Theory Comput Syst (2011) 48: 79–92
DOI 10.1007/s00224-009-9226-2

Improved Results for a Memory Allocation Problem

Leah Epstein · Rob van Stee

Published online: 1 August 2009

Abstract We consider a memory allocation problem. This problem can be modeled as a version of bin packing where items may be split, but each bin may contain at most two (parts of) items. This problem was recently introduced by Chung et al. (Theory Comput. Syst. 39(6):829–849, 2006). We give a simple $\frac{3}{2}$-approximation algorithm for this problem which is in fact an online algorithm. This algorithm also has good performance for the more general case where each bin may contain at most k parts of items. We show that this general case is strongly NP-hard for any $k \geq 3$. Additionally, we design an efficient approximation algorithm, for which the approximation ratio can be made arbitrarily close to $\frac{7}{5}$.

Keywords Approximation algorithms · Bin packing · Cardinality constraints · Splittable items

1 Introduction

Parallel processing requires the allocation of available memory to the processors. This needs to be done in such a way that each processor has access to sufficient memory,

A preliminary version of this paper appeared in the Proceedings of Tenth Workshop on Algorithms and Data Structures (WADS 2007). LNCS, vol. 4619, pp. 362–373. Springer, 2007.

Work performed while R. van Stee was at University of Karlsruhe, Germany. Research supported by Alexander von Humboldt Foundation.

L. Epstein
Department of Mathematics, University of Haifa, 31905 Haifa, Israel
e-mail: lea@math.haifa.ac.il

R. van Stee (✉)
Max-Planck-Institut für Informatik, Saarbrücken, Germany
e-mail: vanstee@mpi-inf.mpg.de

and the amount of wasted memory is as small as possible. If processors have memory requirements that vary wildly over time, any memory allocation where a single memory can only be accessed by one processor will be inefficient. A solution to this problem would be to allow memory sharing between processors. However, if there is a single shared memory for all the processors, there will be much contention, which is also undesirable. It is currently infeasible to build a large, fast shared memory, and in practice, such memories are time-multiplexed. For n processors, this increases the effective memory access time by a factor of n.

Chung et al. [3] studied this problem and described the drawbacks of the methods stated above. Moreover, they suggested a new architecture where each memory may be accessed by at most *two* processors, avoiding the disadvantages of the two extreme earlier models. They abstract the memory allocation problem as a bin packing problem, where the bins are the memories and the items to be packed represent the memory requirements of the processors. This means that the items may be of any size (in particular, they can be larger than 1, which is the size of a bin), and an item may be split, but each bin may contain at most two parts of items.

We study approximation algorithms in terms of the *absolute approximation ratio* or the *absolute performance guarantee*. Let $\mathcal{B}(\mathcal{I})$ (or $\mathcal{B}$, if the input $\mathcal{I}$ is clear from the context), be the cost of algorithm $\mathcal{B}$ on the input $\mathcal{I}$. An algorithm $\mathcal{A}$ is an $\mathcal{R}$-approximation (with respect to the absolute approximation ratio) if for every input $\mathcal{I}$, $\mathcal{A}(\mathcal{I}) \leq \mathcal{R} \cdot \mathrm{OPT}(I)$, where OPT is an optimal algorithm for the problem. The absolute approximation ratio of an algorithm is the infimum value of $\mathcal{R}$ such that the algorithm is an $\mathcal{R}$-approximation. The *asymptotic* approximation ratio or *asymptotic* performance guarantee of an algorithm $\mathcal{A}$ is defined to be

$$\mathcal{R}_{\mathcal{A}}^{\infty} = \limsup_{N\to\infty} \sup_{\mathcal{I}} \left\{ \frac{\mathcal{A}(\mathcal{I})}{\mathrm{OPT}(\mathcal{I})} \middle| \mathrm{OPT}(\mathcal{I}) = N \right\}.$$

The authors of [3] give a $\frac{3}{2}$-approximation for the memory allocation problem which they study. We continue the study of this problem and also consider a generalized variant where items can still be split arbitrarily, but each bin can contain up to k parts of items, for a given value of $k \geq 2$.

Bin packing algorithms are often studied using the asymptotic approximation ratio. The reason for that is that for most bin packing problems, a simple reduction from the PARTITION problem (see problem SP12 in [7]) shows that no polynomial-time algorithm has an absolute performance guarantee which is strictly smaller than $\frac{3}{2}$ (unless P = NP). However, since in our problem items can be split, but cannot be packed more than a given number of parts to a bin, this reduction is not valid. In [3], the authors show that the problem which they study is NP-hard in the strong sense for $k = 2$. They use a reduction from the 3-PARTITION problem (see problem [SP15] in [7]). Their result does not seem to imply any consequences with respect to hardness of approximation.

Independently of our work and simultaneously with it, Mao and Graham [12] analyzed the asymptotic approximation ratio of several algorithms, giving upper bounds of 1.498 for $k = 2$, $\frac{3}{2}$ for $k = 3$ and $2 - 2/k$ for $k \geq 4$. They also showed an upper bound of $2 - 1/k$ on the (asymptotic) approximation ratio of NEXT FIT and a lower bound of $1 + (k + \frac{1}{k+1})^{-1}$ on the approximation ratio of any online algorithm.

