

Approximation schemes for packing splittable items with cardinality constraints

Leah Epstein*

Rob van Stee†

November 19, 2007

Abstract

We continue the study of bin packing with splittable items and cardinality constraints. In this problem, a set of items must be packed into as few bins as possible. Items may be split, but each bin may contain at most k (parts of) items, where k is some fixed constant. Complicating the problem further is the fact that items may be larger than 1, which is the size of a bin. We close this problem by providing a polynomial-time approximation scheme for it. We first present a scheme for the case $k = 2$ and then for the general case of constant k .

Additionally, we present *dual* approximation schemes for $k = 2$ and constant k . Thus we show that for any $\varepsilon > 0$, it is possible to pack the items into the optimal number of bins in polynomial time, if the algorithm may use bins of size $1 + \varepsilon$.

1 Introduction

In bin packing problems, a set of *items* is given and the goal is to pack them into the smallest possible number of containers, called *bins*. The items are typically given as numbers between 0 and 1, which is the bin size. In this paper we consider items that may be larger than 1. Items are allowed to be *split* and distributed among an arbitrary number of bins.

Clearly, if we allow items to be split and have no other constraints, a simple Next Fit-type algorithm can generate an optimal solution. However, we require that at most k (parts of) different items are packed together in a single bin. This is called a *cardinality constraint*, and it makes the problem NP-hard in the strong sense for any fixed $k \geq 2$ [3, 6].

This problem was introduced by Chung et al. [3], who discussed the problem of allocating memory to parallel processors. The goal is that each processor has sufficient memory and not too much memory is being wasted. 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 is to allow memory sharing between processors. However, if there is a single shared memory for all the processors, there will be a lot of 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] suggested a new architecture where each memory may be accessed by at most *two* processors, avoiding the disadvantages of the two extreme earlier models. This leads to the bin packing problem described above, where in their paper $k = 2$: the bins are the memories and the items to be packed represent the memory requirements of the processors. The problem was further studied in [6, 8]. We describe the results of these papers below.

*Department of Mathematics, University of Haifa, 31905 Haifa, Israel. lea@math.haifa.ac.il.

†Department of Computer Science, University of Karlsruhe, D-76128 Karlsruhe, Germany. vanstee@ira.uka.de. Research supported by the Alexander von Humboldt Foundation and the German Research Society (DFG).

In this paper, 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} , $\mathcal{A}(\mathcal{I}) \leq \mathcal{R} \cdot \text{OPT}(\mathcal{I})$, 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* for an algorithm \mathcal{A} is defined to be

$$\mathcal{R}_{\mathcal{A}}^{\infty} = \limsup_{n \rightarrow \infty} \sup_{\mathcal{I}} \left\{ \frac{\mathcal{A}(\mathcal{I})}{\text{OPT}(\mathcal{I})} \mid \text{OPT}(\mathcal{I}) = n \right\} .$$

This ratio is relevant if we are particularly interested in the performance of algorithms on large inputs, that cannot be packed in few bins. Fernandez de la Vega and Lueker [4] designed an APTAS for standard bin packing. Their work was followed by the work of Karmarkar and Karp [10] who developed an AFPTAS.

Regarding the absolute approximation ratio, for the classical bin packing problem a simple reduction from the PARTITION problem (see problem SP12 in [7]) shows that no polynomial-time algorithm has an absolute performance guarantee better than $\frac{3}{2}$ unless $\text{P}=\text{NP}$. This reduction is no longer valid for our problem, where items may be split.

Chung et al. [3] showed that the problem with splittable items is NP-hard in the strong sense for $k = 2$. They use a reduction from the 3-PARTITION problem (see problem [SP15] in [7]). In a recent paper [6], we showed that this problem is NP-hard in the strong sense for any fixed value of k .

Chung et al. [3] also gave a $3/2$ -approximation for the case $k = 2$. Graham and Mao [8] analyzed the asymptotic approximation ratio of several algorithms, giving upper bounds of 1.498 for $k = 2$, $3/2$ for $k = 3$ and $2 - 2/k$ for $k \geq 4$. In [6], we gave a simple algorithm with an absolute approximation ratio of $2 - 1/k$ for $k \geq 2$, and an algorithm with absolute approximation ratio of $7/5$ for $k = 2$.

Bin packing with cardinality constraints (and regular, non-splittable items) was introduced and studied in an offline environment as early as in 1975 by Krause, Shen and Schwetman [12, 13]. 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. Kellerer and Pferschy [11] designed an improved offline approximation algorithm with performance guarantee 1.5 and finally a PTAS was designed in [2] (for a more general problem).

On the other hand, 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 $k = 2, 3$. Finally, Epstein [5] 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 $3 \leq k \leq 6$.

A related problem was studied recently by Shachnai, Tamir and Yehezkeley [14]. They considered an offline bin packing problem where items may be split arbitrarily. They consider two models: one where splitting items comes at a cost, as each part of a split item increases by a constant additive factor, and one where there is an upper bound on the total number of splits. They showed that both these problems do not admit a PTAS unless $\text{P} = \text{NP}$. They designed approximation schemes for both problems. Their problem is different from our problem since in their case all items have size 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 Our first main result is a polynomial-time approximation scheme. Recall that for standard bin packing, this is impossible unless $\text{P} = \text{NP}$. We first present our scheme for the special case of $k = 2$

and then show how to extend it to the general case. The main difficulty here is that we have less structure in the packing, making it harder to search all potential packings.

We also present a dual PTAS for this problem, first for $k = 2$ and then for general k . That is, given bins of size $1 + \varepsilon$ for an arbitrary $\varepsilon > 0$, we give an algorithm to pack these items into at most N bins, where N is the number of bins (of size 1) in an optimal solution. The difficulty of designing such a dual PTAS lies in the packing of large items. Since they can be arbitrarily large, the number of items does not imply any upper bounds on the optimal cost, and no known rounding techniques apply in this case.

Note that a dual PTAS for standard bin packing is a component in the PTAS for scheduling on identical machines, which was given by Hochbaum and Shmoys [9].

2 PTAS for $k = 2$

2.1 The structure of the optimal packing

Before we begin our analysis, we make some observations regarding the packing of OPT . A 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. 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. Thus the graph consists of a forest together with some loops.

