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.

Keywords: Streaming algorithms · Bin packing · Scheduling.

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 [37] and McGregor [36]), there has been little attention paid to the core problems

[*] The work is supported by European Research Council grant ERC-2014-CoG 647557.

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 $s_1, \dots, s_N$ (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 $\{1, \dots, N\}$ into subsets $B_1, \dots, B_m$, 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 [25] with techniques for approximation schemes [17,32]. 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 [27]. 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 $\mathcal{O}\left(\frac{d}{\varepsilon}\cdot\log\frac{d}{\varepsilon}\cdot\log\mathsf{OPT}\right)$. This is done by a reduction to the 1-dimensional case and using the aforementioned streaming algorithm; the details are deferred to the full version of the paper. We remark that if vectors are rounded into a sublinear number of types, then better than d-approximation is not possible [7].

[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.

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.

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 $2-\frac{1}{m}$. 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 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 [17]. Karmarkar and Karp [32] provided an algorithm which returns a solution with $\mathsf{OPT}+\mathcal{O}(\log^2\mathsf{OPT})$ bins. Recently, Hoberg and Rothvoß [27] 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

[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.

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 [23] 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 [31], 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 [21,22] 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 $d = 2$ [39], 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. A streaming algorithm can postpone such decisions to 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 [34], which for some integer K classifies items into size groups $(0, \frac{1}{K}], (\frac{1}{K}, \frac{1}{K-1}], \dots, (\frac{1}{2}, 1]$. 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 [34], which can be seen as the intersection of online and streaming models.

For VECTOR BIN PACKING, the best known competitive ratio of $d + 0.7$ [19] 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 [20], 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 [30] 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 state-of-the-art approximation ratio for large d is $\mathcal{O}(\log d/(\log \log d))$ [26,29], 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 $(1.88, 1.9201)$ [1,24,28,18], 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.* [29] 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 $\Omega(\log d/(\log\log d))$ [29,4], 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 [29] 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, $\mathsf{SIZE}(I) \le \mathsf{OPT}(I)$. 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 $\mathcal{O}\big(f(\frac{1}{\varepsilon}) \cdot \mathrm{polylog}\,\frac{1}{\varepsilon} \cdot \mathrm{polylog}\,\mathsf{OPT}(I)\big)$.

Overview. We first briefly describe the approximation scheme of Fernandez de la Vega and Lueker [17], 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 [17], 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 $\mathsf{OPT}(I_{\mathrm{B}}) \le \mathsf{OPT}(I_{\mathrm{R}}) \le (1+\varepsilon) \cdot \mathsf{OPT}(I_{\mathrm{B}})$, 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 [32] 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* $\sigma = \mathcal{O}\left(\frac{1}{\varepsilon} \cdot \log \frac{1}{\varepsilon} \cdot \log \mathit{OPT}(I_{\mathrm{B}})\right)$.

Before describing the rounding itself, 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 [21,22], 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 [23]; 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.

It holds 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}. The proof is deferred to the full version of the paper.

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 $\mathit{OPT}(I_{\mathrm{R}}) + \sigma$ bins. Let $\mathit{ALG}(I)$ be the number of bins that our algorithm outputs. Then, it holds that $\mathit{OPT}(I) \leq \mathit{ALG}(I) \leq (1+3\varepsilon)\cdot \mathit{OPT}(I) + \sigma + \mathcal{O}\left(\log \frac{1}{\varepsilon}\right)$.*

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 [25]. Given a precision $\delta > 0$ and an input stream of numbers $s_1, \dots, s_N$, 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.[3] 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.[4] 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 practice the space used is observed to scale linearly with $\frac{1}{\delta}$ [35]. (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

[3] Note that if s appears more times in the stream, its rank is an interval rather than a single number. Also, unlike in [25], we order numbers non-increasingly, which is more convenient for BIN PACKING.

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

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_B). 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 $q = \mathcal{O}(\frac{1}{\delta} \cdot \log \delta N_\mathrm{B}) = \mathcal{O}\big(\frac{1}{\varepsilon^2} \cdot \log(\varepsilon \cdot \mathsf{OPT}(I_\mathrm{B}))\big)$. 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.)

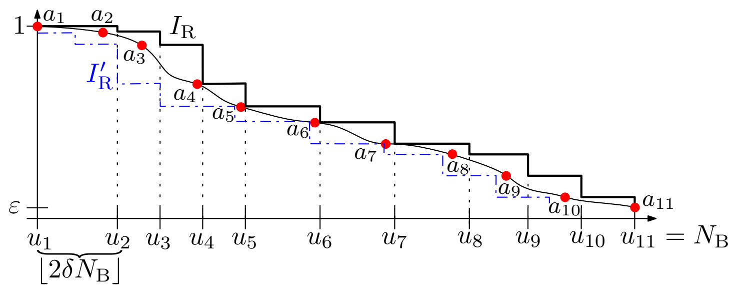

Figure 1. An illustration of the original distribution of sizes of big items in I_B, depicted by a smooth curve, and the distribution of item sizes in the rounded instance I_R, depicted by a bold "staircase" function. The distribution of I'_R (which is I_R without the $\lfloor 4\delta N_\mathrm{B} \rfloor$ 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.

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 [32].

The full description of rounding, which also gives the proof of Lemma 1, is deferred to the full version of the paper. 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 [25] allows us to obtain a streaming approximation scheme for Bin Packing. We argue that this connection runs deeper.