A related problem is known as bin packing with cardinality constraints. In this problem, all items have a size of at most 1 as in regular bin packing, and the items cannot be split, however there is an upper bound of k on the number of items that can be packed into a single bin. This problem was studied with respect to the asymptotic approximation ratio. It was introduced and studied in an offline environment as early as in 1975 by Krause, Shen and Schwetman [10, 11]. They showed that the performance guarantee of the well known FIRST FIT algorithm is at most $2.7 - \frac{12}{5k}$. Additional results were offline approximation algorithms of performance guarantee 2. These results were later improved in two ways. Kellerer and Pferschy [9] designed an improved offline approximation algorithm with performance guarantee $\frac{3}{2}$. Finally an APTAS was designed in [2] (for a more general problem), and an AFPTAS was designed in [5].

For the same problem, Babel et al. [1] designed a simple *online* algorithm with asymptotic approximation ratio 2 for any value of k. They also designed improved algorithms for $k = 2, 3$ of asymptotic approximation ratios $1 + \frac{\sqrt{5}}{5} \approx 1.44721$ and 1.8 respectively. The same paper [1] also proved an almost matching lower bound of $\sqrt{2} \approx 1.41421$ for $k = 2$ and mentioned that the lower bounds of [15, 16] for the classic problem hold for cardinality constrained bin packing as well. The lower bound of $\frac{3}{2}$ given by Yao [16] holds for small values of $k > 2$ and the lower bound of 1.5401 given by Van Vliet [15] holds for sufficiently large k. No other lower bounds are known. Finally, Epstein [4] gave an optimal online bounded space algorithm (i.e., an algorithm which can have a constant number of active bins at every time) for this problem. Its asymptotic worst-case ratio is an increasing function of k and tends to $1 + h_\infty \approx 2.69103$, where h_∞ is the best possible performance guarantee of an online bounded space algorithm for regular bin packing (without cardinality constraints). Additionally, she improved the online upper bounds for $3 \leq k \leq 6$. In particular, the upper bound for $k = 3$ was improved to $\frac{7}{4}$.

Another related problem was studied recently by Shachnai, Tamir and Yehezkely [14]. They considered an offline bin packing problem where items may be split arbitrarily. However, to make the problem non-trivial, there are some restrictions. In one model, each part of a split item increases by a constant additive factor. Another variant gives an upper bound on the number of split items. They showed that both these problems do not admit an approximation algorithm with an constant additive error, unless P = NP. They designed a dual PTAS, an APTAS and a dual AFPTAS for each one of the problems. An AFPTAS for each one of the problems was designed in [13]. Their problem is different from our problem since in their case all items have a size of at most 1. In their case it is possible to exploit the existence of simple structures of optimal solutions, which are more complicated in our case.

Our Results In the current paper, we begin by showing that this problem is NP-hard in the strong sense for any fixed value of k. This generalizes a result from Chung et al. [3]. We also show that the simple NEXT FIT algorithm has an absolute approximation ratio of $2 - 1/k$. Note that Mao and Graham [12] prove only an *asymptotic* upper bound of $2 - 1/k$ for NEXT FIT. Finally, we give an efficient approximation algorithm for $k = 2$. The approximation ratio of this algorithm can be made arbitrarily close to $7/5$.

2 NP-hardness of the Problem (in the Strong Sense)

Theorem 1 *Packing splittable items with a cardinality constraint of k parts of items per bin is NP-hard in the strong sense for any fixed* $k \geq 3$.

Proof Given a fixed value of k, we show a reduction from the 3-Partition problem defined as follows (see problem [SP15] in [7]). We are given a set of $3m$ positive integers $s_1, s_2, \ldots, s_{3m}$ such that $\sum_{j=1}^{3m} s_j = mB$ and each s_i satisfies $\frac{B}{4} < s_i < \frac{B}{2}$. The goal is to find out whether there exists a partition of the numbers into m sets, where each set contains three items, and the sum of elements of each set is exactly B. The 3-Partition problem is known to be NP-hard in the strong sense.

Given such an instance of the 3-Partition problem, we define an instance of the splittable item packing with cardinality constraints as follows. A first set of items contains $m(k-3)$ identical items of size $\frac{3k-1}{3k(k-3)}$ (for $k=3$, no items are defined at this point). These items are called padding items. In addition, there are $3m$ items, where item j has size $\frac{s_j}{3kB}$ (for $k=3$ we define the size to be $\frac{s_j}{B}$). These items are called adapted items. The goal is to find a packing with exactly m bins. Since there are mk items, clearly a solution which splits items must use at least $m+1$ bins. Therefore, any algorithm which finds an optimal packing for this instance does not in fact split any items. Moreover, a solution in m bins contains exactly k items per bin. Since the sum of items is exactly m, all bins in such a solution are completely occupied with respect to size.

If there exists a partition of the numbers into m sets of sum B each, then there is a partition of the adapted items into M sets of sum $\frac{1}{3k}$ each (the sum is 1 for $k=3$). Each bin is packed with $k-3$ padding items and one such triple, giving m sets of k items, each set of sum exactly 1.

If there is a packing into exactly m bins, as noted above, no items are split and each bin must contain exactly k items. If $k=3$, this implies the existence of a partition. Consider the case $k \geq 4$. We first prove that each bin contains exactly $k-3$ padding items.

