data structure $Q_j := Q(\delta)$ with $\delta = \frac{1}{8}\varepsilon$.

After all items arrive, we extract stored items from each data structure Q_j and create the rounded instance for each group as in the previous section. Then, the input summary is just the union of the rounded instances over all groups. We show that properties (P1)-(P4) hold for the input summary in a similar way as for the simple rounding algorithm, also using the following observation: Let N_j be the number of big items in group j. Then $\mathsf{SIZE}(I_{\mathrm{B}}) > \sum_j N_j \cdot 2^{-j-1}$. This holds as any item in group j has size exceeding 2^{-j-1}.

3.2 Bin Packing and Quantile Summaries

In the previous section, the deterministic quantile summary data structure from [26] allows us to obtain a streaming approximation scheme for BIN PACKING. We argue that this connection runs deeper.

We start with different scenarios for which there exist better quantile summaries. First, if all big item sizes belong to a universe $U \subset (\varepsilon, 1]$, then it can be better to use the quantile summary of Shrivastava *et al.* [39], which provides a guarantee of $\mathcal{O}(\frac{1}{\delta} \cdot \log |U|)$ on the space complexity, where δ is the precision requirement. Thus, by using k copies of this quantile summary in a similar way as in Section 3.1, we get a streaming $1+\varepsilon$-approximation algorithm for BIN PACKING that runs in space $\mathcal{O}(\frac{1}{\varepsilon} \cdot \log \frac{1}{\varepsilon} \cdot \log |U|)$.

Second, if we allow the algorithm to use randomization and fail with probability γ, we can employ the optimal randomized quantile summary of Karnin, Lang, and Liberty [34], which, for a given precision δ and failure probability η, uses space $\mathcal{O}(\frac{1}{\delta} \cdot \log\log \frac{1}{\eta})$ and does not provide a δ-approximate quantile for some quantile query with probability at most η. In particular, using k copies of their data structure with precision $\delta = \Theta(\varepsilon)$ and failure probability $\eta = \gamma/k$, similarly as in Section 3.1, gives a streaming $1+\varepsilon$-approximation algorithm for BIN PACKING which fails with probability at most γ and runs in space $\mathcal{O}\left(\frac{1}{\varepsilon} \cdot \log \frac{1}{\varepsilon} \cdot \log\log(\log \frac{1}{\varepsilon}/\gamma)\right)$.

More intriguingly, the connection between quantile summaries and BIN PACKING also goes in the other direction. Namely, we show that a streaming $1+\varepsilon$-approximation algorithm for BIN PACKING with space bounded by $S(\varepsilon, \mathsf{OPT})$ (or $S(\varepsilon, N)$) implies a data structure of size $S(\varepsilon, N)$ for the following ESTIMATING RANK problem: Create a summary of a stream of N numbers which is able to provide a δ-approximate rank of any query q, i.e., the number of items in the stream which are larger than q, up to an additive error of $\pm\delta N$. A summary for ESTIMATING RANK is essentially a quantile summary and we can actually use it to find an approximate quantile by doing a binary search over possible item names. However, this approach does *not* guarantee that the item name returned will correspond to one of the items present in the stream.

The reduction from ESTIMATING RANK to BIN PACKING goes as follows: Suppose that all numbers in the input stream for ESTIMATING RANK are from interval $(\frac{1}{2}, \frac{2}{3})$ (this is without loss of generality by scaling) and let q be a query in $(\frac{1}{2}, \frac{2}{3})$. For each number a_i in the stream for ESTIMATING RANK, we introduce two items of size a_i in the stream for BIN PACKING. After these $2N$ items (two copies each of $a_1, \ldots, a_N$) are inserted in the same order as in the stream for ESTIMATING RANK, we then insert a further $2N$ items in the stream for BIN PACKING, all of size $1-q$. Observe first that no pair of the first $2N$ items can be placed in the

same bin, so we must open at least $2N$ bins, two for each of $a_1, \ldots, a_N$. Since $\frac{1}{2} > (1-q) > \frac{1}{3}$, and $a_i > \frac{1}{2}$, we can place at most one of the $2N$ items of size $(1-q)$ in a bin with a_i in it, provided that $a_i + (1-q) \leq 1$, i.e. $a_i \leq q$. Thus, we can pack a number of the $(1-q)$-sized items, equivalent to $2(N - \mathrm{rank}(q))$, in the first $2N$ bins. This leaves $2\,\mathrm{rank}(q)$ items, all of size $(1-q)$. We pack these optimally into $\mathrm{rank}(q)$ additional bins, for a total of $2N + \mathrm{rank}(q)$ bins.

We claim that a $1+\varepsilon$-approximation of the optimum number of bins provides a 4ε-approximate rank of q. Indeed, let m be the number of bins returned by the algorithm and let $r = m - 2N$ be the estimate of $\mathrm{rank}(q)$. We have that the optimal number of bins equals $2N + \mathrm{rank}(q)$ and thus $2N + \mathrm{rank}(q) \leq m \leq (1+\varepsilon) \cdot (2N + \mathrm{rank}(q)) + o(N)$. By using $r = m - 2N$ and rearranging, we get

$$\mathrm{rank}(q) \leq r \leq \mathrm{rank}(q) + \varepsilon\,\mathrm{rank}(q) + 2\varepsilon N + o(N)\,.$$

Since the right-hand side can be upper bounded by $\mathrm{rank}(q) + 4\varepsilon N$ (provided that $o(N) < \varepsilon N$), r is a 4ε-approximate rank of q. Hence, the memory state of an algorithm for BIN PACKING after processing the first $2N$ items (of sizes $a_1, \ldots, a_N$) can be used as a data structure for ESTIMATING RANK.

In [15] we show a space lower bound of $\Omega(\frac{1}{\varepsilon} \cdot \log \varepsilon N)$ for comparison-based data structures for ESTIMATING RANK (and for quantile summaries as well).

Theorem 3 (Theorem 13 in [15])**.** *For any $0 < \varepsilon < \frac{1}{16}$, there is no deterministic comparison-based data structure for* ESTIMATING RANK *which stores $o\left(\frac{1}{\varepsilon} \cdot \log \varepsilon N\right)$ items on any input stream of length N.*

We conclude that there is no comparison-based streaming algorithm for BIN PACKING which stores $o(\frac{1}{\varepsilon} \cdot \log \mathsf{OPT})$ items on any input stream (recall that $N = \mathcal{O}(\mathsf{OPT})$ in our reduction). Note that our algorithm is comparison-based if we employ the comparison-based quantile summary of Greenwald and Khanna [26], except that it needs to determine the size group for each item, which can be done by comparisons with 2^{-j} for integer values of j. Nevertheless, comparisons with a fixed set of constants does not affect the reduction from ESTIMATING RANK (i.e., the reduction can choose an interval to avoid all constants fixed in the algorithm), thus the lower bound of $\Omega\left(\frac{1}{\varepsilon} \cdot \log \mathsf{OPT}\right)$ applies to our algorithm as well. This yields near optimality of our approach, up to a factor of $\mathcal{O}\left(\log \frac{1}{\varepsilon}\right)$. Finally, we remark that the lower bound of $\Omega(\frac{1}{\varepsilon} \cdot \log\log \frac{1}{\delta})$ for randomized comparison-based quantile summaries [34] translates to BIN PACKING as well.