We now consider items that are larger than $1/\varepsilon$. We modify the input as follows. Any item of size $x > 1/\varepsilon$ is replaced by $\lfloor \varepsilon x \rfloor$ items of size $1/\varepsilon$ and one additional item of size $x - \frac{1}{\varepsilon} \lfloor \varepsilon x \rfloor$ (if this last amount is nonzero). Denote the original input by I and the modified input by I' . We have the following lemma.

Lemma 1 $\text{OPT}(I') \leq (1 + \varepsilon)\text{OPT}(I)$.

Proof Consider an optimal packing for the input I . We show how to modify this packing to pack the input I' , opening at most $\varepsilon\text{OPT}(I)$ extra bins in the process. This proves the lemma.

For some item x of size more than $1/\varepsilon$, consider the bins in which it is packed in some order, and keep track of the total size of x packed so far. Let us number these bins $1, 2, \dots$. When we reach a bin j where the total size packed reaches $1/\varepsilon$, we cut the part of x in bin j into two parts, and put one of the parts in a new, empty bin, in such a way that the total size of x that is packed in bins $1, \dots, j$ is exactly $1/\varepsilon$. (If the 'part' placed in the new bin has size 0, do not open a new bin.)

We then continue starting from bin $j + 1$ in the same way. We reset the size counter to the size of the part of x that is packed into the new bin that we just opened, and consider the bins in sequence until the size counter again reaches $1/\varepsilon$, etc. This takes at least $1/\varepsilon$ bins each time. Thus we need one extra bin per entire multiple of $1/\varepsilon$. \square

This lemma implies that for an input with items larger than $1/\varepsilon$, we can begin by splitting these items into pieces of size $1/\varepsilon$. Then if we find a solution which approximates $\text{OPT}(I')$, this solution approximates $\text{OPT}(I)$ nearly as well.

The optimal packing for the modified input I' consists of a forest and some loops. The trees can be arbitrarily large, where the size of a tree is the number of its nodes. However, given an optimal solution with large trees (possibly with loops), we can split these trees into trees (with loops) of constant size. Denote by $\text{OPT}'(I')$ an optimal solution for the case where there is an additional constraint that all trees that are created in the packing must have size at most $1/\varepsilon^2$. We then have the following lemma.

Lemma 2 $\text{OPT}'(I') \leq (1 + 2\varepsilon)\text{OPT}(I')$.

Proof Let us assume the packing for $\text{OPT}(I')$ contains a tree of size $S > 1/\varepsilon^2$. We can find a centroid: a node which can be taken as root such that no subtree has size more than $S/2$. We repeatedly remove a centroid from large trees, along with all its outgoing edges and loops. That is, we pack the item under concern in new, empty bins.

Specifically, we do the following. Consider a tree of size S . For $i = \log\lfloor S\varepsilon^2 \rfloor, \dots, 1$, for every tree of size larger than $2^i/\varepsilon^2$ that is created in this process (or for the original tree), find a centroid and remove it. For a given i , the number of centroids is at most $S\varepsilon^2/2^i$. Moreover, because we modified the input, no item has size more than $1/\varepsilon$. This means that for each centroid, we need at most $1/\varepsilon$ additional bins to pack it into empty bins. The total number of bins required to pack all the centroids is then at most $2S\varepsilon$ for an original tree of size $S > 1/\varepsilon^2$. In other words, the number of additional bins to repack all the trees into trees of size at most $1/\varepsilon^2$ is at most $2\varepsilon\text{OPT}(I')$. This proves the lemma. \square

2.2 Description of the PTAS

We look for solutions with trees of size at most $1/\varepsilon^2$. We start by using techniques introduced by Lueker and Fernandez de la Vega [4]. Let n be the number of items in I' . Assume first that $n \geq 1/\varepsilon^2$. The easier case $n < 1/\varepsilon^2$ is treated below. We sort the items in order of nonincreasing size and put the items into groups of $\lceil n\varepsilon^2 \rceil$ successive items (possibly less items for the last group). Say that this gives $p + 1$ groups.

We now modify I' as follows. We remove the first group (the one with the largest items). For each other group, we round the item sizes inside this group up to the size of the largest item in the group. This creates an input I'' which does not require more bins to be packed than I' (since we can map every item in I'' to an item of I' that is no smaller than it), and does not require less bins than I' without the first group (since we rounded up the item sizes).

We are going to consider all possible packings for I'' , that is, all possible forests with trees of size at most $1/\varepsilon^2$.

From a packing of I'' to a packing of I' Given a packing (represented by a forest), we are going to change it as follows: wherever an item of group i is needed, we are going to take an arbitrary unpacked item from this group. Since it is smaller than the rounded version, it definitely fits in the space that is allocated to it. This leaves the $\lceil n\varepsilon^2 \rceil$ largest items of I' (the items in group 1) unpacked and we pack them into separate bins (into chains where possible). The size of each such item (in I') is at most $1/\varepsilon$ so this requires at most $(n\varepsilon^2 + 1)/\varepsilon \leq 2n\varepsilon$ extra bins (using that $n \geq 1/\varepsilon^2$).

The packing that we will finally use is the one that uses the least amount of bins *and* that gives a feasible packing. The important thing to note is that we are going to try all possible forests that pack all the items, thus we also try the one that corresponds to the packing of $\text{OPT}'(I')$. Our modified packing then requires only $2n\varepsilon$ more bins than $\text{OPT}'(I')$.

From a tree to a packing of I'' An important ingredient of our algorithm is still missing. We need to actually allocate items to bins based on the tree representation.

To begin with, we may assume that the trees in $\text{OPT}'(I')$ are *minimal* in the sense that any partition of the items in a tree into two sets requires more bins to pack the items than the original tree.

Second, note that in any tree, items can be packed starting from the leaves without wasting any space, so we do not have any empty space in bins apart from possibly one bin that contains part of the root. This also means that given a tree, the number of bins required to pack this tree follows immediately from the total size of the items packed.

Third, to pack the items into bins we do not need the loops explicitly: we can ignore them, and it will be clear from the size of an item whether or not we need to pack some bins that contain only a part of this item. Thus from now on we work with real trees.

1. Any item of size $x > 1/\varepsilon$ is replaced by $\lfloor \varepsilon x \rfloor$ items of size $1/\varepsilon$ and one additional item of size $x - \frac{1}{\varepsilon} \lfloor \varepsilon x \rfloor$ (if this last amount is nonzero).
2. Put the items into a constant number of groups and round them as in Lueker and Fernandez de la Vega [4]. All items in a group have the same size. The largest group is packed separately in new bins.
3. Determine the set of valid patterns (trees plus specification of groups of nodes).
4. Determine the forest (combination of patterns) which uses the least number of bins and which packs all the items.
5. Replace the rounded items by the original items.

