Approximation schemes for packing splittable items with cardinality constraints [*]

Leah Epstein[†] Asaf Levin[‡] Rob van Stee[§]

August 3, 2010

Abstract

We continue the study of bin packing with splittable items and cardinality constraints. In this problem, a set of n 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 given parameter. Complicating the problem further is the fact that items may be larger than 1, which is the size of a bin. The problem is known to be strongly NP-hard for any fixed value of k.

We essentially close this problem by providing an efficient polynomial-time approximation scheme (EPTAS) for most of its versions. Namely, we present an efficient polynomial time approximation scheme for $k = o(n)$. A PTAS for $k = \Theta(n)$ does not exist unless P = NP.

Additionally, we present *dual* approximation schemes for $k = 2$ and for constant values of 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 I of n *items* is given and the goal is to pack them into the minimum number of containers, called *bins*. The items are typically given as numbers in $(0, 1]$, where 1 is the bin size. In this paper we consider items that may be larger than 1, that is, their sizes are in $(0, \infty)$. Items are allowed to be *split* and distributed among an arbitrary number of bins. The size of an item i is denoted by s_i.

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 $k \geq 2$ [4, 10].

This problem was introduced by Chung et al. [4], 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 have large variations 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 large amount of contention which is also undesirable. It is currently infeasible to build a large, fast shared memory and in practice, such memories

[*]Some of the results in this paper (for constant k) have appeared in the extended abstract [9].

[†]Department of Mathematics, University of Haifa, 31905 Haifa, Israel. `lea@math.haifa.ac.il`.

[‡]Chaya fellow. Faculty of Industrial Engineering and Management, The Technion, 32000 Haifa, Israel. `levinas@ie.technion.ac.il`.

[§]Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany. `vanstee@mpi-inf.mpg.de`. Research supported by the Alexander von Humboldt Foundation and the German Research Society (DFG).

are time-multiplexed. For n processors, this increases the effective memory access time by a factor of n.

Chung et al. [4] suggested a new architecture where each memory may be accessed by at most *two* processors, avoiding the disadvantages of the two extreme models discussed above. 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 [10, 12]. We describe the results of these papers below. First, we define the performance measure that we use.

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 $\mathcal{I}$. We also denote by $\mathcal{B}(\mathcal{I})$ the solution which algorithm $\mathcal{B}$ returns on the instance $\mathcal{I}$. Note that $\mathcal{B}(\mathcal{I})$ refers both to the cost of the solution and to the solution itself, however, it will be clear from the context which is used in every case. An algorithm $\mathcal{A}$ is an (absolute) $\mathcal{R}$-approximation if for every input $\mathcal{I}$, $\mathcal{A}(\mathcal{I}) \leq \mathcal{R} \cdot \textsc{opt}(\mathcal{I})$, where $\textsc{opt}$ 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. A polynomial time approximation scheme (PTAS) is a family of $(1+\varepsilon)$-approximations for every value of $\varepsilon > 0$. An efficient polynomial time approximation scheme (EPTAS) is a PTAS whose time complexity is of the form $f(\frac{1}{\varepsilon}) \cdot \textsc{poly}(\textsc{length}(\mathcal{I}))$ where f can be an arbitrary function (typically, an exponential function) and $\textsc{poly}(\textsc{length}(\mathcal{I}))$ is a polynomial of the input length. The notion of EPTAS is a modern one, motivated in the *fixed parameterized tractable* (FPT) community (see e.g. Cesati and Trevisan [3]). Recall that a fully polynomial time approximation scheme (FPTAS) is a PTAS whose time complexity is polynomial in the input length and in $\frac{1}{\varepsilon}$.

The *asymptotic* approximation ratio for an algorithm $\mathcal{A}$ is defined to be

$$\mathcal{R}^{\infty}_{\mathcal{A}} = \limsup_{N\to\infty} \sup_{\mathcal{I}} \{ \frac{\mathcal{A}(\mathcal{I})}{\textsc{opt}(\mathcal{I})} | \textsc{opt}(\mathcal{I}) = N \} \ .$$

This ratio is relevant if we are particularly interested in the performance of algorithms on large inputs, which cannot be packed into just a few bins. If the generic approximation ratio involved in an approximation scheme is defined according to the asymptotic measure, then it is called asymptotic (resulting in the concepts of an APTAS, AEPTAS, and AFPTAS).