If a bin contains at least $k-2$ padding items, their total size is at least $\frac{(3k-1)(k-2)}{3k(k-3)} = \frac{3k^2-7k+2}{3k^2-9k} = 1 + \frac{2k+2}{3k(k-3)}$. For $k \geq 4$ this is strictly larger than 1 and cannot fit into a bin. If there are at most $k-\ell \leq k-4$ padding items, then there are ℓ additional items of size at most $\frac{1}{6k}$ ($\ell \geq 4$). The total size is therefore at most $\frac{(3k-1)(k-\ell)}{3k(k-3)} + \frac{\ell}{6k} = \frac{6k^2-2k-5\ell k-\ell}{6k(k-3)}$. This value is maximized for the smallest value of ℓ which is $\ell = 4$. We get a size of at most $\frac{6k^2-22k-4}{6k(k-3)} = 1 - \frac{4(k+1)}{6k(k-3)}$. For $k \geq 4$ this is strictly less than 1, which as noted above does not admit a packing into m bins.

Since each bin contains exactly $k-3$ padding items, it contains exactly three adapted items, with total size exactly $\frac{1}{3k}$. The original sum of such three items is B, and we get that a solution in m bins implies a partition. □

3 Definitions and Properties

Let n denote the number of input items. We denote the size of item i by $s_i \geq 0$. If $s_i \leq 1/2$, the item is called small. If $s_i > 1$, it is large, else it is called medium.

Basic Weights We define the basic weight of item i to be $w_i = \lceil s_i \rceil / k$. Note that in any packing, there are at least $\lceil s_i \rceil$ parts of item i. Since there can be at most k parts in a bin, this means

$$\text{OPT} \geq \frac{1}{k} \sum_{i=1}^{n} \lceil s_i \rceil = \sum_{i=1}^{n} \frac{\lceil s_i \rceil}{k} = \sum_{i=1}^{n} w_i \ . \tag{1}$$

This explains our definition of the weights and generalizes the weight definition from Chung et al. [3]. We use these weights as well as other types of weights throughout the paper.

Compact Representations of Packings Since an item may have an arbitrarily large size, even if this is the only item in the input, an explicit list of the bins it is packed into may result in an output of exponential size in the input size. Thus, to avoid very large running times, we assume that when a large item is split into parts, some of which are of size 1, the bins which contain such parts of size 1 are not given explicitly in the output, but only the number of such bins is stated.

The Optimal Packing Before we begin our analysis, we make some observations regarding the packing of OPT. Any packing can be represented by a graph where the items are nodes and edges correspond (one-to-one) to bins. If there is a bin which contains (parts of) two items, there is an edge between these items. A bin with only one item corresponds to a loop on that item. One item may have multiple loops. The paper [3] showed that for any given packing, it is possible to modify the packing such that there are no cycles in the associated graph, apart from the loops. This results in the following lemma.

Lemma 3.1 *There exists an optimal packing for which the above graph representation consists of a forest, together with loops. The number of edges in each tree is exactly the number of bins in which the items are packed.*

Clearly, if a large item if a leaf, it must have some loops. Just one part of this item is combined with a part of another item in a bin. Without loss of generality, we can assume that for such an item of size $g > 1$, $\lfloor g \rfloor$ parts of size 1 each are packed in bins, and the remainder of size $g - \lfloor g \rfloor \geq 0$ is combined with the other part of item.

4 The NEXT FIT Algorithm

We can define NEXT FIT for the current problem as follows. This is a straightforward generalization of the standard NEXT FIT algorithm [8]. An item is placed (partially) in the current bin if the bin is not full *and* the bin contains less than k item parts so far. If the items fits into the bin, then it is placed there completely. If the item does not fit entirely in the current bin, the bin is filled, closed, and as many new bins are opened as necessary to contain the item.

NEXT FIT performs a constant number of operations on every bin which contains two parts of items. Since every such bin contains the first part of some item, the

number of such bins is no larger than n. Except for the very last bin, every bin which contains just one part of item, contains a part of size 1 of some item. Since only the number of such bins is specified for each item, an upper bound on the number of such bins is $\sum_{i=1}^{n} \lfloor s_i \rfloor$, and the size of output is therefore at most $n + 1 + \sum_{i=1}^{n} \log \lfloor s_i \rfloor$. The length of the input is at least $\max\{n, \sum_{i=1}^{n} \log \lfloor s_i \rfloor\}$. Therefore, the running time is linear in the input size.

Note that this is an online algorithm. The absolute approximation ratio of NEXT FIT for the classical bin packing problem is 2, as Johnson [8] showed. Surprisingly, its approximation ratio for our problem tends to this value for large k. The two problems are different, and the two results seem to be unrelated.

We show that the approximation ratio of NEXT FIT is exactly $2 - 1/k$. Thus, this simple algorithm performs as well as the algorithm from [3] for $k = 2$.

Theorem 2 *The approximation ratio of NEXT FIT is* $2 - 1/k$.

Proof We first show a lower bound. The instance contains an item of size $Mk - 1$ followed by $M(k-1)k$ items of size ε, where M is large and $\varepsilon = 1/(Mk(k-1))$. Then the first item occupies $Mk - 1$ bins, and the rest of the items are packed k per bin, in $M(k-1)$ bins. OPT has Mk bins in total. This proves a lower bound of $(M(2k-1) - 1)/(Mk)$ on the approximation ratio, which tends to $2 - 1/k$ for $M \to \infty$.