Figure 1: The PTAS for $k = 2$ and $n \geq 1/\varepsilon^2$

We define a type of a tree to be a pair (j, E) where j is the number of vertices ($1 \leq j \leq 1/\varepsilon^2 + 1$). We assume that these vertices are always numbered $1, \dots, j$ and E is a subset of $j - 1$ edges. A *pattern* consists of two parts. The first one is a type of a tree (defined above), and the second is a vector of length j , where component i (for $1 \leq i \leq j$) is the group to which node i belongs (a number between 1 and $p + 1$). The number of patterns is constant, since there is a constant amount of trees with at most $1/\varepsilon^2 + 1$ nodes, and $p + 1 \leq 1/\varepsilon^2$ possible groups for each node. For a given pattern, items are packed starting from the leaves. During the process, we sometimes *assign* items to bins without immediately packing them into those bins. A pattern is *valid* if this process leads to all items being packed without violating any cardinality constraints and the representation of the final packing is the original tree.

Packing a leaf is done as follows: first fill up the bins it is assigned to, if any. Then open as many new bins as you need to pack this item. If the final bin is not filled completely, assign the item at the other side of the edge leading to this leaf to that bin. (If this violates the cardinality constraint of k , or if the final bin is not used at all, the pattern is not valid.) Remove the leaf and the edge that leads to it (possibly creating a new leaf).

Note that in this process, items may be assigned to multiple bins before finally being packed into them. When a leaf completely fits into bins it was already assigned to, and it is not the last node packed in the tree, we have in effect found a smaller tree and we know that this pattern is not valid since the tree was not minimal. This proves the following lemma.

Lemma 3 *Given a tree representation of the optimal packing with minimal trees, it is possible to assign the items to bins such that for each tree, there is at most one bin which is not completely full.*

Enumerating the forests There is a constant number of patterns, so certainly a constant number of valid patterns. Each valid pattern can be picked at most n times, thus we have at most $(n + 1)^{O(1)}$ assignments to check. Once we know how many instances of each pattern you have, we can check whether this gives a valid packing (the right number of items of each group), and finally select the one that uses the least number of bins. This can be done in polynomial time.

Our PTAS (for the case $n \geq 1/\varepsilon^2$) is summarized in Figure 1.

A trivial lower bound on the amount of bins needed to pack the entire input is $n/2$. Putting it all together, we find that our algorithm uses the following number of bins:

$$\text{OPT}'(I') + 2n\varepsilon \leq (1 + 4\varepsilon)\text{OPT}'(I') \leq (1 + 10\varepsilon)\text{OPT}(I') \leq (1 + 16\varepsilon)\text{OPT}(I),$$

where we have applied $\text{OPT}'(I') \geq n/2$, Lemma 2, and Lemma 1 in this order, as well as $\varepsilon \leq 1/2$.

Finally we consider the case where $n < 1/\varepsilon^2$. For a constant number of items, there exists only a constant number of forests including the allocation of items to nodes. A given forest can be filled up as described above (starting from the leaves), using the exact sizes of items. Hence in this case we do not split large items or large trees and find the optimal solution in polynomial time.

Altogether, this proves the following theorem.

Theorem 1 *There exists a polynomial-time approximation scheme for cardinality constrained bin packing of splittable items where each bin is allowed to have at most two items or parts of items.*

3 PTAS for constant k

Our PTAS for constant k will be essentially the same as the one in Figure 1 for $k = 2$. However, we need to implement Step 3 for this more general case. For this we use a modified graph representation. If a bin contains parts x_1, \dots, x_k , we order them in some way and create edges only between successive items in this ordering. Thus the edges in this case are e.g. $(x_1, x_2), \dots, (x_{k-1}, x_k)$. Each item is still represented by a single node, thus one node might be involved in several of such chains, where each chain represents one bin.

So in this case, we no longer have a one-to-one correspondence of edges and bins. Instead, there are now at least as many edges as there are bins. Note that the order of the parts inside a bin is irrelevant. Thus we can reorder the parts in a chain arbitrarily. This gives a different graph for the same packing into bins. It is now more difficult to construct a packing into bins from a given graph. We have the following two important lemmas.

Lemma 4 *If the graph of a packing contains a cycle, it is possible to modify the packing such that this cycle is removed without increasing the number of bins.*

Proof For a given packing, consider a cycle in the associated graph. If this cycle passes through the same bin more than once, we reorder the parts within that bin such that all items involved in the cycle are in a chain. Having done this for all bins, we now focus on cycles that do not have chords.

If two or more successive edges in this cycle connect items that all have a part in the same bin, we reorder the parts inside this bin as follows. Suppose x_i, \dots, x_j all have a part in one bin, but x_{i-1} and x_{j+1} do not have a part in that bin. We put the parts in this bin in the order $x_i, x_j, x_{i+1}, \dots, x_{j-1}$. In this way, the parts x_{i+1}, \dots, x_{j-1} are no longer a part of this cycle.

By doing this repeatedly, we can ensure that each edge within a cycle corresponds to a (unique) bin. At this point we can apply Lemma 1 from [3] to remove this cycle by repacking those bins. \square

Lemma 5 *There exists an optimal packing such that each item of size at most i/k is split into at most i parts, for all $i > 1$.*

Proof By Lemma 4, the optimal packing can be assumed to form a forest. For a given tree, we root it at some item. We will consider the items that are in too many bins level by level, starting from the root. Thus, consider an item x of size at most i/k that is in $i' > i$ bins. We ignore the bin that contains the uplevel item (if any) and sort the other $i' - 1 \geq i$ bins by increasing size of the part of x .

We are going to repack the bins without using more bins. Consider the two bins with the two smallest parts of x , m_1 and m_2 (where we have ignored the bin with the uplevel item). These parts have total size at most $2/k$. If there exists an item part y in the first bin (not a part of x) which has size at least m_2 , we can split it into a part of size m_2 and another part, and switch the part of size m_2 with m_2 in bin 2. This reduces the number of bins that contain x and gives a valid packing (no more than k parts in any bin). The item of which y is a part will be treated later if necessary, since it is not uplevel.