Fernandez de la Vega and Lueker [5] designed an APTAS for standard bin packing. Their work was followed by the work of Karmarkar and Karp [14] 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 [11]) shows that no polynomial-time algorithm has an absolute performance guarantee better than $\frac{3}{2}$ unless P = NP. This reduction is no longer valid for our problem, where items may be split.

Chung et al. [4] showed that the bin packing 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 [11]). In [10], Epstein and van Stee showed that this problem is NP-hard in the strong sense for any fixed constant value of k. Chung et al. [4] also gave a $3/2$-approximation for the case $k = 2$. Graham and Mao [12] 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 [10], a simple algorithm with an absolute approximation ratio of $2 - 1/k$ for $k \geq 2$, and an algorithm with an asymptotic approximation ratio of $7/5$ for $k = 2$ were presented.

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 [16, 17]. They showed that the performance guarantee of the well known First Fit algorithm is at most $2.7 - \frac{12}{5k}$. Additional results of [16, 17] were offline approximation algorithms of performance guarantee 2. Kellerer and Pferschy [15] designed an improved offline approximation algorithm with performance guarantee 1.5, and an APTAS was designed by Caprara, Kellerer and Pferschy in [2] (for a more general problem). Finally, an AFPTAS for this problem was obtained by Epstein and Levin in 2007 [8].

From a different perspective, Babel et al. [1] designed a simple *online* algorithm with an asymptotic approximation ratio of 2 for any value of k. They also designed improved algorithms for $k = 2, 3$. Epstein [7] 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 $3 \le k \le 6$.

A related problem was studied by Shachnai, Tamir and Yehezkely [18]. They considered two variants of an offline bin packing problem where items may be split arbitrarily: one where splitting items comes at a cost, as each part of a split item increases the size of the item by a constant additive factor, and one where there is an upper bound on the total number of splits. They showed that both variants do not admit a PTAS unless P = NP. They designed asymptotic approximation schemes for both variants. 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. In a follow up paper, Shachnai and Yehezkely [19] designed an AFPTAS for each of the two variants.

Our results Our first main result is an efficient polynomial-time approximation scheme. Recall that for standard bin packing, this is impossible unless P $=$ NP. We present some special cases, then we present our scheme for the case $k = 2$ and finally we show how to extend it to the case $k = o(n)$. The main difficulty of packing splittable items, especially for variable k, is that we have less structure in the packing, due to possible splits of items, and in particular, of very large items, making it harder to search all potential packings efficiently. Note that it is NP-hard to approximate the problem with $k = \frac{n}{2}$ within a factor smaller than $\frac{3}{2}$. Specifically, this follows from a reduction from the EQUAL CARDINALITY PARTITION problem. Therefore, it is impossible to obtain a PTAS for all values of k.

We also present a dual PTAS for this problem, first for $k = 2$ and then for general constant values of 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 procedure used in the PTAS for scheduling on identical machines, which was given by Hochbaum and Shmoys [13].

Throughout this paper, we let $0 < \varepsilon \le \frac{1}{20}$ be such that $\frac{1}{\varepsilon}$ is an even integer.

2 Efficient polynomial time approximation schemes

Denote by W the total size of all items. We will assume that $W \le n$ if $k \ge 3$ and $W \le n^2$ if $k = 2$. These assumptions are made without loss of generality, since otherwise we can solve the problem in polynomial time as shown in the following lemma.

Lemma 1 *The problem is polynomially solvable in either case: (i) $W > n$ and $k \ge 3$; or (ii) $W > n^2$ and $k = 2$.*

Proof First assume that $k \ge 3$ and $W > n$. Then we have $\text{OPT} \ge \lceil W \rceil > n$. Consider the following solution. The solution packs each item of size at most 1 to a bin. The larger items are packed into bins using (fractional) Next-Fit, starting with the bins containing single items, and possibly using empty bins afterwards. As a result, no bin contains parts of more than three different items (as it contains parts of at most two larger items, and at most one item of size at most 1). Moreover, all bins but possibly the last one will contain a total size of 1. Thus the number of created bins is at most $\lceil W \rceil \leq \text{OPT}$, and hence the problem can be solved optimally by this algorithm.