4 Vector Scheduling

We provide a novel approach for creating an input summary for VECTOR SCHEDULING, based on combining small items into containers. Our streaming algorithm stores all big jobs and all containers, created from small items, that are relatively big as well. Thus, there is a bounded number of big jobs and containers, and the space used is bounded as well. We show that this simple summarization preserves the optimal makespan up to a factor of $2 - \frac{1}{m} + \varepsilon$ for any $0 < \varepsilon \leq 1$. Take $m \geq 2$, since for $m = 1$ there is a trivial streaming algorithm that just sums up the vectors of all jobs to get the optimal makespan. We assume that the algorithm knows (an upper bound on) m in advance.

Algorithm description. For $0 < \varepsilon \leq 1$ and $m \geq 2$, the algorithms works as follows: For each $k = 1, \ldots, d$, it keeps track of the total load of all jobs in dimension k, denoted L_k. Note that the optimal makespan satisfies $\mathsf{OPT} \geq \max_k \frac{1}{m} \cdot L_k$. Assume for simplicity that when a new job arrives, $\max_k \frac{1}{m} \cdot L_k = 1$; if not, we rescale every quantity by this maximum. Hence, the optimum makespan for jobs that arrived so far is at least one, while $L_k \leq m$ for any $k = 1, \ldots, d$ (an alternative lower bound on OPT is the maximum ℓ_∞ norm of a job seen so far, but our algorithm does not use this).

Let $\gamma = \Theta\left(\varepsilon^2 / \log \frac{d^2}{\varepsilon}\right)$; the constant hidden in Θ follows from the analysis below. We also ensure that $\gamma \leq \frac{1}{4}\varepsilon$. We say that a job with vector $\mathbf{v}$ is *big* if $\|\mathbf{v}\|_\infty > \gamma$; otherwise it is *small*. The algorithm stores all big jobs (i.e., the full vector of each big job), while it aggregates small jobs into containers, and does not

store any small job directly. A *container* is simply a vector $\mathbf{c}$ that equals the sum of vectors for small jobs assigned to this container, and we ensure that $\|\mathbf{c}\|_\infty \leq 2\gamma$. Furthermore, container $\mathbf{c}$ is *closed* if $\|\mathbf{c}\|_\infty > \gamma$, otherwise, it is *open*. As two open containers can be combined into one (open or closed) container, we maintain only one open container. We execute a variant of the NEXT FIT algorithm to pack the containers, adding the incoming small job into the open container, where it always fits as any small vector $\mathbf{v}$ satisfies $\|\mathbf{v}\|_\infty \leq \gamma$. All containers are retained in the memory.

When a new small job arrives or when a big job becomes small, we assign it in the open container. If this container becomes closed, we open a new, empty one. Moreover, it may happen that a previously closed container becomes open again. In this case, we combine open containers as long as we have at least two of them. This completes the description of the algorithm. (We remark that for packing the containers, we may also use another, more efficient algorithm, such as FIRST FIT, which however makes no difference in the approximation guarantee.)

Properties of the input summary. After all jobs are processed, we assume again that $\max_k \frac{1}{m} \cdot L_k = 1$, which implies that $\mathsf{OPT} \geq 1$. Since any big job and any closed container, each characterized by a vector $\mathbf{v}$, satisfy $\|\mathbf{v}\|_\infty > \gamma$, it holds that there are at most $\frac{1}{\gamma} \cdot d \cdot m$ big jobs and closed containers. As at most one container remains open in the end and any job or container is described by d numbers, the space cost is $\mathcal{O}\left(\frac{1}{\gamma} \cdot d^2 \cdot m\right) = \mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot d^2 \cdot m \cdot \log \frac{d}{\varepsilon}\right)$.

We now analyze the maximum approximation factor that can be lost by this summarization. Let I_{R} be the resulting instance formed by big jobs and containers with small items (i.e., the input summary), and let I be the original instance, consisting of jobs in the input stream. We show that $\mathsf{OPT}(I_{\mathrm{R}})$ and $\mathsf{OPT}(I)$ are close together, up to a factor of $2 - \frac{1}{m} + \varepsilon$, and an example in Appendix C shows that this bound is tight for our approach. Note, however, that we still need to execute an offline algorithm to get (an approximation of) $\mathsf{OPT}(I_{\mathrm{R}})$, which is not an explicit part of the summary.

The crucial part of the proof is to show that containers for small items can be assigned to machines so that the loads of all machines are nearly balanced in every dimension, especially in the case when containers constitute a large fraction of the total load of all jobs. Let L_k^{C} be the total load of containers in dimension k (equal to the total load of small jobs). Let $I_{\mathrm{C}} \subseteq I_{\mathrm{R}}$ be the instance consisting of all containers in I_{R}.

Lemma 4. *Supposing that* $\max_k \frac{1}{m} \cdot L_k = 1$*, the following holds:*

(i) There is a solution for instance I_{C} *with load at most* $\max(\frac{1}{2}, \frac{1}{m} \cdot L_k^{\mathrm{C}}) + 2\varepsilon + 4\gamma$ *in each dimension* k *on every machine.*

(ii) $\mathsf{OPT}(I) \leq \mathsf{OPT}(I_{\mathrm{R}}) \leq \left(2 - \frac{1}{m} + 3\varepsilon\right) \cdot \mathsf{OPT}(I)$.

Proof. (i) We obtain the solution from the randomized online algorithm by Im *et al.* [30]. Although this algorithm has ratio $\mathcal{O}(\log d / \log\log d)$ on general instances, we show that it behaves substantially better when jobs are small enough. In a nutshell, this algorithm works by first assigning each job j to a uniformly random machine i and if the load of machine i exceeds a certain threshold, then the job is reassigned by GREEDY. The online GREEDY algorithm works by assigning jobs one by one, each to a machine so that the makespan increases as little as possible (breaking ties arbitrarily).

Let $L'_k = \max(\frac{1}{2}, \frac{1}{m} \cdot L_k^{\mathrm{C}})$. We assume that each machine has its capacity of $L'_k + 2\varepsilon + 4\gamma$ in each dimension k split into two parts: The first part has capacity $L'_k + \varepsilon + 2\gamma$ in dimension k for the containers assigned randomly, and the second part has capacity $\varepsilon + 2\gamma$ in all dimensions for the containers assigned by GREEDY. Note that GREEDY cares about the load in the second part only.