Suppose that all items in bin 1 are smaller than m_2 . By the size constraint on x and our sorting, we have

$$m_1 + (i' - 2)m_2 \leq i/k$$

which implies

$$m_2 \leq \frac{i}{i' - 2} \frac{1}{k} - \frac{m_1}{i' - 2}$$

and therefore

$$1 - m_2 \geq 1 + \frac{m_1}{i' - 2} - \frac{i}{i' - 2} \frac{1}{k}.$$

We want to know whether there is an item of size at least m_1 in bin 2 (apart from the part m_2 of x). We may assume that bin 2 is full (if not, we can add dummy size to some item and move this as well when we decide to switch it with a part in bin 1). That means we need to check whether $(1 - m_2)/(k - 1) \geq m_1$ (if there are less than k items in bin 2, the condition becomes less strict). This holds if

$$1 + \frac{m_1}{i' - 2} - \frac{i}{i' - 2} \frac{1}{k} \geq (k - 1)m_1$$

which for $i > 1$ and $k > 2$ is equivalent to

$$m_1 \leq \frac{k(i' - 2) - i}{k(k - 2)(i' - 2) - k}.$$

However, it is easy to check that $\frac{k(i' - 2) - i}{k(k - 2)(i' - 2) - k} \geq 1/k$ for $i' > i > 1$ and we know that $1/k \geq m_1$. Thus in this case, we can move m_1 to bin 2, and split the largest item in bin 2 and move a part of size m_1 to bin 1. We can repeat this process as long as x is in more than i bins. \square

From a tree to a packing of rounded items Lemmas 1 and 2 also hold for general k . However it is now not so clear how to construct the bins from a given forest.

To do this, we further modify our graph representation, and let each item be represented by x nodes if and only if it is split into x parts in the packing. The parts of one item are connected by a simple chain, as are the parts that are in one bin. We can then start packing bins from the leaves of the tree and repeatedly remove leaves similar to before. There are now two cases, since the edge that connects the leaf to the tree leads either to a copy of the leaf (same item) or to a different item.

If a leaf leads to another item, then the remaining unpacked part of the leaf item must be small enough that it can be packed entirely inside the bin it is assigned to, so do that. (If not, there would be an edge leading to a copy of the leaf.) If it does not fit, we know that the tree is not valid. Also, assign the item at the other end of this edge to this bin. If there are already k items in the bin, the tree is not valid.

If a leaf leads to another part of the same item, then the bin this leaf is assigned to (or a new bin, if it is not assigned to anything) can be filled up by this leaf. This holds because no future unpacked items can be assigned to this bin (otherwise there would be an edge to that item from this leaf).

Using this packing process, it can be seen that Lemma 3 also holds for this case. In order to apply this process, we do not only need to know the group to which each node belongs but also *which* of the items of that size is packed there. Again, let the type of a tree be a pair (j, E) where j is the number of nodes in the tree (as mentioned above, there is one node in the tree for every part of an item) and E is a set of $j - 1$ edges.

We now need a vector (a, b) for each node (to get a pattern for the tree). Thus, a is the group ($a \in \{1, \dots, p + 1\}$, where $p + 1$ is the number of groups like in Section 2.2) and b is the number of the item of this group ($b \in \{1, \dots, 1/\varepsilon^2\}$, as there are at most $1/\varepsilon^2$ items in a tree, there are certainly at most $1/\varepsilon^2$ items of any one of the groups). In a valid tree, the nodes of type (a, b) for any fixed a and b must be in a chain, since they represent parts of one item. The maximum length of such a chain is bounded by the following Lemma.

Lemma 6 *The length of a chain representing one item in a valid pattern is bounded by $1/\varepsilon^2 + 1/\varepsilon$.*

Proof There can be at most $1/\varepsilon^2$ nodes in the chain that have an edge to another item, because otherwise there would be two nodes having edges to the same item, giving a cycle.

There can be at most $1/\varepsilon$ nodes in the chain that do not have an edge to another item, since each such node has a bin to itself and such a bin (apart from at most one) will be fully packed in an optimal solution by Lemma 3. The size of an item is at most $1/\varepsilon$. \square

We can now determine in polynomial time how often each pattern is used in an optimal packing, as in the previous PTAS. We now have that a trivial lower bound on the cost of packing n items is n/k . Thus our PTAS only works for constant k , and requires at most $(1 + (2k + 4)\varepsilon)\text{OPT}(I)$ bins.

Again, for a constant number of items, only a constant amount of forests needs to be checked, and we can find an optimal solution. Thus we have the following theorem.

Theorem 2 *For any constant $k \geq 2$, there exists a polynomial-time approximation scheme for cardinality constrained bin packing of splittable items where each bin is allowed to have at most k items or parts of items.*

4 A dual PTAS for $k = 2$

We have already seen that the optimal packing can be represented by a forest together with some loops. Moreover, in each tree, the only items that have degree more than two have size more than 1. Items of size in $(\frac{1}{2}, 1]$ have at most two neighbors. We call such items *medium*. Items of size in $(0, \frac{1}{2}]$ have at most one neighbor. We call such items *small*.

Our algorithm tries to find a good way to cut items, i.e., split them into parts. The cuts are performed in two stages. As a first step we cut a single piece off medium and large items. Our algorithm performs an enumeration on such possible cuts. Clearly, these are not the only cuts that an optimal algorithm may perform on these items for its packing. However, by Lemmas 7 and 8, proved in [6], no further cuts are required for items of size at most 1.

Lemma 7 *There exists an optimal packing in which all items of size at most $1/2$ are leaves.*

Lemma 8 *There exists an optimal packing in which any item of size in $((i - 1)/2, i/2]$ has at most i neighbors for all $i \geq 2$.*

When we perform cuts on items, our algorithm considers the two resulting parts to be two independent items and thus allows to cut them further (for parts that have size more than 1) while creating a packing. The enumeration considers a set of cut options which cover sufficiently many packings to find a very good one. The options include the “empty cut”, i.e. the case that this item is not cut at all.

We do this initial cutting in order to simplify the tree structure. We would like to work with trees that contain at most one large item, and each tree is a star rooted at a large item or a part of a large item. We now show that by cutting off a piece of size at most 1 from each item that is medium or large, and treating this piece as an independent item, we get a packing which has this property without increasing the number of bins required to pack the input. Note that these techniques are useful only for the dual PTAS and not for the PTAS since the modification of the input is done by cutting some items. We later use the fact that we can slightly increase the sizes of bins in order to efficiently enumerate the possible cutting points.