Next assume that $k = 2$, and $W \geq n^2$. Then there exists an item of size at least n. We use the same algorithm as described above, but particularly starting the application of Next-Fit with the largest item. Since there are at most n bins containing items of size at most 1, and the size of the largest item is at least n, Next-Fit will complete the packing of the largest item no earlier than in the $(n + 1)$st bin. Therefore, among the first n bins, no bin will contain parts of more than two items, and this property will be maintained while packing the remaining items of size greater than 1 as well. Therefore, the algorithm returns an optimal solution for this case as well.

To prove the lemma, it suffices to show that the two algorithms run in polynomial time. For both algorithms, any item X of size more than 1 is packed into bins of only two types. A bin of the first type is a bin which is either not full or contains an item (or a part of an item) different from X. A bin of the second type contains a part of X of size 1. Note that the number of bins of the first type is at most n (since each such bin contains the last part of some item) while bins of the second type are consecutive and hence it is possible to give a compact representation of the list of these bins. □

We now present structural properties which will be used in this section. In addition, we provide (approximation) algorithms for several special cases, including the case $\varepsilon > \frac{1}{\sqrt{k}}$. Afterwards, we present the scheme for $k = 2$, and finally we present the scheme for $k = o(n)$.

2.1 A first modification to the input

We start with a modification to the input which will be used for the design of all efficient approximation schemes. We show that it can be assumed that no item has a size above $\frac{1}{\varepsilon}$. Recall that the original input is denoted by I. We modify the input as follows and let the modified input be denoted by I'. 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). The following holds for any $k \geq 2$.

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

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

For an item X of size $x > 1/\varepsilon$, consider the bins in which it is packed in some order, and let us number these bins $1, 2, \ldots$. Let S_j denote the total size of parts of X packed into bins $1, 2, \ldots, j$, and let $S_0 = 0$. For every integer $1 \leq i \leq \varepsilon \cdot x$, let j_i be the value of j for which $S_{j-1} < \frac{i}{\varepsilon}$ and $S_j \geq \frac{i}{\varepsilon}$. The value j_i is well defined for any $1 \leq i \leq \lfloor \varepsilon \cdot x \rfloor$. The part in bin j_i is cut into two parts of sizes $S_{j_i} - \frac{i}{\varepsilon}$ and $\frac{i}{\varepsilon} - S_{j_i - 1}$. If $S_{j_i} > \frac{i}{\varepsilon}$ then the first part has a nonzero size, and we pack it into a new, empty bin. Thus we need at most one extra bin for each multiple of $1/\varepsilon$ of the total size of X. □

Note that in this process, the number of items cannot increase by a large number. For any item of size $\Gamma > \frac{1}{\varepsilon}$, the number of new items is at most $\lfloor \varepsilon\Gamma \rfloor$. If $k \geq 3$, then the number of new items is at most $\varepsilon W \leq \varepsilon n$. If $k = 2$, then the number of new items is at most εn^2. For $k \geq 3$, the resulting number of items in I' is no larger than $n(1 + \varepsilon)$, while for $k = 2$, it is polynomial in n and $\frac{1}{\varepsilon}$. Therefore, this process takes polynomial time. In this section, we abuse notation and use n to denote the number of items in I'. For $k \geq 3$ this does not change the fact that $k = o(n)$.

Lemma 2 implies that it is sufficient to apply an efficient polynomial time approximation scheme on I'. Note that a packing of I' implies a packing for I having the same cost.