The algorithm assigns containers one by one as follows: For each container $\mathbf{c}$, it first chooses a machine i uniformly and independently at random. If the load of the first part of machine i already exceeds $L'_k + \varepsilon$ in some dimension k, then $\mathbf{c}$ is passed to GREEDY, which assigns it according to the loads in the second part. Otherwise, the algorithm assigns $\mathbf{c}$ to machine i.

As each container $\mathbf{c}$ satisfies $\|\mathbf{c}\|_\infty \leq 2\gamma$, it holds that randomly assigned containers fit into capacity $L'_k + \varepsilon + 2\gamma$ in any dimension k on any machine. We show that the expected amount of containers assigned by GREEDY is small enough so that they fit into machines with capacity of $\varepsilon + 2\gamma$, which in turn implies that there is a choice of random bits for the assignment so that the capacity for GREEDY is not exceeded. The existence of a solution with capacity $L'_k + 2\varepsilon + 4\gamma$ in each dimension k will follow.

Consider a container $\mathbf{c}$ and let i be the machine chosen randomly for $\mathbf{c}$. We claim that for any dimension k, the load on machine i in dimension k, assigned before processing $\mathbf{c}$, exceeds $L'_k + \varepsilon$ with probability of at most $\frac{\varepsilon}{d^2}$. To show the claim, we use the following Chernoff-Hoeffding bound:

Fact 5. *Let $X_1, \ldots, X_n$ be independent binary random variables and let $a_1, \ldots, a_n$ be coefficients in $[0, 1]$. Let $X = \sum_i a_i X_i$. Then, for $0 < \delta \leq 1$ and $\mu \geq \mathbb{E}[X]$, it holds that $\Pr[X > (1+\delta)\cdot\mu] \leq \exp\left(-\frac{1}{3}\cdot\delta^2\cdot\mu\right)$.*

We use this bound with variable $X_{\mathbf{c}'}$ for each vector $\mathbf{c}'$ assigned randomly before vector $\mathbf{c}$ and not reassigned by GREEDY. We have $X_{\mathbf{c}'} = 1$ if $\mathbf{c}'$ is assigned on machine i. Let $a_{\mathbf{c}'} = \frac{1}{2\gamma}\cdot\mathbf{c}'_k \leq 1$. Let $X = \sum_{\mathbf{c}'} a_{\mathbf{c}'} X_{\mathbf{c}'}$ be the random variable equal to the load on machine i in dimension k, scaled by $\frac{1}{2\gamma}$. It holds that $\mathbb{E}[X] \leq \frac{1}{m}\cdot\frac{1}{2\gamma}\cdot L'_k\cdot m = \frac{1}{2\gamma}\cdot L'_k$, since each container $\mathbf{c}'$ is assigned to machine i with probability $\frac{1}{m}$ and $L'_k\cdot m$ is the upper bound on the total load of containers in dimension k. Using the Chernoff-Hoeffding bound with $\mu = \frac{1}{2\gamma}\cdot L'_k$ and $\delta = \varepsilon \leq 1$, we get that $\Pr[X > (1+\varepsilon)\cdot\frac{1}{2\gamma}\cdot L'_k] \leq \exp\left(-\frac{1}{3}\cdot\varepsilon^2\cdot\frac{1}{2\gamma}\cdot L'_k\right)$. Using $\gamma = \mathcal{O}\left(\varepsilon^2/\log\frac{d^2}{\varepsilon}\right)$ and $L'_k \geq \frac{1}{2}$, we obtain $\exp\left(-\frac{1}{3}\cdot\varepsilon^2\cdot\frac{1}{2\gamma}\cdot L'_k\right) \leq \exp\left(-\Omega\left(\log\frac{d^2}{\varepsilon}\right)\right) \leq \frac{\varepsilon}{d^2}$, where the last inequality holds for a suitable choice of the multiplicative constant in the definition of γ. This is sufficient to show the claim as $X > (1+\varepsilon)\cdot\frac{1}{2\gamma}\cdot L'_k$ if and only if the load on machine i in dimension k, assigned randomly before $\mathbf{c}$, exceeds $(1+\varepsilon)\cdot L'_k$.

By the union bound, the claim implies that each container $\mathbf{c}$ is reassigned by GREEDY with probability at most $\frac{\varepsilon}{d}$. Let G be the random variable equal to the sum of the ℓ_1 norms (where $\|\mathbf{c}\|_1 = \sum_{k=1}^{d}\mathbf{c}_k$) of containers assigned by GREEDY. Using the linearity of expectation and the claim, we have

$$\mathbb{E}[G] \leq \textstyle\sum_{\mathbf{c}}\frac{\varepsilon}{d}\cdot\|\mathbf{c}\|_1 \leq \frac{\varepsilon}{d}\cdot m\cdot d = \varepsilon\cdot m,$$

where the second inequality uses that the total load of containers in each dimension is at most m. Let $\mu_{\mathbf{G}}$ be the makespan of the containers created by GREEDY. Observe that each machine has a dimension with load at least $\mu_{\mathbf{G}} - 2\gamma$. Indeed, otherwise, if there is a machine i with load less than $\mu_{\mathbf{G}} - 2\gamma$ in all coordinates, the last container $\mathbf{c}$ assigned by GREEDY that caused the increase of the makespan to $\mu_{\mathbf{G}}$ would be assigned to machine i, and the makespan after assigning $\mathbf{c}$ would be smaller than $\mu_{\mathbf{G}}$ (using $\|\mathbf{c}\|_\infty \leq 2\gamma$). It follows that $\mu_{\mathbf{G}} - 2\gamma \leq \frac{1}{m}\cdot G$ and, using $\mathbb{E}[G] \leq \varepsilon\cdot m$, we get that $\mathbb{E}[\mu_{\mathbf{G}}] - 2\gamma \leq \varepsilon$. Thus (i) holds.

(ii) The first inequality is straightforward as any solution for I_{R} can be used as a solution for I, just packing small items first in containers and then the containers according to the solution for I_{R}.