Lemma 9 *It is possible to modify the input in such a way that the optimal packing for the new input requires the same number of bins as the old input, and there exists an optimal packing for the new input such that all medium items have degree 1.*

Proof Consider the optimal packing P for the original input. For each medium item in this packing, create one or two new items with sizes depending on where (and whether) this item is cut into parts. Two parts are sufficient by Lemma 8. An optimal packing for the modified input requires the same number of bins. First, we can use the packing P , so we do not need more bins. Second, if there were a better packing for this set, it could also have been used for the original instance. This holds because our modification restricts the options for packing the items.

Note that a part of a medium item might still be medium in an optimal packing. So it appears that there might now be more options, because this new medium item might be cut again to pack the items. However, by Lemma 8 we know that for the original input there is an optimal packing that cuts each medium item at most once. So cutting a medium item more than once cannot help to find a better packing.

In the packing P for the modified input, each newly created item is a leaf. □

We first show that there exists some set of cuts where each medium or large item is cut at most once, that converts an optimal solution which is a set of trees into a set of stars. Later we show how we can restrict ourselves to a small set of possible cuts which results in the need of slightly larger bins.

This works as follows. Given a medium item, since it has at most two neighbors, cutting one piece off the item and considering the two parts to be independent items separates the tree into two trees where the degree of the two parts of this medium item is 1. Note that if the degree of the item is at most 1 before the cut, then an empty cut is applied.

Lemma 10 *It is possible to modify the input in such a way that the optimal packing for the new input requires the same number of bins as the old input, and there exists an optimal packing for the new input such that each tree contains at most one large item.*

Proof Given a tree with more than one large item x , rooted at an arbitrary node, consider a large item of maximum distance from the root. Compute the part of this item that should be combined with each of its children. The sum of these parts is the part that should be cut off in order to split the node of the large item into two nodes, one that is the root of a new tree that has no large item besides x and the other one is a leaf of the old tree (which is now smaller). The size of the first piece follows from Lemma 3. To be precise, we choose the size to be the maximum value s such that an item of size s can be packed together with the subtree rooted at x without leaving any empty space in any bin, and s is smaller than the size of x (if it is equal, the tree is not minimal). Some of these bins may be filled with a part of x of size 1.

The second part of the item has the remaining size (less than 1!) which should be combined in a bin with the item of the uplevel edge. Repeat this process until there is only one (entire) large item in the tree. Since each such process for one large item results in a new tree where one part of the item is its root, and a leaf in the original tree, each large item is cut at most once. □

We conclude that by modifying the input appropriately, there exists an optimal packing which consists of stars with large items in the middle (where such a large item that is a root of a star might be smaller by at most 1 than the corresponding large item in the original input), single edges, and loops. We will look for a packing that has this structure. Denote the number of input items by n .

4.1 Description of the algorithm

Our dual PTAS works as follows. We use a parameter δ which is based on ε , and which is the inverse of some odd integer. Specifically, we let $K = \min\{i \mid i \geq 2/\varepsilon, 2 \nmid i\}$ and $\delta = 1/K$. We begin by rounding item sizes (of all items that are not large) up to the nearest multiple of δ (possibly to 0). There are $K + 1$ possible sizes of such items. For a given tree, we can fill the bins starting with these items. This means that each cut of an item will now occur at an integer multiple of δ . This also holds for a tree

1. Let $K = \min\{i \mid i \geq 2/\varepsilon, 2 \nmid i\}$ and $\delta = 1/K$.
2. Round each item size which is no larger than 1 up to the nearest multiple of δ . Let the number of items of size $i\delta$ be M_i for $i = (K + 1)/2, \dots, K$.
3. For each medium item size, *guess* how many items of this size are cut at $j\delta$ for $j = 0, \dots, (K - 1)/2$.
4. *Guess* how many items of size $j\delta$ are cut off from large items for $j = 0, \dots, K$.
5. Create a graph with L layers, plus source and sink. The construction of the graph is shown in Figure 3. This graph represents all possible packings for the current set of guesses. Find a path with minimal cost from the source to the sink. This is the cost of packing the input with these guesses.
6. Use the packing of this guess to create a packing for the original instance.

Figure 2: The dual PTAS for $k = 2$

that contains no small items (items of size at most $1/2$) but does contain medium items. By the above, if a tree contains no items of size at most 1, it consists of only a loop (a single item).

Denote the number of items of size $i\delta$ by M_i for $i = (K + 1)/2, \dots, K$. For each size, we guess how many items of this size are cut at each integer multiple of δ that is at most $1/2$. Note that we do not need to consider cuts above $1/2$, since cutting an item of size $i\delta$ at the point $j\delta$ or at the point $(i - j)\delta$ gives the same parts. Thus the possible cutting points are $i\delta$ for $i = 0, \dots, (K - 1)/2$.

Our dual PTAS is summarized in Figure 2. Each guessing step can be emulated via an exhaustive enumeration of all the possibilities for this piece of information. So our algorithm runs all the possibilities, and among them chooses the best solution achieved. Denote the number of large items by L . For convenience of notation, we will also denote this number by $M_{(K-1)/2}$. We guess how many pieces of each size of at most 1 that is an integer multiple of δ are cut off. Note that a large item may stop being large when some part of it is cut off. However, in our algorithm, we still group it among the large items (and in particular, allow it to be cut further). The cuts can be represented by a vector of size $(K + 1)^2/4 + (K + 1)$, which tells us how many items of each size $(K + 1)\delta/2, \dots, K\delta$ are cut off at each point, and how many pieces of each size are cut off from the large items.

Construction of the graph For every possible set of cuts, we do the following. We construct a layered graph which represents possible packings. The graph starts at a single source node, then there are L layers which correspond to the L large items, and finally there is a sink. We maintain a *summary vector* which describes how many **unpacked** (parts of) items there are of every size $i\delta$ ($i = 0, \dots, K$). This vector is denoted by $s(u)$ for a node u . Additionally, we maintain a *cutoff vector* which contains unpacked parts of size less than 1 **of large items**. This vector is denoted by $c(u)$ for a node u . We concatenate both vectors into a single *packing vector* of length $2(K + 1)$ which contains all relevant information needed to find the optimal packing for these parts. Note that the parts which were cut off large items are listed twice, once in the main list of unpacked items, so that they can be packed, and once in the list of parts of large items, to make sure that the pieces that were cut off are matched to the large items.

For two nonnegative integer vectors a and b of length ℓ , we say that $a \geq b$ if $a_i \geq b_i$ for $i = 1, \dots, \ell$. We say that $a \rightarrow b$ if there exists a unique j such that $a_j = b_j + 1$ and $a_i = b_i$ for $i \in \{1, \dots, \ell\} \setminus \{j\}$. We describe the construction of the layered graph in Figure 3.