Now we show a matching upper bound. We define a *block* as a maximal set of bins which were consecutively filled by NEXT FIT (NF) in which each pair of consecutive bins contains parts of the same item. A block may contain only one bin. Denote the number of blocks by m. We assume $m \geq 1$, since the case $m = 0$ is trivial. Let $u_1, u_2, \ldots, u_m$ be the numbers of bins in the blocks $1, \ldots, m$ of NF. In each block, all bins are full except perhaps the last one, which contains k parts of items (except for block m, perhaps). This immediately implies $\text{OPT} > \text{NF} - m$ and thus

$$\text{OPT} \geq \text{NF} - m + 1. \tag{2}$$

Consider the last bin from a block $j < m$. Since NF started a new bin after this bin, it contains k parts of items. Thus it contains at least $k - 1$ items of weight $1/k$ (the last $k - 1$ items are not split by the algorithm). If $u_j = 1$, there are k such items. If $u_j > 1$, consider all items excluding the $k - 1$ last items packed into the last bin. Let s denote the total size of these items and let w denote their total weight, where $w \geq \frac{s}{k}$. Note that $s > u_j - 1$, thus $w > \frac{u_j - 1}{k}$. The value kw is an integer, thus $kw \geq u_j$. This implies the total weight of all items which are packed in a block of u_j bins is at least $u_j/k + (k-1)/k = (u_j + k - 1)/k$.

Now consider block m. If $u_m = 1$, then the weight is at least $1/k$ since there is at least one item. Else, as above the weight is at least u_m/k, since the last bin of this block has at least one item or a part of an item.

We have $\text{NF} = \sum_{j=1}^{m} u_j$. Therefore

$$\text{OPT} \geq \sum_i w_i \geq \frac{\sum_{j=1}^{m} (u_j + k - 1) - (k-1)}{k} = \frac{\text{NF} + (m-1)(k-1)}{k}. \tag{3}$$

Multiply inequality (2) by $(k-1)/k$ and add it (3) to get

$$\frac{2k-1}{k}\cdot \mathrm{OPT} \geq \mathrm{NF}\left(\frac{1}{k}+\frac{k-1}{k}\right)+(m-1)\frac{k-1}{k}-(m-1)\frac{k-1}{k}=\mathrm{NF}.$$

We conclude $\mathrm{NF} \leq (2-1/k)\mathrm{OPT}$. □

5 A $(7/5+\varepsilon)$-approximation for $k=2$

Our algorithm ALG works as explained in Fig. 1. It is easy to see that the running time of this algorithm is dominated by the sorting in step 1. In the other steps, we use only constant time per item, if its size is at most 2 and a time of $O(\log\lfloor s\rfloor)$ for an item of size $s>2$. Thus the running time is $O(S+n\log n)$, where $S=\sum_{i\,:\,s_i>1}\log\lfloor s_i\rfloor$. Note that the size of the input is $\Theta(S+n)$.

We begin by giving an example which shows that this algorithm is not an absolute $\frac{7}{5}$-approximation. For some integer N, consider the input which consists of $4N+1$ small items of size $1/(6N+4)$, $2N$ medium items of size $1-1/(12N+8)$, one medium item of size $\frac{1}{2}+1/(12N+8)$, and one large item of size $3N+1+1/(12N+8)$.

ALG packs the items of size $1-1/(12N+8)$ in $4N$ bins, together with $4N$ small items. The additional small item is combined with the medium item of size $\frac{1}{2}+1/(12N+8)$. The large item consumes $3N+2$ bins, giving a total of $7N+3$.

OPT places $3N+1$ small items together with parts of size $1-1/(6N+4)$ of the large item. The residual part of the large item has a size of $3N+1+1/(12N+8)-(3N+1)(1-1/(6N+4))=\frac{6N+3}{12N+8}$. This part can be combined with the item of size $\frac{1}{2}+1/(12N+8)=\frac{6N+5}{12N+8}$. Every additional small item is split into two identical parts, each of which is combined with an item of size $1-1/(12N+8)$. This gives a total of $5N+2$ bins. The approximation ratio is $\frac{7N+3}{5N+2}>\frac{7}{5}$.

We give another example which shows that an approximation ratio of $\frac{7}{5}$ can result from different steps of the algorithm. For some integer N, consider the input which consists of $4N$ small items of size $2/N$, $2N$ medium items of size $1-1/N$, and $3N$ medium items of size $1-2/N$.

ALG packs the items of size $1-1/N$ in $4N$ bins, together with $4N$ small items. It needs $3N(1-2/N)=3N-6$ bins for the remaining medium items. Thus it needs $7N-6$ bins in total.

OPT places $3N$ small items in separate bins (one per bin), and N small items are split into two equal parts. This gives $5N$ bins in which there is exactly enough room to place all the medium items.

Theorem 3 *The algorithm achieves an asymptotic approximation ratio of* $7/5$.