We create a solution of I_{R} of makespan at most $\left(2 - \frac{1}{m} + 3\varepsilon\right)\cdot\mathsf{OPT}(I)$ as follows: We take an optimal solution $\mathcal{S}_{\mathrm{B}}$ for instance $I_{\mathrm{R}} \setminus I_{\mathrm{C}}$, i.e., for big jobs only, and combine it, in an arbitrary way, with solution $\mathcal{S}_{\mathrm{C}}$ for containers from (i), to obtain a solution $\mathcal{S}$ for I_{R}. Let μ_k be the largest load assigned to a machine in dimension k in solution $\mathcal{S}_{\mathrm{B}}$; we have $\mu_k \leq \mathsf{OPT}(I)$. Note that $L_k^{\mathrm{C}} \leq m - \mu_k$, since the total load of big jobs and containers together is at most m. Consider the load on machine i in dimension k in solution $\mathcal{S}$. If $\frac{1}{m}\cdot L_k^{\mathrm{C}} \geq \frac{1}{2}$, then this load is bounded by $\mu_k + \frac{1}{m}\cdot L_k^{\mathrm{C}} + 2\varepsilon + 4\gamma \leq \mu_k + \frac{1}{m}\cdot(m - \mu_k) + 3\varepsilon = \left(1 - \frac{1}{m}\right)\cdot\mu_k + 1 + 3\varepsilon \leq \left(2 - \frac{1}{m} + 3\varepsilon\right)\cdot\mathsf{OPT}(I)$, where the first inequality uses $L_k^{\mathrm{C}} \leq m - \mu_k$ and $\gamma \leq \frac{1}{4}\varepsilon$ (ensured by the definition of γ), and the last inequality holds by $\mu_k \leq \mathsf{OPT}(I)$ and $1 \leq \mathsf{OPT}(I)$. Otherwise, $\frac{1}{m}\cdot L_k^{\mathrm{C}} < \frac{1}{2}$, in which case the load on machine i in dimension k is at most $\mu_k + \frac{1}{2} + 2\varepsilon + 4\gamma \leq (1.5+3\varepsilon)\cdot\mathsf{OPT}(I) \leq (2 - \frac{1}{m} + 3\varepsilon)\cdot\mathsf{OPT}(I)$, using similar arguments as in the previous case and $m \geq 2$. □

Acknowledgments. The work is supported by European Research Council grant ERC-2014-CoG 647557. The authors wish to thank Michael Shekelyan for fruitful discussions.

References

[1] S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999.

[2] David Applegate, Luciana S Buriol, Bernard L Dillard, David S Johnson, and Peter W Shor. The cutting-stock approach to bin packing: Theory and experiments. In *ALENEX*, volume 3, pages 1–15, 2003.

[3] Yossi Azar, Ilan Reuven Cohen, Seny Kamara, and Bruce Shepherd. Tight bounds for online vector bin packing. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, STOC '13, pages 961–970. ACM, 2013.

[4] Yossi Azar, Ilan Reuven Cohen, and Debmalya Panigrahi. Randomized algorithms for online vector load balancing. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 980–991. SIAM, 2018.

[5] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new and improved algorithm for online bin packing. In *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *LIPIcs*, pages 5:1–5:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

[6] János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. *Theoretical Computer Science*, 440-441:1 – 13, 2012.

[7] Nikhil Bansal, Marek Eliáš, and Arindam Khan. Improved approximation for vector bin packing. In *Proceedings of the 27th annual ACM-SIAM symposium on Discrete algorithms*, SODA '16, pages 1561–1579. SIAM, 2016.

[8] Nikhil Bansal, Tim Oosterwijk, Tjark Vredeveld, and Ruben van der Zwaan. Approximating vector scheduling: Almost matching upper and lower bounds. *Algorithmica*, 76(4):1077–1096, Dec 2016.

[9] Tugkan Batu, Petra Berenbrink, and Christian Sohler. A sublinear-time approximation scheme for bin packing. *Theoretical Computer Science*, 410(47-49):5082–5092, 2009.

[10] Richard Beigel and Bin Fu. A dense hierarchy of sublinear time approximation schemes for bin packing. In *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, pages 172–181. Springer, 2012.

[11] Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM journal on computing*, 33(4):837–851, 2004.

[12] Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of approximation schemes for the classical scheduling problem. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 657–668. SIAM, 2014.

[13] Henrik I Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

[14] 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 New York, 2013.

[15] Graham Cormode and Pavel Veselý. Tight Lower Bound for Comparison-Based Quantile Summaries. *arXiv e-prints*, page arXiv:1905.03838, May 2019.

[16] György Dósa and Jiří Sgall. First Fit bin packing: A tight analysis. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *LIPIcs*, pages 538–549. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.

[17] Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-dynamic bin packing with little repacking. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *LIPIcs*, pages 51:1–51:24. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

[18] W. Fernandez de la Vega and G.S. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[19] Rudolf Fleischer and Michaela Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000.

[20] 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.

[21] Michael R Garey and David S Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. Freeman, 1979.

[22] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[23] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting stock problem—part ii. *Operations research*, 11(6):863–888, 1963.

[24] Michel X. Goemans and Thomas Rothvoß. Polynomiality for bin packing with a constant number of item types. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 830–839. SIAM, 2014.

[25] Todd Gormley, Nicholas Reingold, Eric Torng, and Jeffery Westbrook. Generating adversaries for request-answer games. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 564–565. SIAM, 2000.

[26] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 58–66, November 2001.

[27] D. G. Harris and A. Srinivasan. The Moser-Tardos framework with partial resampling. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, FOCS '13, pages 469–478, Oct 2013.

[28] Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 2616–2625. SIAM, 2017.

[29] J.F. Rudin III. *Improved bounds for the on-line scheduling problem*. PhD thesis, The University of Texas at Dallas, 2001.

[30] S. Im, N. Kell, J. Kulkarni, and D. Panigrahi. Tight bounds for online vector scheduling. *SIAM Journal on Computing*, 48(1):93–121, 2019.

[31] Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 72:1–72:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

[32] David S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.

[33] Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 312–320, Nov 1982.

[34] Z. Karnin, K. Lang, and E. Liberty. Optimal quantile approximation in streams. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 71–78, Oct 2016.

[35] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32:562–572, July 1985.

[36] Ge Luo, Lu Wang, Ke Yi, and Graham Cormode. Quantiles over data streams: Experimental comparisons, new analyses, and further improvements. *The VLDB Journal*, 25(4):449–472, August 2016.

[37] Andrew McGregor. Graph stream algorithms: A survey. *SIGMOD Rec.*, 43(1):9–20, May 2014.

[38] Shanmugavelayutham Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

[39] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: New aggregation techniques for sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 239–249. ACM, 2004.

[40] Gerhard J. Woeginger. There is no asymptotic PTAS for two-dimensional vector packing. *Information Processing Letters*, 64(6):293 – 297, 1997.