1. Layer 0 and layer $L + 1$ contain a single node. The node in layer 0 is labeled with the packing vector, while the node in layer L is labeled with the all-zero vector.
2. Sort the large items in some way. Each large item is associated with a layer between 1 and L . Each of these layers contains one node for *every* (nonnegative, integer) vector that is smaller than the original packing vector.
3. For a node u , denote the cutoff vector by $c(u)$ and the summary vector by $s(u)$. For any node u in layer i ($i = 0, \dots, L - 1$), there is an arc to every node v in layer $i + 1$ such that $c(u) \rightarrow c(v)$ and $s(u) \geq s(v)$.
4. The cost of arc (u, v) , where v is in layer i ($i = 1, \dots, L$), is the cost of packing the i th large item excluding a piece of size specified by the nonzero entry in $c(u) - c(v)$ (this size may be 0), together with the items specified by $s(u) - s(v)$.
5. For every node u in layer L , there is an arc to the single node in layer $L + 1$. The cost of this arc is the cost of packing all items in $s(u)$.

Figure 3: Construction of the layered graph for one set of guesses (cuts)

The cost of an edge (u, v) that is mentioned in Step 4 of Figure 3 can be computed as follows. This step creates a star rooted at a given large item (the i -th item in the list of large items is associated with layer i). The size of the large item that needs to be packed, is given by its original size minus the size of the part of item which corresponds to the nonzero entry of $c(u) - c(v)$. This item is to be packed with items specified by $s(u) - s(v)$. The only item that we cut further at this point is the large item associated with the current layer. Moreover, that is the only item that may be combined with other items. Thus, if we denote the sizes of items specified by $s(u) - s(v)$ by a_1, \dots, a_p and the size of the part of the large item that needs to be packed by X , then the number of bins is $\max\{p, \lceil X + \sum_{i=1}^p a_i \rceil\}$.

The cost of an edge (u, v) that is mentioned in Step 5 of Figure 3 can be computed as follows. The items to pack here are specified by $s(u)$. These items are not split further, they are packed in bins containing one or two of these items. We apply the First-Fit-Decreasing algorithm with the restriction that no bin can contain more than two items. By Lemma 11, this gives an optimal packing.

Lemma 11 *FFD is an optimal algorithm for cardinality constrained bin packing for $k = 2$.*

Proof We modify the input as follows. For an item $x > 0$ let $x' = (x + 1)/3$. Then $1/3 < x' \leq 2/3$. Three modified items clearly do not fit together, and for two items $x' + y' \leq 1 \iff x + y \leq 1$.

Thus the number of bins required to pack the modified input is the same as for the original input. We now have an input where all items are larger than $1/3$. It is known [15] that for such an input, FFD gives an optimal solution. \square

Packing the original input Once we have found the set of cuts that allows the best packing, it is easy to find the packing for the original input items. Say large item 1 (in our ordering) is packed into bins together with parts of size $k_1\delta, k_2\delta, \dots, k_{a_1}\delta$. Using the original vector that represents the set of cuts, we find the first i such that there exists an item of size $i\delta < 1$ which is cut at $k_1\delta$, or at $(i - k_1)\delta$, and the part of size $k_1\delta$ that is created by this cut is so far unpacked. We then mark this part as packed and continue. (For each item size less than 1, we keep track of how many first and second parts are packed of each size.)

The correct part of this item of size less than 1 is put in bin 1. Bin 1 is filled with some part of large item 1 (namely, $1 + 2\delta - k_1\delta$). Then we find an unpacked part for bin 2 in the same manner, etc. At

the end we have some part of the large item left, exactly how large this is determined by what piece was cut off from the first large item. If this part has a positive size, it is packed in consecutive bins, and we move to the next large item. Finally, we find parts that are paired up in the same manner. Each bin contains only two parts, and we rounded up to the nearest multiple of δ , so we can use bins of size $1 + 2\delta$ to pack the unrounded parts.

Lemma 12 *The running time of this algorithm is $n^{O(1/\varepsilon^2)}$.*

Proof As stated above, a set of cuts can be represented by a vector of length $(K + 1)^2/4 + K + 1$.

The total number of options for such a vector is

$$\binom{L + K - 1}{L} \prod_{i=(K+1)/2}^K \binom{M_i + K - 1}{M_i} \leq \left(\prod_{i=(K-1)/2}^K (M_i + K - 1) \right)^{(K+3)/2}$$

where $L + \sum M_i \leq n$. This implies $\sum_i (M_i + K - 1) \leq n + \frac{K+3}{2}(K-1)$ and therefore $\prod_i (M_i + K - 1) \leq \left(\frac{2n}{K+3} + K - 1\right)^{(K+3)/2}$ which means we have at most $\left(\frac{2n}{K+3} + K - 1\right)^{(K+3)^2/4}$ options to cut the input items. This is an upper bound for the number of graphs that we need to consider, and it is polynomial in n .

How many nodes are there in layer 1 of one of these graphs? Denote the number of parts of size $i\delta$ in the summary vector by n_i , and in the cutoff vector by m_i . We have $\sum_{i=0}^K n_i = n + L$ and $\sum_{i=0}^K m_i = L$. For entry i in the summary vector, there are $n_i + 1$ possibilities, and similarly in the cutoff vector. This gives us

$$\prod_{i=0}^K ((n_i + 1)(m_i + 1))$$

possibilities overall. This number is upper bounded by

$$\left(\frac{2n}{K+1} + 1\right)^{2(K+1)},$$

which is polynomial in n . There are at most n layers in the graph. Thus, the overall size of the graph is bounded by $n \left(\frac{2n}{K+1} + 1\right)^{2(K+1)}$, which means that we can find the path with minimal cost in time

$$n^2 \left(\frac{2n}{K+1} + 1\right)^{4(K+1)}.$$

Overall this gives a running time of

$$\begin{aligned} & \left(\frac{2n}{K+3} + K - 1\right)^{(K+3)^2/4} n^2 \left(\frac{2n}{K+1} + 1\right)^{4(K+1)} \\ & \leq n^2 \left(\frac{2n}{K+1} + K - 1\right)^{\frac{1}{4}K^2 + \frac{11}{2}K + 6\frac{1}{4}} = n^{O(1/\varepsilon^2)}. \end{aligned}$$