2.2 Optimal solutions for the cases $n \leq \frac{1}{\varepsilon^{10}}$ and $k \geq \varepsilon n$

Recall that we assume that $k = o(n)$, that is, k is a function $k(n)$ of n, such that $\limsup_{n\to\infty} \frac{k(n)}{n} = 0$. This assumption clearly holds for any constant value of k, but also for functions such as $k(n) = \frac{n}{\log_2 \log_2 n}$. Therefore, by the assumption $k = o(n)$, and considering the case $k \geq \varepsilon n$, then there exists a constant n_ε such that $n \leq n_\varepsilon$. In the example where $k(n) = \frac{n}{\log_2 \log_2 n}$, we have $n_\varepsilon = 2^{2^{1/\varepsilon}}$. Note that if $\text{OPT}(I') \leq \frac{1}{\varepsilon}$, then $\frac{n}{k} \leq \frac{1}{\varepsilon}$, and therefore this case will be covered in this section.

Lemma 3 *Given the input I', if n is a constant, it is possible to compute a solution of cost $\text{OPT}(I')$ in constant time (which depends on the value of ε). That is, the problem can be solved (optimally) in constant time for the instance I'.*

Proof Since the size of each item in I' is at most $\frac{1}{\varepsilon}$, $\text{OPT}(I') \leq \frac{n}{\varepsilon}$. Our algorithm for this case initializes $\frac{n}{\varepsilon}$ bins. We use enumeration to assign at most k items to each bin. There are at most n^k subsets of at most k items, so there are at most $(n^k)^{n/\varepsilon}$ possible assignments. For each assignment, the number of bins which it requires is the number of bins which have at least one item assigned to them.

Next, for each assignment, we construct a feasibility-checking linear program which outputs a packing if such a packing exists. There are at most $\frac{n^2}{\varepsilon}$ variables, where there is a variable $z_{i,j}$, if one of the items assigned to the j-th bin is the i-th item. If the i-th item was not assigned to the j-th bin then we let $z_{i,j} = 0$. In addition to the non-negativity constraints, $z_{i,j} \geq 0$, there are two families of constraints. A constraint of the first one is defined for each bin j, and it states that the total size of parts of items packed into bin j is at most 1, that is, $\sum_{i \in I'} z_{i,j} \leq 1$. A constraint of the second family state that each item i is completely packed (fractionally), that is, $\sum\limits_{1 \leq j \leq n/\varepsilon} z_{i,j} = s_i$. Note that the cardinality constraints are ensured by the guessing step.

Since both the number of variables and the number of constraints are polynomial in n and $\frac{1}{\varepsilon}$, and since all coefficients are binary an exact solution can be computed in strongly polynomial time (since the dual can be solved in strongly polynomial time, using the algorithm of [6]). The output is the valid packing with the minimum number of required bins. □

Thus, we have solved this special case and in the remainder of Section 2, we can make the following assumptions:

$$k \leq \varepsilon n, \; n > \frac{1}{\varepsilon^{10}}, \text{ and } \text{OPT}(I') \geq \frac{1}{\varepsilon} \, . \tag{1}$$

2.3 An approximation algorithm for large values of k

We next deal with the case $k > \frac{1}{\varepsilon^2}$, or equivalently $\varepsilon > \frac{1}{\sqrt{k}}$.

Lemma 4 *If $\varepsilon > \frac{1}{\sqrt{k}}$, then there exists an approximation algorithm for cardinality constrained bin packing of splittable items with an approximation ratio of $1 + 6\varepsilon$ and a running time of $O(\frac{n^2}{\varepsilon^2})$.*

Proof Recall that I' consists of n items, each of size at most $\frac{1}{\varepsilon}$. Thus the number of bins in an optimal solution take at most $\frac{n}{\varepsilon}$ distinct values.

In this proof, we say that an item is large if its size is at least ε, and otherwise it is small. Small items are partitioned into medium items, of size in $(2\varepsilon^2, \varepsilon)$ and tiny items, of size in $(0, 2\varepsilon^2]$.

The first step is to partition each large item into one or more pieces such that each piece will have size between ε and 2ε (this can be achieved by cutting off pieces of size ε, until a piece of size at most 2ε remains). Using $W \leq n$, the resulting number of large items is at most $\frac{n}{\varepsilon}$. We sort the resulting set of items (pieces of large items and small items) in non-increasing order of their size in time $O(n \log n)$ (the newly created items of size ε can be "sorted" in a single step).