A Omitted Proofs from Section 3

A.1 Proof of Lemma 1

Lemma 1. *Given a steam I_{B} of big items, there is a deterministic streaming algorithm that outputs a* HIGH-MULTIPLICITY BIN PACKING *instance satisfying (P1)-(P4) with $\sigma = \mathcal{O}\left(\frac{1}{\varepsilon} \cdot \log \frac{1}{\varepsilon} \cdot \log \mathsf{OPT}(I_{\mathrm{B}})\right)$.*

Proof. Let $k = \lceil \log_2 \frac{1}{\varepsilon} \rceil$. We first group big items in k groups $0, \ldots, k-1$ by size such that in group j there are items with sizes in $(2^{-j-1}, 2^{-j}]$. That is, the size intervals for groups are $(0.5, 1], (0.25, 0.5]$, etc. Let $N_j, j = 0, \ldots, k-1$, be the number of big items in group j; clearly, $N_j < 2^{j+1}\mathsf{SIZE}(I_{\mathrm{B}}) \leq 2^{j+1}\mathsf{OPT}(I_{\mathrm{B}})$. Note that the total size of items in group j is in $(2^{-j-1} \cdot N_j, 2^{-j} \cdot N_j]$. Summing over all groups, we get in particular that

$$\mathsf{SIZE}(I_{\mathrm{B}}) > \sum_{j=0}^{k} \frac{N_j}{2^{j+1}}\,. \tag{1}$$

For each group j, we use a separate data structure $Q_j := Q(\delta)$ with $\delta = \frac{1}{8}\varepsilon$, where $Q(\delta)$ is the quantile summary from [26] with precision δ. So when a big item of size s_i arrives, we find j such that $s_i \in (2^{-j-1}, 2^{-j}]$ and insert s_i into Q_j. After processing all items, for each group j, we do the following: We extract from Q_j the set of stored input items (i.e., their sizes) together with upper bounds on their rank. Let $(a_1^j, u_1^j = 1), (a_2^j, u_2^j), \ldots, (a_{q_j}^j, u_{q_j}^j = N_j)$ be the pairs of an item size and the upper bound on its rank in group j, ordered as in the simpler algorithm so that $a_1^j \geq a_2^j \geq \cdots \geq a_{q_j}^j$. We have

$$q_j = \mathcal{O}\left(\frac{1}{\delta} \cdot \log \delta N_j\right) = \mathcal{O}\left(\frac{1}{\varepsilon} \cdot \log\left(\varepsilon \cdot 2^{j+1}\mathsf{OPT}(I_{\mathrm{B}})\right)\right) = \mathcal{O}\left(\frac{1}{\varepsilon} \cdot \log \mathsf{OPT}(I_{\mathrm{B}})\right),$$

since $\varepsilon 2^j \leq \varepsilon 2^k \leq 1$.

An auxiliary instance I_{R}^j is formed by $(u_{i+1}^j - u_i^j)$ items of size a_i for $i = 1, \ldots, q_j - 1$ plus one item of size a_{q_j}. To create the rounded instance I_{R}, we take the union of all auxiliary instances I_{R}^j, $j = 0, \ldots, k-1$. Note that the number of item sizes in I_{R} is

$$\sigma \leq \sum_{j=0}^{k-1} q_j = \sum_{j=0}^{k-1} \mathcal{O}\left(\frac{1}{\varepsilon} \cdot \log \mathsf{OPT}(I_{\mathrm{B}})\right) = \mathcal{O}\left(\frac{k}{\varepsilon} \cdot \log \mathsf{OPT}(I_{\mathrm{B}})\right) = \mathcal{O}\left(\frac{1}{\varepsilon} \cdot \log \frac{1}{\varepsilon} \cdot \log \mathsf{OPT}(I_{\mathrm{B}})\right).$$

We show that the desired properties (P1)-(P4) are satisfied. Property (P1) follows easily from the definition of I_{R} as the union of instances I_{R}^j and the design of data structures Q_j. To see property (P2), for every group j, it holds that the i-th biggest item in group j in I_{R} is at least as large as the i-th biggest item in group j in I_{B}. Indeed, for any $p = 0, \ldots, q_j$, u_p^j is a valid upper bound on the rank of a_p^j in group j in I_{B} and ranks of items of size a_p^j in group j in I_{R} are at least u_p^j. Moreover, the number of items is preserved in every group. Hence, overall, the i-th biggest item in I_{R} cannot be smaller than the i-th biggest item in I_{B}.

Next, we prove properties (P3) and (P4), i.e., the bounds on $\mathsf{OPT}(I_{\mathrm{R}})$ and on $\mathsf{SIZE}(I_{\mathrm{R}})$. For each group j, we pack the $\lfloor 4\delta N_j \rfloor$ biggest items in I_{R} with size in group j into "extra" bins, each containing 2^j items, except for at most one extra bin which may contain fewer than 2^j items. This is possible as any item in group j has size at most 2^{-j}. Using the choice of $\delta = \frac{1}{8}\varepsilon$ and (1), we bound the total number of extra bins by

$$\sum_{j=0}^{k} \left\lceil \frac{4\delta N_j}{2^j} \right\rceil \leq 4 \cdot \frac{1}{8}\varepsilon \cdot \sum_{j=0}^{k} \frac{N_j}{2^j} + k \leq \frac{1}{2}\varepsilon \cdot 2 \cdot \mathsf{SIZE}(I_{\mathrm{B}}) + k \leq \varepsilon \cdot \mathsf{OPT}(I_{\mathrm{B}}) + k\,. \tag{2}$$

Let I'_{R} be the remaining items in I_{R}. Consider group j and let $I_{\mathrm{B}}(j)$ and $I'_{\mathrm{R}}(j)$ be the items with sizes in $(2^{-j-1}, 2^{-j}]$ in I_{B} and in I'_{R}, respectively. We claim that the i-th biggest item b_i in $I_{\mathrm{B}}(j)$ is at least

as large as the i-th biggest item in $I'_{\mathrm{R}}(j)$ with size equal to a_p for $p = 1, \ldots, q_j$. For a contradiction, suppose that $b_i < a_p$, which implies that the rank r_p of a_p in $I_{\mathrm{B}}(j)$ is less than i. Note that $p < q_j$ as a_{q_j} is the smallest item in $I_{\mathrm{B}}(j)$. Since we packed the largest $\lfloor 4\delta N_j \rfloor$ items from $I_{\mathrm{R}}(j)$ separately, we have $i + \lfloor 4\delta N_j \rfloor < u_{p+1} \leq u_p + \lfloor 2\delta N_j \rfloor$, where the last inequality is by the design of data structure Q_j. It follows that $i < u_p - \lfloor 2\delta N_j \rfloor$. Combining it with $r_p < i$, we obtain that the rank of a_p in $I_{\mathrm{B}}(j)$ is less than $u_p - \lfloor 2\delta N_j \rfloor$, which contradicts that $u_p - \lfloor 2\delta N_j \rfloor$ is a valid lower bound on the rank of a_p. Hence, the claim holds for any group and it immediately implies $\mathsf{OPT}(I'_{\mathrm{R}}) \leq \mathsf{OPT}(I_{\mathrm{B}})$ and $\mathsf{SIZE}(I'_{\mathrm{R}}) \leq \mathsf{SIZE}(I_{\mathrm{B}})$.