□

Lemma 13 *This algorithm uses at most $\text{OPT}(L)$ bins of size $1 + 2\delta$ to pack the input L .*

Proof The optimal solution of the original instance (in bins of size 1) can be adapted to pack the rounded items (to the nearest multiple of δ) in the same number of bins of size $1 + 2\delta$, using only cuts at multiples of δ . Denote this packing by P . The PTAS tries all possible packings of this form

for the rounded items and thus tries the packing P at some point. Therefore, it manages to pack the original items in bins of size $1 + 2\delta$, needing at most the optimal number of bins for these items. Our algorithm begins by rounding down the input L to a modified input L' . Clearly, $\text{OPT}(L') \leq \text{OPT}(L)$. By the observations above, for the input L' it is sufficient to cut items at integer multiples of δ . Since the algorithm tries all possible such cuts, and finds an optimal packing for each set of cuts, we conclude that our algorithm does not use more than $\text{OPT}(L')$ bins. To get a packing of the original input items, we need to take into account that these items were rounded down to the nearest multiple of δ . To place the original input items, it is sufficient if we enlarge the space for **one** part of each item by δ . Since two items of size just over $1/2$ could get rounded down to below $1/2$, we need to enlarge each bin by 2δ . That is, for each part in the packing, we enlarge the piece of the bin that is allocated to it by δ . Since there are at most two items per bin, this gives us enough room to place all the original input items. \square

Taken together, these two lemmas prove the following theorem.

Theorem 3 *For any $\varepsilon > 0$, there exists a polynomial-time algorithm for cardinality constrained bin packing of splittable items where each bin is allowed to have at most two items or parts of items. This algorithm gives packs the items in the optimal number of bins, but uses bins of size $1 + \varepsilon$.*

5 A dual PTAS for constant k

We give an algorithm for packing the input items into the optimal number of bins, but where the bins have size $1 + \varepsilon$. In fact we will pack the items in bin of size $1 + k\delta$, where δ depends on ε and k . Therefore, we only have a dual PTAS for the case where k is constant. We choose ε , so that δ is the inverse of some odd integer. Let $M = 1/\delta + k$. All items of size more than $1 + k\delta = M\delta$ are called large.

We will again use the fact that there is an optimal packing which is a forest (Lemma 4). We modify the input in two steps.

Sizes of items and parts. A first step would be a revision of sizes of items and parts of items. We take an optimal packing, and replace any part of size x with a part of size $\lceil \frac{x}{\delta} \rceil \delta$. As a result, the total size of parts in a bin can increase by an additive factor of at most $k\delta$. Therefore from this time on, we use bins of size $1 + k\delta$. One problem is that the sizes of items may have increased in an unbounded way. We would like to change the size of an item of size y to exactly $\lceil \frac{y}{\delta} \rceil \delta$. Therefore, we repeatedly pick a packed piece of an item whose size increased too much, and decrease its size by δ . This is done until all items are back to the desired size.

All parts in the packing now have sizes that are multiples of δ . Note that it may happen that the number of bins used decreases, if there are bins where all the pieces in it have their size reduced to 0.

Large items. As in the previous Section (Lemma 10), we would like to pack the large items one by one and not combine them together into bins. Note that we showed in Section 4 that Lemma 3 still holds in this case. It is straightforward to adapt the proof of Lemma 10 for the case where the bins have size $1 + k\delta$ and large items have size more than $1 + k\delta$ (instead of 1). Thus we find that for each large item, we can cut off one part of size at most $1 + k\delta$, and moreover this part is not cut further later.

Non-large items. We now consider the non-large items (size at most $1 + k\delta$). We need to allow these items (except non-large parts cut off from large items) to be cut at every integer multiple of δ . This is sufficient since in the optimal packing all parts have sizes that are integer multiples of δ . Moreover, by Lemma 5, it is sufficient to let non-large items be cut at most $k' = k(1 + k\delta) < 2k$ times. Therefore our scheme need only check such packings.

Description of the dual PTAS We begin by rounding up all items into integer multiples of δ . To convert our packing into a packing of the original instance, for each item of original size y we need to decrease the size of at most one of its parts by $\lceil \frac{y}{\delta} \rceil \delta - y$ (this amount may be zero). From now we only

discuss the rounded items. Note that these items are the same as used in the adapted optimal packing described above.

After rounding, the non-large items in the input can be represented by a vector indicating how many items exist of each non-large size, out of M possible non-large sizes. For each size, the number of parts cut off from those items of a particular smaller size can also be represented by a vector. We need to try all possibilities for these cutoff vectors. For each possibility, we will enumerate all possible packings of the items of size at most $1 + k\delta$ into bins of size $1 + k\delta$ such that no bin is empty. Here we use the fact that there is only a constant number of different packings of one bin (patterns), and a packing can be specified by giving how often each pattern is used.

For each such packing, we will construct a layered graph similar to the one in the previous section, with one layer for each large item. Each node now represents a subset of the bins of the current packing. The cost of an edge between two nodes is determined by the difference packing vector and the size of the large item of the current layer.

Guess vectors We construct a guess vector with at most $M(2 + (M + 1)^{k'})$ entries. We will try all guess vectors to find the best solution among them, which is not worse than the adapted optimal solution we consider. In the guess vector, for each non-large size, there are at most $2 + (M + 1)^{k'}$ entries. We have a first entry which is the amount of such items in the input. A second entry is a number of large items from which this size of non-large item is cut off. The other entries are amounts of items of this size that are cut according to a given pattern. There are $M + 1$ options for each cut and at most k' cuts for each non-large item, therefore there are at most $(M + 1)^{k'}$ possible patterns (actually the number is less for smaller items). A guess vector is valid if the following conditions hold.

- The number of items of each non-large size in the input is identical to the respective first entries.
- The sum of second entries is at most the number of large items. (Some large items might not have a part cut off.)
- The sum of other entries (not first or second) of each size is equal to the first entry for this size.

The number of non-large items is at most n , therefore no component in the guess vector exceeds n , and there are $n + 1$ options for each component. Therefore there are at most $(n + 1)^{M(2 + (M + 1)^{k'})}$ valid guess vectors. This number is polynomial in n for constant ε and k .