Proof Overview The analysis has two cases, depending on the step in which the algorithm halts. The easier case is the case where the algorithm halts in step 5, or earlier, at step 2 or step 4. This case is treated in Sect. 5.1, using the basic weights. For the remaining cases, items which are difficult to handle are the medium items

1. Partition the items into small, medium and large items. Sort each one of the sets of items by non-increasing size. In the cases where it is not specified which item of a given set is considered, this is the largest such available item.
2. Pack the medium items one by one, as follows, until running out of medium or small items.
   (a) If the current item fits with the smallest unpacked small item, pack them into a bin.
   (b) Else, if at least two small items are left, pack the current medium item together with the two *largest* small items in *two* bins.
3. If no small items remain unpacked, or a single small item is unpacked, pack all remaining items using NEXT FIT and halt. The items are considered in the following order. First the small item, if exists, then the medium items (sorted by non-increasing size), and then the large items.
4. Otherwise, no medium items remain. Pack all remaining small items, each in a separate bin. Pack the large items one by one into these bins using NEXT FIT (starting with the largest large item and smallest small item).
5. If any bins that have only one small item remain from the previous step, repack these small items (expect for at most one item, if the number of remaining small items is odd).
6. Pack remaining large items using NEXT FIT, starting from a large item which was packed partially in step 4, if such an item exists.

Fig. 1 The approximation algorithm for $k = 2$

which are packed in step 2(b) (*critical items*). We define adapted weights ω_i, while taking a special care of these items. We prove that the adapted weight of a bin, packed by a specific optimal solution which we consider, is at most 7. On the other hand, we prove that the average adapted weight of a bin packed by the algorithm is at least 5, except for a constant number of bins which may have a smaller adapted weight. We let c be a constant such that $\sum_{i=1}^{n} \omega_i \geq 5\text{ALG} - c$. Since $\sum_{i=1}^{n} \omega_i \leq 7\text{OPT}$, we get $\text{ALG} \leq \frac{7}{5}\text{OPT} + \frac{c}{5}$.

We prove that the constant $c = 4$ satisfies the above. For our analysis, we will assume there is a counterexample and consider a smallest counterexample I (in terms of smallest number of items). We begin by proving the following lemma.

Lemma 5.1 *In I, if step* 3 *is performed, there is at most one block which contains large items, and this block consists of one large item with no other items. If step* 6 *is performed, all bins resulting from steps* 4 *and* 6, *which contain no small items form a single block, which consists of one large item with no other items.*

Proof Once the algorithm moves to the phase where all additional bins that will be packed are empty, and will be filled using NEXT FIT on large items, all future bins will be in the same block. Consider this block of large items. Combine all items in this block into a single item. The action of the algorithm remains unchanged, whereas the optimal cost can only decrease: the previous optimal assignment is still a valid

assignment for the new instance. Since we consider an minimal counterexample, it is possible to assume the claimed property. □

5.1 A Simple Case: the Algorithm Halts in Step 2, Step 4 or Step 5

By our weight definition in Sect. 3, small and medium items have a basic weight of $1/2$. Therefore, we have the following bounds on total weight of bins packed in the different steps.

A pair of items packed in step 2(a) or step 5 has a total weight of 1. A triple of items packed in step 2(b) has a total weight of $\frac{3}{2}$, thus every packed bin has a weight of $\frac{3}{4}$. If the algorithm halts in step 5, every part of a large item is packed with a complete small item. Consider a large item which is packed in g bins, that is, together with a total of g small items. Its size is strictly larger than $\frac{g-1}{2}$ and thus its weight is at least $g/4$. Each small item has a weight of $1/2$, so we pack a weight of at least $3g/4$ in these g bins. Note that there may be one bin packed in step 5 which has only a weight of $\frac{1}{2}$. Using $\mathrm{OPT} \geq \sum_{i=1}^{n} w_i$ and $\sum_{i=1}^{n} w_i \geq \frac{3}{4}\mathrm{ALG} - \frac{1}{4}$, we get $\mathrm{ALG} \leq \frac{4}{3}\mathrm{OPT} + \frac{1}{3}$.

This is at most $\frac{7}{5}\mathrm{OPT}$ as long as $\mathrm{OPT} \geq 3$, since ALG is integer. For the cases where there are at most four items and the total size of all the items is at most 2, it is easy to verify whether a packing into two bins exists. If it does, we use that packing instead.

5.2 Critical Items

Definition 1 *A* critical *item is a medium item that the algorithm packs in step* 2(*b*).

From now on, for the analysis we use a fixed optimal packing, denoted by OPT. We assume that out of optimal solutions, we consider one which has the following property. A maximum number of bins containing two complete items, one of which is medium (and the other one is small), are packed with exactly the same contents as such bins which are packed by the algorithm.

Lemma 5.2 *In the optimal solution, a critical item is not packed as a complete item which is combined with another complete item in one bin.*

Proof Assume by contradiction that OPT combines a critical item x with a *complete* small item y. Since our algorithm starts by considering the smallest small items, and item y was apparently no longer available when x was packed, y must have been packed earlier by our algorithm, i.e. with a medium item x' which appears before x in the sorted list (x' is not critical, because for critical items we use the *largest* available small items).