For each possible value g for $\text{OPT}(I')$ out of the $\frac{n}{\varepsilon}$ possible values, we apply the following. We first allocate the items to g bins in a round-robin fashion, so that bin i receives items with index $i + gt$ for $t = 0, 1, \ldots$, where the indices of the items are their positions in the sorted list of items. Since in every round the first bin received an item which is not smaller than the item which the last bin received, while in the last round, the last bin possibly did not receive an item at all, the last bin is the least loaded one, while the first bin is the most loaded one. However, if we remove the first item from each bin except for the last one, we are in the situation that the assignment started from the last bin.

For $g = \text{OPT}(I')$, the total size of items is at most g, and therefore, the last bin contains a total size of items of at most 1. Therefore, if we remove the largest item from each bin, then the resulting set of items in each bin have total size of at most 1. The removed items are packed into $\lceil 2\varepsilon g \rceil$ additional bins where $\frac{1}{2\varepsilon}$ such items are packed into a common bin. Note that these additional bins are feasible both with respect to the total size of their items (since the size of each item after partitioning the large items is at most 2ε), and with respect to their cardinality since $k > \frac{1}{\varepsilon^2}$ and $\frac{1}{2\varepsilon} < \frac{1}{\varepsilon^2}$.

We next note that for the correct value of g, the number of parts of small items in each bin is at most k. To see this denote by n' the number of small items, and recall that small items are not cut into pieces. Each bin contains therefore at most $\lceil \frac{n'}{g} \rceil = \lceil \frac{n'}{\text{OPT}(I')} \rceil \leq \lceil \frac{n'}{n'/k} \rceil = k$ small items, where the inequality holds since in $\text{OPT}(I')$ each bin contains up to k parts of small items. Recall that each piece of a large item has a size of at least ε and hence there are up to $\frac{1}{\varepsilon}$ pieces of large items in each bin. Therefore, the current set of items in each bin may exceed the cardinality constraint of k but no bin has more than $k + \frac{1}{\varepsilon}$ parts of items. Moreover, the number of parts of items which are either medium or large is at most $\frac{1}{2\varepsilon^2}$, since each such part has a size of at least $2\varepsilon^2$. For a bin which contains at most $\frac{1}{\varepsilon}$ tiny items, all of them are removed, leaving at most $\frac{1}{2\varepsilon^2} < k$ items in the bin. Otherwise, there are at least $\frac{1}{\varepsilon}$ tiny items, so it is possible to remove the smallest $\frac{1}{\varepsilon}$ items of each bin, leaving at most k items in the bin. The removed items are partitioned into groups, where a group of removed items consists of items removed from $\frac{1}{2\varepsilon}$ bins. Each group of items contains at most $\frac{1}{2\varepsilon^2} < k$ items and it is packed into one common dedicated bin, so the new dedicated bins clearly satisfy the cardinality constraint. Each removed item is tiny, so the total size of $\frac{1}{2\varepsilon^2}$ removed items does not exceed 1. The number of new bins is at most $\lceil 2\varepsilon g \rceil$ again.

To conclude the performance of our approximation algorithm, note that for the correct value of g, it returns a feasible solution whose cost is at most $\text{OPT}(I') + 2\varepsilon\text{OPT}(I') + 2\varepsilon\text{OPT}(I') + 2 = (1 + 4\varepsilon)\text{OPT}(I') + 2$. By (1), $\text{OPT}(I') \geq \frac{1}{\varepsilon}$, and hence our algorithm returns a solution of cost at most $(1 + 6\varepsilon)\text{OPT}(I')$, as claimed. For each guess g, the algorithm can be implemented to run in $O(g + n)$ time by treating the items of size ε as a block in each bin. Hence the overall running time is $O(\frac{n^2}{\varepsilon^2})$. $\square$

We are left with the case $k \leq \frac{1}{\varepsilon^2}$. Since $\text{OPT}(I') \geq \frac{n}{k}$ we get that in the remaining cases it holds that

$$\text{OPT}(I') \geq n\varepsilon^2. \tag{2}$$

2.4 A second modification to the input - linear grouping

Given an input to be packed, create groups of items, and let p be the number of *groups*, which is defined to be $p = \frac{1}{\varepsilon^4}$, so $p < n$ since $n \geq \frac{1}{\varepsilon^{10}}$ by (1). We sort the items of I' by nonincreasing size and put the items into groups of $\lceil \frac{n}{p} \rceil$ successive items (the last group may contains a smaller number of items).