Combining with (2), we get that $\mathsf{OPT}(I_{\mathrm{R}}) \leq \mathsf{OPT}(I'_{\mathrm{R}}) + \varepsilon \cdot \mathsf{OPT}(I_{\mathrm{B}}) + k \leq (1+\varepsilon) \cdot \mathsf{OPT}(I_{\mathrm{B}}) + k$, thus (P3) holds. Similarly, to bound the total wasted space, observe that the total size of items of I_{R} that are not in I'_{R} is bounded by

$$\sum_{j=0}^{k} \frac{4\delta N_j}{2^j} \leq 4 \cdot \frac{1}{8}\varepsilon \cdot 2 \cdot \sum_{j=0}^{k} \frac{N_j}{2^{j+1}} \leq \varepsilon \cdot \mathsf{SIZE}(I_{\mathrm{B}})\,,$$

where we use (1) in the last inequality. We obtain that $\mathsf{SIZE}(I_{\mathrm{R}}) \leq \mathsf{SIZE}(I'_{\mathrm{R}}) + \varepsilon \cdot \mathsf{SIZE}(I_{\mathrm{B}}) \leq (1+\varepsilon) \cdot \mathsf{SIZE}(I_{\mathrm{B}})$. We conclude that properties (P1)-(P4) hold for the rounded instance I_{R}. □

A.2 Proof of Lemma 2

Lemma 2. *Let I be given as a stream of items. Suppose that $0 < \varepsilon \leq \frac{1}{3}$, that the rounded instance I_{R}, created from I, satisfies properties (P1)-(P4), and that the solution $\mathcal{S}$ of I_{R} uses at most $\mathsf{OPT}(I_{\mathrm{R}}) + \sigma$ bins. Let $\mathsf{ALG}(I)$ be the number of bins that our algorithm outputs. Then, it holds that $\mathsf{OPT}(I) \leq \mathsf{ALG}(I) \leq (1+3\varepsilon) \cdot \mathsf{OPT}(I) + \sigma + \mathcal{O}\left(\log \frac{1}{\varepsilon}\right)$.*

Proof. We analyze the two cases of the algorithm:

Case $s \leq W$: In this case, small items fit into the bins of $\mathcal{S}$ and $\mathsf{ALG}(I) = |\mathcal{S}|$. For the inequality $\mathsf{OPT}(I) \leq \mathsf{ALG}(I)$, observe that the packing $\mathcal{S}$ can be used as a packing of items in I_{B} (in a straightforward way) with no less free space for small items by property (P2). Thus $\mathsf{OPT}(I) \leq |\mathcal{S}|$.

To upper bound $\mathsf{ALG}(I)$, note that

$$|\mathcal{S}| \leq \mathsf{OPT}(I_{\mathrm{R}}) + \sigma \leq (1+\varepsilon) \cdot \mathsf{OPT}(I_{\mathrm{B}}) + \mathcal{O}\left(\log \frac{1}{\varepsilon}\right) + \sigma \leq (1+\varepsilon) \cdot \mathsf{OPT}(I) + \mathcal{O}\left(\log \frac{1}{\varepsilon}\right) + \sigma\,,$$

where the second inequality follows from property (P3) and the third inequality holds as I_{B} is a subinstance of I.

Case $s > W$: Recall that $\mathsf{ALG}(I) = |\mathcal{S}| + \lceil s'/(1-\varepsilon) \rceil$. We again have that $\mathcal{S}$ can be used as a packing of I_{B} with no less free space for small items. Thus, the total size of small items that do not fit into bins in $\mathcal{S}$ is at most s' and these items clearly fit into $\lceil s'/(1-\varepsilon) \rceil$ bins. Hence, $\mathsf{OPT}(I) \leq |\mathcal{S}| + \lceil s'/(1-\varepsilon) \rceil$.

For the other inequality, consider starting with solution $\mathcal{S}$ for I_{R}, first to (almost) fill up the bins of $\mathcal{S}$ with small items of total size W, then using $\lceil s'/(1-\varepsilon) \rceil$ additional bins for the remaining small items. Note that in each bin, except the last one, the unused space is less than ε, thus the total size of items in I_{R} and small items is more than $(\mathsf{ALG}(I) - 1) \cdot (1-\varepsilon)$. Finally, we replace items in I_{R} by items in I_{B} and the total size of items decreases by $\mathsf{SIZE}(I_{\mathrm{R}}) - \mathsf{SIZE}(I_{\mathrm{B}}) \leq \varepsilon \cdot \mathsf{SIZE}(I_{\mathrm{B}}) \leq \varepsilon \cdot \mathsf{SIZE}(I)$ by property (P4). Hence, $\mathsf{SIZE}(I) \geq (\mathsf{ALG}(I) - 1) \cdot (1-\varepsilon) - \varepsilon \cdot \mathsf{SIZE}(I)$. Rearranging and using $\varepsilon \leq \frac{1}{3}$, we get

$$\mathsf{ALG}(I) \leq \frac{1+\varepsilon}{1-\varepsilon} \cdot \mathsf{SIZE}(I) + 1 \leq (1+3\varepsilon) \cdot \mathsf{OPT}(I) + 1\,.$$

Considered together, these two cases both meet the claimed bound. □

B Vector Bin Packing

As already observed by Fernandez de la Vega and Lueker [18], a $1+\varepsilon$-approximation algorithm for (scalar) BIN PACKING implies a $d \cdot (1+\varepsilon)$-approximation algorithm for VECTOR BIN PACKING, where items are d-dimensional vectors and bins have capacity d in every dimension. Indeed, we split the vectors into d groups according to the largest dimension (chosen arbitrarily among dimensions that have the largest value) and in each group we apply the approximation scheme for BIN PACKING, packing just according to the largest dimension. Finally, we take the union of opened bins over all groups. Since the optimum of the BIN PACKING instance for each group is a lower bound on the optimum of VECTOR BIN PACKING, we get that that the solution is a $d \cdot (1+\varepsilon)$-approximation.

This can be done in the same way in the streaming model. Hence there is a streaming algorithm for VECTOR BIN PACKING which outputs a $d \cdot (1+\varepsilon)$-approximation of OPT, the offline optimal number of bins, using $\mathcal{O}\left(\frac{d}{\varepsilon} \cdot \log \frac{1}{\varepsilon} \cdot \log \mathsf{OPT}\right)$ memory. By scaling ε, there is a $d+\varepsilon$-approximation algorithm with $\widetilde{\mathcal{O}}(\frac{d^2}{\varepsilon})$ memory. We can, however, do better by one factor of d.

Theorem 6. *There is a streaming $d+\varepsilon$-approximation for* VECTOR BIN PACKING *algorithm that uses $\mathcal{O}\left(\frac{d}{\varepsilon} \cdot \log \frac{d}{\varepsilon} \cdot \log \mathsf{OPT}\right)$ memory.*

Proof. Given an input stream I of vectors, we create an input stream I' for BIN PACKING by replacing each vector $\mathbf{v}$ by a single (scalar) item a of size $\|\mathbf{v}\|_\infty$. We use our streaming algorithm for BIN PACKING with precision $\delta = \frac{\varepsilon}{d}$ which uses $\mathcal{O}\left(\frac{1}{\delta} \cdot \log \frac{1}{\delta} \cdot \log \mathsf{OPT}\right)$ memory and returns a solution with at most $B = (1+\delta) \cdot \mathsf{OPT}(I') + \widetilde{\mathcal{O}}(\frac{1}{\delta})$ scalar bins. Clearly, B bins are sufficient for the stream I of vectors, since in the solution for I' we replace each item by the corresponding vector and obtain a valid solution for I.