We can modify OPT so that x' is packed together with y, and x takes the place of x' in OPT, that is, x is split exactly in the same proportion as x' was, an its parts take the place of the parts of x'. Since x is no larger than x', this packing is valid. However, as a result, OPT has one additional bin with a complete medium item and a complete small item, packed exactly as in the algorithm, which contradicts the choice of the optimal solution. □

Table 1 Adapted weights for complete and residual items. In the second column, s_i indicates the current (residual) size of the item. Amounts in the second column are upper bounds

Item i	Adapted weight (ω_i)	
	Complete item	Residual item
Critical item	$4s_i + 2$	$3s_i + 2$
Regular medium item	$3s_i + 2$	$3s_i + 2$
Small item	$3s_i + 2$	$4s_i + 1$
Large item	$3s_i + \lceil 2s_i \rceil$	$3s_i + \lceil 2s_i \rceil$

We define adapted weights for the items in the first column of Table 1. We are going to prove by induction that every bin in the optimal packing has a total adapted weight of at most 7. In order to build the induction, we will repeatedly remove parts of items from a tree in the optimal packing starting from a leaf, and pack them into separate bins. The parts that remain (which we will call residual items) will have adapted weights as stated in the second column of Table 1. Regarding small items, note that $4s + 1 < 3s + 2$ for any $0 < s < 1$.

Each time that we pack a bin in this process, we will pack as much as possible from the items under consideration there. This may lead to a different packing from the optimal packing that we started with. Consider for instance an input that contains two items of size 0.7 and one item of size 0.2. It can be packed in a chain of length 2, where one item of size 0.7 is split over two bins. When we start by packing the item of size 0.2, we will put an entire item of size 0.7 in the bin with it, thus splitting the chain in two pieces. We first prove the following technical lemma.

Lemma 5.3 *Consider a tree in an optimal packing. Suppose we pack this tree into bins, starting from the leaves and each time packing as much as possible into the bins that we use. Then the tree may split into pieces, but the amount of bins needed does not change as a result.*

Proof We can prove this by induction. For trees with one edge, there is nothing to show. Suppose we have a larger tree, of size x, and we pack more (possibly all) of a leaf in a bin than what happens in the optimal solution that we started with, or that we pack more (or all) of the item that the leaf is connected to in that bin. Note that these are the only two cases which can occur.

In both cases, the remaining input (that may contain a partial item) is now smaller compared to what the optimal solution packed into the remaining $x - 1$ bins, in the sense that exactly one of the items is smaller than it was before, or there is one item less. It is clear that this input can be packed into at most $x - 1$ bins. Moreover, given that we are in fact considering an optimal packing, it must be the case that the residual input still needs the same number of bins as before, i.e. exactly $x - 1$ bins. □

Lemma 5.4 *Every bin in the optimal packing has a total adapted weight of at most* 7.

Proof As stated, we use a proof by induction. Thus, we consider a set of bins which corresponds to a tree in the forest. We assume that some bins may have been removed from this packing, in a way that the packing is still a tree. The sizes of items, for which

parts of them were packed in the removed bins, have been reduced accordingly. We prove by induction on the number of edges in the tree, that is, on the number of bins in the packing, that every bin resulting from this tree has an adapted weight of at most 7. Without loss of generality, we assume that whenever the total size of the two items, or the single item which are to be packed into a bin is large enough, the total size packed into the bin has a size of 1.

Base Case Consider a given tree with a single edge. This edge could be a loop on a single node, or an edge between two nodes. Consider first a tree in the forest which consists of a single node. Since there is exactly one loop, the size of this item s satisfies $s \leq 1$. The weight of the item is no larger than $4s + 2 \leq 6$.

Next, consider a tree which consists of an edge between two items which are packed in one bin, i and i'. At least one item must be small. If the items are both small, or one of the two items is medium, but not a complete critical item, then their total adapted weight is at most $3s_i + 3s_{i'} + 4 \leq 7$. If one of the items, say item i, is a complete critical item, then by Lemma 5.2, the other item (of size $s_{i'} < 1/2$) is not complete. Thus, the total adapted weight is at most $4s_i + 2 + 4s_{i'} + 1 \leq 7$.

Induction Step We next show that in a tree which contains at least two edges, removing a bin of a leaf results in a bin for which the total adapted weight is at most 7, and the adapted weight of the remaining (parts of) items are as in Table 1, second column.

If the leaf is an item of size s with a loop, we remove a part of size $\min\{s, 1\}$ of the item and pack it in a bin. If the item is large ($s > 1$), we partition its adapted weight, which is $3s + \lceil 2s \rceil$ as follows. The part of size 1 which is packed into the bin gets a weight of 7. The remaining part of size $s - 1$ has a weight of $3s + \lceil 2s \rceil - 7 = 3(s-1) + \lceil 2(s-1) \rceil - 2$. If $s - 1 > 1$, then the residual item is large, and its adapted weight is defined properly. Otherwise, we have $\lceil 2(s-1) \rceil - 2 \leq 0$, so the new weight is at most $3(s-1)$, which is small enough for a residual item of size at most 1.

If $s \leq 1$, the packed part of size s is assigned the entire weight of the item, which is at most 6, and the remainder, which has size 0, has an adapted weight of zero.

If the leaf has no loops, it must be either small or medium. The other endpoint of its edge can be small, medium or large. Denote the size of the leaf by s and the other endpoint by s'.

If the two items are packed completely in one bin, as shown above, their total adapted weight is at most 7, and the residue of the item of size s' has both a size and an adapted weight of 0. We therefore assume $s + s' > 1$, and a part of size $1 - s$ of the item of size s' is combined with the item of size s.