We start with two scenarios for which there exist better quantile summaries, thus implying a better space bound for achieving a streaming $1+\varepsilon$-approximation for Bin Packing in a similar way as in Section 3.1. First, if all big item sizes belong to a universe $U \subset (\varepsilon, 1]$, known in advance, then it can be better to use the quantile summary of Shrivastava *et al.* [38], which provides a guarantee of $\mathcal{O}(\frac{1}{\delta} \cdot \log |U|)$ on the space complexity, where δ is the precision requirement. 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 [33], 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.

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 is deferred to the full version. 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 1 (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 ($N = \mathcal{O}(\mathsf{OPT})$ in our reduction). Note that our algorithm is comparison-based if we employ the comparison-based quantile summary of Greenwald and Khanna [25], 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, 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 $\mathcal{O}\left(\log \frac{1}{\varepsilon}\right)$.

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 also bounded. 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 $k = 1, \dots, d$, it keeps track of the total load of all jobs in dimension k, denoted L_k. Note that the optimal makespan satisfies $\mathsf{OPT} \geq \max_k \frac{1}{m} \cdot L_k$ (an alternative lower bound on OPT is the maximum ℓ_∞ norm of a job seen so far, but our algorithm does not use this). For brevity, let $\mathsf{LB} = \max_k \frac{1}{m} \cdot L_k$.

Let $\gamma = \Theta\left(\varepsilon^2 / \log \frac{d^2}{\varepsilon}\right)$; the constant hidden in Θ follows from the analysis. We say that a job with vector $\mathbf{v}$ is *big* if $\|\mathbf{v}\|_\infty > \gamma \cdot \mathsf{LB}$; 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 $\|\mathbf{c}\|_\infty \leq 2\gamma \cdot \mathsf{LB}$. Furthermore, container $\mathbf{c}$ is *closed* if $\|\mathbf{c}\|_\infty > \gamma \cdot \mathsf{LB}$, 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 an incoming small job into the open container, where it always fits as any small vector $\mathbf{v}$ satisfies $\|\mathbf{v}\|_\infty \leq \gamma \cdot \mathsf{LB}$. All containers are retained in the memory.

When a new job vector $\mathbf{v}$ arrives, we update the values of L_k for $k = 1, \dots, d$ (by adding $\mathbf{v}_k$) and also of LB. If LB increases, any previously big job $\mathbf{u}$ that has become small (w.r.t. new LB), is considered to be an open container. Moreover, it may happen that a previously closed container $\mathbf{c}$ becomes open again, i.e., $\|\mathbf{c}\|_\infty \leq \gamma \cdot \mathsf{LB}$. If we indeed have more open containers, we keep aggregating arbitrary two open containers as long as we have at least two of them. Finally, if the new job $\mathbf{v}$ is small, we add it in an open container (if there is no open container, we first open a new, empty one). 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 can assume that $\mathsf{LB} = \max_k \frac{1}{m} \cdot L_k = 1$, which implies that $\mathsf{OPT} \geq 1$. This is without loss of generality by scaling every quantity by $1/\mathsf{LB}$. 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 prove that $\mathsf{OPT}(I_\mathrm{R})$ and $\mathsf{OPT}(I)$ are close together, up to a factor of $2 - \frac{1}{m} + \varepsilon$. 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 3. *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) $\mathsf{OPT}(I) \leq \mathsf{OPT}(I_\mathrm{R}) \leq \left(2 - \frac{1}{m} + 3\varepsilon\right) \cdot \mathsf{OPT}(I)$.

The full proof is deferred to the full version; here we give its brief outline. To show (i), we obtain the solution with the desired load from the randomized online algorithm by Im *et al.* [29]. 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, namely, that it creates a nearly balanced assignment as claimed in (i). Item (ii) follows from an arbitrary combination of an optimal solution for big jobs in I_{R} only (i.e., excluding containers) and the solution ensured by (i) for containers.

It remains open whether or not the above algorithm with $\gamma = \Theta(\varepsilon)$ also gives $(2 - \frac{1}{m} + \varepsilon)$-approximation, which would imply a better space bound of $\mathcal{O}(\frac{1}{\varepsilon} \cdot d^2 \cdot m)$. The approximation guarantee of this approach cannot be improved, however, which we demonstrate by an example in the full version of the paper. More importantly, it would be interesting to know whether or not there is a streaming $\mathcal{O}(1)$-approximation with space $\mathrm{poly}(\frac{1}{\varepsilon}, d, \log m)$, that is, polylogarithmic in m, or even independent of m. Recall that rounding from [8] achieves space independent of m, but exponential in d.

Acknowledgments. 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. 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.
18. Rudolf Fleischer and Michaela Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000.
19. 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.
20. Michael R Garey and David S Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. Freeman, 1979.
21. Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.
22. 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.
23. 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.
24. 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.
25. 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.
26. 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.
27. 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.
28. J.F. Rudin III. *Improved bounds for the on-line scheduling problem*. PhD thesis, The University of Texas at Dallas, 2001.
29. S. Im, N. Kell, J. Kulkarni, and D. Panigrahi. Tight bounds for online vector scheduling. *SIAM Journal on Computing*, 48(1):93–121, 2019.
30. 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.

31. David S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.
32. 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.
33. 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.
34. C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32:562–572, July 1985.
35. 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.
36. Andrew McGregor. Graph stream algorithms: A survey. *SIGMOD Rec.*, 43(1):9–20, May 2014.
37. Shanmugavelayutham Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.
38. 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.
39. Gerhard J. Woeginger. There is no asymptotic PTAS for two-dimensional vector packing. *Information Processing Letters*, 64(6):293 – 297, 1997.