Finally, we show that $(1+\delta) \cdot \mathsf{OPT}(I') + \mathcal{O}_\delta(1) \leq (d+\varepsilon) \cdot \mathsf{OPT}(I) + \widetilde{\mathcal{O}}(\frac{d}{\varepsilon})$ for which it is sufficient to prove that $\mathsf{OPT}(I') \leq d \cdot \mathsf{OPT}(I)$ as $\delta = \frac{\varepsilon}{d}$. Namely, from an optimal solution $\mathcal{S}$ for I, we create a solution for I' with at most $d \cdot \mathsf{OPT}(I)$ bins. For each bin B in $\mathcal{S}$, we split the vectors assigned to B into d groups according to the largest dimension (chosen arbitrarily among those with the largest value) and for each group i we create bin B_i with vectors in group i. Then we just replace each vector v by an item of size $\|v\|_\infty$ and obtain a valid solution for I' with at most $d \cdot \mathsf{OPT}(I)$ bins. □

Interestingly, a better than d-approximation using sublinear memory, which is rounding-based, is not possible, due to the following result in [7]. (Note that the result requires that the numbers in the input vectors can take arbitrary values in $[0,1]$, i.e., vectors do not belong to a bounded universe.)

Theorem 7 (Implied by the proof of Theorem 2.2 in [7])**.** *Any algorithm for* VECTOR BIN PACKING *that rounds up large coordinates of vectors to $o(N/d)$ types cannot achieve better than d-approximation, where N is the number of vectors.*

It is an interesting open question whether or not we can design a streaming $d+\varepsilon$-approximation with $o(\frac{d}{\varepsilon})$ memory or even with $\widetilde{\mathcal{O}}\left(d + \frac{1}{\varepsilon}\right)$ memory.

C Tight Example for the Algorithm for Vector Scheduling

For any $m \geq 2$, we present an instance I in $d = m+1$ dimensions such that $\mathsf{OPT}(I) = 1$, but $\mathsf{OPT}(I_{\mathrm{R}}) \geq 2 - \frac{1}{m}$, where I_{R} is the instance created by our algorithm described in Section 4.

Let γ be as in the algorithm and assume for simplicity that $\frac{1}{\gamma}$ is an integer. First, m big jobs with vectors $\mathbf{v^1}, \ldots, \mathbf{v^m}$ arrive, where $\mathbf{v^i}$ is a vector with dimensions i and $m+1$ equal to 1 and with zeros in the other dimensions (that is, $\mathbf{v^i_i} = 1$ and $\mathbf{v^i_{m+1}} = 1$, while $\mathbf{v^i_k} = 0$ for $k \notin \{i, m+1\}$). Then, small

jobs arrive in groups of $d-1=m$ jobs and there are $(m-1)\cdot\frac{1}{\gamma}$ groups. Each group consists of items $(\gamma,0,\ldots,0,0),(0,\gamma,\ldots,0,0),\ldots,(0,0,\ldots,\gamma,0)$, i.e, for each $i=1,\ldots,d-1$, it contains one item with value γ in coordinate i and with zeros in other dimensions. The groups arrive one by one, with an arbitrary ordering inside the group. Note, however, that these jobs with ℓ_∞ norm equal to γ become small for the algorithm only once the first job from the last group arrives as they are compared to the total load in each dimension, which increases gradually. When they become small, the algorithm will combine each group into one container $(\gamma,\gamma,\ldots,\gamma,0)$, which can be achieved by processing the jobs in their arrival order and by having the last vector of the group larger by an infinitesimal amount (we do not take these infinitesimals into account in further calculations). Thus, I_{R} consists of m big jobs and $(m-1)\cdot\frac{1}{\gamma}$ containers $(\gamma,\gamma,\ldots,\gamma,0)$.

Observe that $\mathsf{OPT}(I)=1$, since in the optimal solution, each machine i is assigned big job $\mathbf{v^i}$ and $\frac{1}{\gamma}$ small jobs with γ in dimension k for each $k\in\{1,\ldots,d-1\}\setminus\{i\}$. Thus the load equals one on any machine and dimension.

We claim that $\mathsf{OPT}(I_{\mathrm{R}})\geq 2-\frac{1}{m}$. Indeed, only one big job can be assigned on one machine, as all of them have value one in dimension $m+1$, so each machine contains one big job. Observe that some machine gets at least $\frac{m-1}{m}\cdot\frac{1}{\gamma}$ containers and thus, it has load of at least $2-\frac{1}{m}$ in one of the $d-1$ first dimensions, which shows the claim.

Note that for this instance to show ratio $2-\frac{1}{m}$ it suffices that the algorithms creates $(m-1)\cdot\frac{1}{\gamma}$ containers $(\gamma,\gamma,\ldots,\gamma,0)$. This can be enforced for various greedy algorithms used for packing the small jobs into containers. We conclude that we need a different approach for input summarization to get a ratio below $2-\frac{1}{m}$.

On the other hand, it remains open whether or not the algorithm in Section 4 with $\gamma=\Theta(\varepsilon)$ also gives $(2-\frac{1}{m}+\varepsilon)$-approximation, which would imply a better space bound of $\mathcal{O}(\frac{1}{\varepsilon}\cdot d^2\cdot m)$.

D Rounding Algorithms for Vector Scheduling in a Constant Dimension

Makespan Scheduling. We start by outlining a simple streaming algorithm for $d=1$ based on rounding. Here, each job j on input is characterized by its processing time p_j only. The algorithm uses the size of the largest job seen so far, denoted $p_{\max}$, as a lower bound on the optimum makespan. This makes the rounding procedure (and hence, the input summary) oblivious of m, the number of machines, which is in contrast with the algorithm in Section 4 that uses just the sum of job sizes as the lower bound.

The rounding works as follows: Let q be an integer such that $p_{\max}\in((1+\varepsilon)^q,(1+\varepsilon)^{q+1}]$, and let $k=\lceil\log_{1+\varepsilon}\frac{1}{\varepsilon}\rceil=\mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$. A job is *big* if its size exceeds $(1+\varepsilon)^{q-k}$; note that any big job is larger than $\varepsilon\cdot p_{\max}/(1+\varepsilon)^2$. All other jobs are *small* and have size less than $\varepsilon\cdot p_{\max}$. The algorithm maintains one variable s for the total size of all small jobs and variables L_i, $i=q-k,\ldots,q$, for the number of big jobs with size in $((1+\varepsilon)^i,(1+\varepsilon)^{i+1}]$ (note that this interval is *not* scaled by $p_{\max}$, i.e., increasing $p_{\max}$ slightly does not move the intervals).