If $s \leq \frac{1}{2}$, then we assign a weight of $3(1-s) + 2$ to the packed part of size $1 - s$. The total adapted weight of the packed bin is at most $3(1-s) + 2 + 3s + 2 = 7$. The residue has a size of $s + s' - 1$. If $s' \leq 1$, then $s' + s - 1 \leq \frac{1}{2}$, and the weight of the residual item is at most $4s' + 2 - 3(1-s) - 2 = 4s' + 3s - 3 = 3(s + s' - 1) + s' \leq 3(s + s' - 1) + 1$, which is less than the requirement in Table 1. If $s' > 1$, the weight of the residual item is at most $3s' + \lceil 2s' \rceil + 3s - 5 = 3(s + s' - 1) + \lceil 2(s' - 1) \rceil \leq 3(s + s' - 1) + \lceil 2(s + s' - 1) \rceil$. If $s + s' - 1 > 1$, we are done by Table 1. Otherwise, if $\frac{1}{2} < s + s' - 1 \leq 1$, then $2(s' - 1) \leq 2$, and the adapted weight is at

most $3(s+s'-1)+2$. If $s+s'-1 \le \frac{1}{2}$, then $2(s'-1) \le 1$, and the adapted weight is at most $3(s+s'-1)+1$, which is less than the bound in Table 1 for residual small items.

If $\frac{1}{2} < s \le 1$, we assign a weight of $4(1-s)+1$ to the packed part of size $1-s$. The total adapted weight of the packed bin is at most $4(1-s)+1+3s+2 = 7-s$. If $s' \le 1$, then the weight of the residual item of is at most $4s'+2-4(1-s)-1 = 4s'+4s-3 = 4(s+s'-1)+1$, which satisfies the bounds from Table 1 for residual items of size at most 1. If $s' > 1$, then $s+s'-1 > \frac{1}{2}$. The weight of the residual item is at most $3s' + \lceil 2s' \rceil + 4s - 5 \le 3(s'+s-1) + \lceil 2(s'-\frac{1}{2}) \rceil + s - 1 \le 3(s+s'-1) + \lceil 2(s+s'-1) \rceil$, using that $\frac{1}{2} \le s \le 1$. If $s+s'-1 > 1$, we are done. Otherwise, we have $\frac{1}{2} < s+s'-1 \le 1$, so $s' \le \frac{3}{2}$ and $2(s'-\frac{1}{2}) \le 2$, and the adapted weight is at most $3(s+s'-1)+2$. □

Lemma 5.5 *Our algorithm packs a weight of at least* $5\text{ALG}-4$ *into the bins that it uses.*

Proof A bin which was created in step 2(a) contains a pair of items i and i', where i is small and i' is medium, but not critical. Their total adapted weight is $(3s_i+2)+(3s_{i'}+2) \ge \frac{3}{2}+4 > 5$. A pair of bins which was created in step 2(b) contains three items i, i' and i'', where i and i' are small and i'' is a critical medium item. Note that $s_{i''}+s_i > 1$ and $s_{i''}+s_{i'} > 1$. In the two bins, we get a total adapted weight of $3s_i+3s_{i'}+4s_{i''}+6 > 6(1-s_{i''})+4s_{i''}+6 = 12-2s_{i''}$. Since $s_{i''} \le 1$, this gives a total weight of at least 10, which implies an average of at least 5 per bin.

Algorithm Halts in Step 3 Each block created in this step, except for possibly the first and last blocks (see Lemma 5.1) contains some number of medium items, which are not critical. If the block contains t bins, then the number of items in it is $t+1$. The last bin of the block must contain a complete medium item. This is true also in the case of the first block, but the first item in it may be small. Still, this small item cannot be packed in a bin with the largest medium item, so a part of the medium item is packed in the next bin. If there is no block which consists of one large item, then the last block of medium items satisfies the same properties, expect for the fact that the last bin of the block does not necessarily contain a complete medium item.

Therefore, a block with t bins, which is not last, has a total size of items of at least $t-1+\frac{1}{2}$. Therefore, the total adapted weight in all bins of this block is at least $3(t-\frac{1}{2})+2(t+1) = 5t+\frac{1}{2}$. This gives an average of at least 5 per bin.

Consider now the last block. If it contains medium items, the total adapted weight is at least $3(t-1)+2(t+1) = 5t-1$. Otherwise, a large item i of size $s_i > t-1$ is packed in those bins. This gives an adapted weight of at least $3s_i + \lceil 2s_i \rceil \ge 3(t-1)+2t-1 = 5t-4$.

Algorithm Halts in Step 6 Each block created in this step, except for last block (see Lemma 5.1) contains one large item packed in some number of bins, t, where each bin contains in addition to the part of the large item, a complete small item. Again, if the block contains t bins, then the number of items in it is $t+1$. The last block contains one large item with no small items.

Therefore, a block with t bins, which is not last, has a total size of items of at least $t-1$. Note that the adapted weight of a large item of size s is at least $3s+3$. Therefore, the total adapted weight in all bins of this block is at least $3(t-1)+2t+3=5t$. This gives an average of at least 5 per bin.

Consider now the last block. We have shown in the previous case that the adapted weight of such a block with t bins is at least $5t-4$. □

Thus, if the algorithm halts in step 3 or step 6, we get $\text{ALG} \leq \frac{7}{5}\text{OPT} + \frac{4}{5}$. This concludes the proof of Theorem 3.