Short guess vectors Once a guess vector is given, we can summarize its contents as follows. We have a summary vector with M entries, where entry i is the total number of parts of size $i\delta$. Additionally, we have a cutoff vector with $M + 1$ entries, where entry i denotes the number of non-large items of size $(i - 1)\delta$ that are cut off from the large items for $i = 1, \dots, M + 1$. For $i > 1$, this is taken from the second entry for the $(i - 1)$ th size in the guess vector. The first entry is simply the number of large items minus the sum of the other entries in the cutoff vector, and is the number of large items that do not have a non-large item cut off from them. We concatenate these vectors into a *short guess vector* of length $2M + 1$. We can build a packing based on the short guess vector, and later specify which items the parts belong to (the parts of items which have the same size are interchangeable in the packing).

Patterns A pattern is a list of $k + 1$ integers i_1, \dots, i_k that indicate where the different parts end: the j th part in this bin starts at $i_{j-1}\delta$ and ends at $i_j\delta$ (let $i_0 = 0$). Note that the bin may contain less than k parts (in this case we write $i_j = 0$ starting from some j) and/or be partially empty (so we cannot omit the number i_k). For each number i_j , there are $M + 1$ options. Thus the number of patterns is at most $T = (M + 1)^k$, which is constant (as a function of k and ε).

Guess packings In order to be able to use a layered graph, we would need to build guess packings. A packing is a set of bins which are partially packed. Each bin is packed according to a pattern. Note that the total number of parts of *non-large* items to be packed does not exceed nk' , since each non-large item is cut into at most k' parts.

A guess packing vector is a vector of length T where entry i denotes the number of bins packed according to pattern i for $i = 1, \dots, T$. A guess packing vector is valid for a given short guess vector if the total number of parts of each size is the same in both. The number of possible guess packing vectors that need to be checked for each short guess vector is at most $(nk' + 1)^T$, since no bin is empty in the packing, the total number of bins is at most nk' . Therefore this number is polynomial in n and ε .

The required information in order to pack the remaining *large* items via a layered graph is the guess packing vector, and the cutoff vector (second part of the short guess vector). We concatenate these two vectors into a single *final* vector of length $T + M + 1$.

Layered graph Finally, we show how to define a layered graph as before, where layers $1, \dots, L$ correspond to the L large items. Bins with exactly k items in the guess packing vector are full, and others can receive parts of large items in the scheme. Therefore the full bins do not need to participate in the scheme, and are removed from the guess packing vector before we construct the graph (noting how many such bins we remove).

We use $|L| + 1$ layers. The vertices of each layer are vectors that are smaller than the final vector (i.e., including the cutoff vector). Layer zero has a single vertex which corresponds to the given packing guess vector, and the full set of cut off parts. All other layers have all possible vectors that are smaller than this vector.

A node v in layer i is connected to a node x in layer $i + 1$ if the following conditions hold.

- The cutoff part of v minus the cutoff part of x is a unit vector (i.e., all components are zero, except for one the is 1). Denote the (non-large) size associated with the non-zero component by a . Note that a may be 0.
- Let z be the size of the large item. Let $z' = z - a$. The size that is left to be packed is z' . (The part of size a is packed in some other step.) Consider now the guess packing part of v minus the guess packing part of x . We require that there are no negative entries in the difference vector. This difference relates to a set of packed bins with $k - 1$ or less parts. Denote the total empty space in these bins by b . If $z' \leq b$, this means that the large item can be packed entirely in these bins. In this case the edge costs 0. Otherwise, the cost of the edge is $\lceil z' - b \rceil$. This is the number of bins still needed to complete the packing.
- We are interested in the shortest path from layer zero to any node in the last layer. (This is the reason we do not count the bins in the guess packing vector in the cost of the edges: they are fixed and we are only interested in the extra cost of packing the large items.)
- The cost of the packing is the cost of the path, plus the number of bins in the packing guess vector, including the full bins that do not participate in the scheme.

Naturally, we choose the best solution ever found, and translate it to a packing of parts of items.

The number of bins used The optimal solution P of the original instance can be adapted to pack the rounded items (to the nearest multiple of δ) in the same number of bins or less, using only cuts at multiples of δ . Denote this packing by P' . The PTAS tries all possible packings of this form for the rounded items and thus tries the packing P' at some point. Therefore, it manages to pack the original items in bins of size $1 + k\delta$, needing at most the optimal number of bins for these items.

6 Conclusions

In this paper, we provided approximation schemes for bin packing of splittable items with cardinality constraints for all values of k . We also provided dual approximation schemes. It should be noted that our upper bounds are absolute, i.e. there is no additive term.

References

- [1] Luitpold Babel, Bo Chen, Hans Kellerer, and Vladimir Kotov. Algorithms for on-line bin-packing problems with cardinality constraints. *Discrete Applied Mathematics*, 143(1-3):238–251, 2004.
- [2] Alberto Caprara, Hans Kellerer, and Ulrich Pferschy. Approximation schemes for ordered vector packing problems. *Naval Research Logistics*, 92:58–69, 2003.
- [3] Fan Chung, Ronald Graham, Jia Mao, and George Varghese. Parallelism versus memory allocation in pipelined router forwarding engines. *Theory of Computing Systems*, 39(6):829–849, 2006.
- [4] Wenceslas Fernandez de la Vega and George S. Lueker. Bin packing can be solved within $1+\epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.
- [5] Leah Epstein. Online bin packing with cardinality constraints. *SIAM Journal on Discrete Mathematics*, 20(4):1015–1030, 2006.
- [6] Leah Epstein and Rob van Stee. Improved results for a memory allocation problem. In *Workshop on Algorithms and Data Structures (WADS 2007)*, pages 362–373, 2007.
- [7] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.
- [8] Ronald L. Graham and Jia Mao. Parallel resource allocation of splittable items with cardinality constraints. Manuscript.
- [9] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.
- [10] Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 312–320, 1982.
- [11] Hans Kellerer and Ulrich Pferschy. Cardinality constrained bin-packing problems. *Annals of Operations Research*, 92:335–348, 1999.
- [12] K. L. Krause, V. Y. Shen, and Herbert D. Schwetman. Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. *Journal of the ACM*, 22(4):522–550, 1975.
- [13] K. L. Krause, V. Y. Shen, and Herbert D. Schwetman. Errata: “Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems”. *Journal of the ACM*, 24(3):527–527, 1977.
- [14] Hadas Shachnai, Tami Tamir, and Omer Yehezkely. Approximation schemes for packing with item fragmentation. In *Proceedings of the 3rd Workshop on Approximation and Online Algorithms (WAOA 2005)*, pages 334–347, 2006.

- [15] D. Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Research Logistics*, 41(4):579–585, 1994.