Maintaining these variables when a new job arrives can be done in a straightforward way. In particular, when an increase of $p_{\max}$ causes that q increases (by 1 or more as it is integral), we discard all variables L_i that do not correspond to big jobs any more, and account for previously big jobs that are now small in variable s. However, as the size of these jobs was rounded to a power of $1+\varepsilon$, variable s can differ from the exact total size of small jobs by a factor of at most $1+\varepsilon$.

The created input summary, consisting of $\mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ variables L_i and variable s, preserves the optimal value up to a factor of $1+\mathcal{O}(\varepsilon)$. This follows, since big jobs are stored with size rounded up to the nearest power of $1+\varepsilon$, and, although we just know the approximate total size of small jobs, they can be taken into account similarly as when calculating a bound on the number of bins in our algorithm for BIN PACKING.

Vector Scheduling. We describe the rounding introduced by Bansal *et al.* [8], which we can adjust into a streaming $1+\varepsilon$-approximation for VECTOR SCHEDULING in constant dimension. The downside of this

approach is that it requires memory exceeding $\left(\frac{2}{\varepsilon}\right)^d$, which becomes unfeasible even for $\varepsilon = 1$ and d being a relatively small constant. Moreover, such an amount of memory may be needed also in the case of a small number of machines.

We first use the following lemma by Chekuri and Khanna [11], where $\delta = \frac{\varepsilon}{d}$:

Lemma 8 (Lemma 2.1 in [11])**.** *Let I be an instance of* VECTOR SCHEDULING*. Let I' be a modified instance where we replace each vector $\mathbf{v}$ by vector $\mathbf{v}'$ as follows: For each $1 \leq i \leq d$, if $\mathbf{v}_i > \delta\|\mathbf{v}\|_\infty$, then $\mathbf{v'}_i = \mathbf{v}_i$; otherwise, $\mathbf{v'}_i = 0$. Let $\mathcal{S}'$ be any solution for I'. Then, if we replace each vector $\mathbf{v}'$ in $\mathcal{S}'$ by its counterpart in I, we get a solution of I with makespan at most $1+\varepsilon$ times the makespan of $\mathcal{S}'$.*

In the following, we assume that the algorithms receives vectors from instance I', created as in Lemma 8. Let $p_{\max}$ be the maximum ℓ_∞ norm over all vectors that arrived so far; we use it as a lower bound on OPT. We again do not use the total volume in each dimension as a lower bound, which makes the input summarization oblivious of m. A job, characterized by vector $\mathbf{v}$, is said to be *big* if $\|\mathbf{v}\|_\infty > \delta \cdot p_{\max}$; otherwise, $\mathbf{v}$ is *small*.

We round all values in big jobs to the powers of $1+\varepsilon$. By Lemma 8, we have that either $\mathbf{v}_k > \delta^2 \cdot p_{\max}$ or $\mathbf{v}_k = 0$ for any big $\mathbf{v}$ and dimension k, thus there are $\left\lceil \log_{1+\varepsilon} \frac{1}{\delta^2} \right\rceil^d = \mathcal{O}\left(\left(\frac{2}{\varepsilon}\log\frac{d}{\varepsilon}\right)^d\right)$ types of big jobs at any time. We have one variable $L_{\mathbf{t}}$ counting the number of jobs for each big type $\mathbf{t}$, where $\mathbf{t}$ is an integer vector consisting of the exponents, i.e., if $\mathbf{v}$ is a big vector of type $\mathbf{t}$, then $\mathbf{v}_i \in \left((1+\varepsilon)^{\mathbf{t}_i}, (1+\varepsilon)^{\mathbf{t}_i+1}\right]$ (we set $\mathbf{t}_i = -\infty$ if $\mathbf{v}_i = 0$). As in the 1-dimensional case, big types change over time, when $p_{\max}$ (sufficiently) increases.

Note that small jobs cannot be rounded to powers of $1+\varepsilon$ directly. Instead, they are rounded relative to their ℓ_∞ norms. More precisely, consider a small vector $\mathbf{v}$ and let $\gamma = \|\mathbf{v}\|_\infty$. For each dimension k, if $\mathbf{v}_k > 0$, let $\mathbf{t}_k \geq 0$ be the largest integer such that $\mathbf{v}_k \leq \gamma \cdot (1+\varepsilon)^{-\mathbf{t}_i}$, and if $\mathbf{v}_i = 0$, we set $\mathbf{t}_i$ to ∞. Then $(\mathbf{t}_1, \ldots, \mathbf{t}_d)$ is the type of small vector $\mathbf{v}$. Observe that small types do not change over time and there are at most $\mathcal{O}\left(\left(\frac{1}{\varepsilon}\log\frac{d}{\varepsilon}\right)^d\right)$ of them. For each small type $\mathbf{t}$, we have one variable $s_{\mathbf{t}}$ counting the sum of the ℓ_∞ norms of all small jobs of that type.

The variables can be maintained in an online fashion. Namely, when $p_{\max}$ increases, the types for previously big jobs that are now small are discarded, while the jobs that become small are accounted for in small types. For each such former big type $\mathbf{t}$, we compute the corresponding small type as follows: Let $\delta = \|\mathbf{t}\|_\infty$ be the maximum value in $\mathbf{t}$ (which is not $-\infty$). The corresponding small type $\hat{\mathbf{t}}$ has then $\hat{\mathbf{t}}_i = \delta - \mathbf{t}_i$ if $\mathbf{t}_i \neq -\infty$, and $\hat{\mathbf{t}}_i = \infty$ otherwise. Then we increase $s_{\hat{\mathbf{t}}}$ by $L_{\mathbf{t}} \cdot (1+\varepsilon)^{\delta+1}$.

There are two types of errors introduced due to maintaining variables in the streaming scenario and not offline, where we know the final value of $p_{\max}$ in advance. First, it may happen that a vector $\mathbf{v}$ that was big upon its arrival becomes small, and the small type of $\mathbf{v}$ is different than the small type computed for the former big type of $\mathbf{v}$ (i.e., the small type of $\mathbf{v}$ with values rounded to powers of $1+\varepsilon$). Second, the sum of ℓ_∞ norms of small vectors of a small type $\mathbf{t}$ is in $(s_{\mathbf{t}}/(1+\varepsilon), s_{\mathbf{t}}]$, and moreover, the error in some dimension i with $\mathbf{t}_i > 0$ (i.e., not the largest one for this type) may be of factor up to $(1+\varepsilon)^2$, since we may round such a dimension two times for some jobs. Note, however, that by giving up a factor of $1+\mathcal{O}(\varepsilon)$, we may disregard both issues.

The offline algorithm of Bansal *et al.* [8] implies that such an input summary, consisting of variables for both small and big types, is sufficient for computing $1+\varepsilon$-approximation.