The presented algorithm is very simple and elegant. However, the approximation ratio of $\frac{7}{5}$ holds only asymptotically. To design an approximation algorithm of absolute approximation ratio $\frac{7}{5}+\varepsilon$ we can act as follows. If the number of input items is at least $\frac{2}{\varepsilon}$, then $\text{OPT} \geq \frac{1}{\varepsilon}$ and we get $\frac{7}{5}\text{OPT} + \frac{4}{5} < (1+\varepsilon)\text{OPT}$. Otherwise, using the techniques from [6], only constant time is required for enumerating all packings, so an optimal solution can be found. In particular, to achieve an absolute approximation ratio of $\frac{10}{7}$ it is sufficient to find an optimal packing for the case of at most 54 items that have total size at most 27. If $\text{OPT} > 27$, then $\text{ALG} \leq \frac{10}{7}\text{OPT}$ because ALG is integer.

6 Conclusions

In this paper, we gave the first absolute upper bounds for $k \geq 3$ for this problem. Furthermore we provided an efficient algorithm for $k=2$.

An interesting question is whether it is possible to give an efficient algorithm with a better approximation ratio for $k=2$ or for larger k. In a companion paper [6] we present polynomial time approximation schemes (PTAS) for these problems. Note that using an approximation scheme as a $(1+\delta)$-approximation for moderate values of δ leads to large running times. The PTAS applies rounding on the items, and them enumerates packing of rounded items using patterns. Using the fact (shown in [3]), that an optimal packing can be represented by a forest with loops, a pattern is defined as a tree with at most $\frac{1}{\delta^2}$ nodes. Each node in the tree can take $\frac{1}{\delta^2}$ distinct sizes, which are rounded sizes of items. Items are not only rounded but also split into sizes of at most $\frac{1}{\delta}$ (before the rounding). For instance, for $\delta = 1/2$, the PTAS from [6] assumes that there are four rounded sizes of items, all sizes are no larger than 2, and all trees have at most 4 nodes. A tree is called a pattern, if it is maximal in the sense that splitting it into two trees would increase the number of bins. Assume that the sizes of items are 1.2501, 1.2502, 1.2503 and 1.2504. We focus on trees which are paths. It can be verified that every path on nodes of these sizes is a maximal tree. If the total number of items is large enough, then the number of such paths is 128. If we consider only paths which contain one item of each size, the number of possible paths is 12. Even if the packing contains only such trees, still a packing must contain $\frac{n}{4}$ trees. Such a packing can be expressed by a vector of length 12, where the sum of the components is $\frac{n}{4}$, and each component is a non-negative integer. The number of such vectors is $\Omega(n^{11})$, which leads to a time complexity of $\Omega(n^{11})$, since the PTAS enumerates all possible combinations of patterns. Comparing this to the running time

of our algorithm, which is $O(S + n \log n)$, we can see that the PTAS is inefficient. Another clear advantage of our algorithm is its simplicity.

References

1. Babel, L., Chen, B., Kellerer, H., Kotov, V.: Algorithms for on-line bin-packing problems with cardinality constraints. Discrete Appl. Math. **143**(1–3), 238–251 (2004)
2. Caprara, A., Kellerer, H., Pferschy, U.: Approximation schemes for ordered vector packing problems. Nav. Res. Logist. **92**, 58–69 (2003)
3. Chung, F., Graham, R., Mao, J., Varghese, G.: Parallelism versus memory allocation in pipelined router forwarding engines. Theory Comput. Syst. **39**(6), 829–849 (2006)
4. Epstein, L.: Online bin packing with cardinality constraints. SIAM J. Discrete Math. **20**(4), 1015–1030 (2006)
5. Epstein, L., Levin, A.: AFPTAS results for common variants of bin packing: A new method to handle the small items. Manuscript (2007)
6. Epstein, L., van Stee, R.: Approximation schemes for packing splittable items with cardinality constraints. In: Fifth Workshop on Approximation and Online Algorithms (WAOA 2007). Lecture Notes in Computer Science, vol. 4927, pp. 232–245. Springer, Berlin (2008)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
8. Johnson, D.S.: Fast algorithms for bin packing. J. Comput. Syst. Sci. **8**(3), 272–314 (1974)
9. Kellerer, H., Pferschy, U.: Cardinality constrained bin-packing problems. Ann. Oper. Res. **92**, 335–348 (1999)
10. Krause, K.L., Shen, V.Y., Schwetman, H.D.: Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. J. ACM **22**(4), 522–550 (1975)
11. Krause, K.L., Shen, V.Y., Schwetman, H.D.: Errata: "Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems". J. ACM **24**(3), 527 (1977)
12. Mao, J., Graham, R.L.: Parallel resource allocation of splittable items with cardinality constraints. Manuscript
13. Shachnai, H., Yehezkely, O.: Fast asymptotic FPTAS for packing fragmentable items with costs. In: Proc. of the 16th International Symposium on Fundamentals of Computation Theory (FCT2007), pp. 482–493 (2007)
14. Shachnai, H., Tamir, T., Yehezkely, O.: Approximation schemes for packing with item fragmentation. Theory Comput. Syst. **43**(1), 81–98 (2008)
15. van Vliet, A.: An improved lower bound for online bin packing algorithms. Inf. Process. Lett. **43**(5), 277–284 (1992)
16. Yao, A.C.C.: New algorithms for bin packing. J. ACM **27**, 207–227 (1980)