We create an instance I'' by modifying I' as follows. First, we remove the first group (the one with the largest items). For each remaining group, we round the item sizes in this group up to the size of the largest item in the group. This creates the input I''.

Lemma 5 $\text{OPT}(I'') \leq \text{OPT}(I')$.

Proof We show that every packing of I' can be converted into a packing of I'' with the same number of bins. This can be done by replacing every item of I' with an item of the next group of I'' (removing some items of I' which are not replaced with any item of I''). Each item $X \in I'$ is replaced with an item $\hat{X} \in I''$ which is not larger than X. □

Lemma 6 *There exists a polynomial time algorithm which converts a packing of I'' into a packing of I' using at most $\frac{2n}{p\varepsilon}$ additional bins.*

Proof We show how to convert a packing of I'' into a packing of I'. Clearly, every packing of I'' can be converted into a packing of I' without the first group, by replacing the size of each item of I'' with its original size. Specifically, wherever an item of group i is supposed to be packed, we are going to take an arbitrary unpacked item from this group. Since it is smaller than its rounded version, it must fit into the union of spaces that are allocated to its parts. The $\lceil \frac{n}{p} \rceil$ items in group 1 are packed into dedicated separate bins. This requires at most $(\frac{n}{p}+1)/\varepsilon \le \frac{2n}{p\varepsilon}$ extra bins, using $p \le n$. □

2.5 An EPTAS for $k = 2$

2.5.1 The structure of the optimal packing

In this section we deal with I'' and $k = 2$. Recall that n denotes the number of items in I' and it is assumed that $n > 1/\varepsilon^{10}$. The number of items in I'' is at most n.

We first make some observations regarding optimal packings for arbitrary inputs. 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, then there is an edge between these items. A bin containing a single part of an item corresponds to a loop on that item in the graph. It was shown in [4] that for any given packing, it is possible to modify it such that there are no cycles in the associated graph. Thus the graph representing the packing consists of a forest together with some loops.

While the graph which represents an optimal packing for the modified input I'' consists of a forest and some loops, the trees of this forest 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 subtrees (with loops) of constant size. Denote by $\text{OPT}'(I'')$ an optimal solution (as well as its cost) for the case where there is an additional constraint that all trees that are created in the packing must have size of at most $1/\varepsilon^2$. We then have the following lemma.

Lemma 7 $\text{OPT}'(I'') \le (1+2\varepsilon)\text{OPT}(I'')$.

Proof A centroid in a tree of size F is defined as a node which can be taken as root such that no subtree has size more than $F/2$. Assume that an optimal packing of I'' contains at least one tree of size larger than $1/\varepsilon^2$ (otherwise we are done). As long as a tree of size larger than $\frac{1}{\varepsilon^2}$ exists, choose such a tree and remove a centroid along with all its outgoing edges and loops. Every item under concern, which was removed as a centroid of some tree, is packed into dedicated new bins. Each time that a centroid is removed, the resulting trees have a size which is at most half the size of the previous tree, and thus the process terminates after a finite number of steps.

Consider a specific tree of size $S > \frac{1}{\varepsilon^2}$. We calculate the number of centroids removed from this tree and from subtrees resulting from it. For $i = \lceil \log_2 S\varepsilon^2 \rceil, \ldots, 1$, we count the number of trees ever considered, from which a centroid was removed (trees created in this process and the original tree), which have a size in $(2^{i-1}/\varepsilon^2, 2^i/\varepsilon^2]$. Since each node participates in at most one such tree, then for a given i, the number of such trees is smaller than $S\varepsilon^2/2^{i-1}$. This gives a total of at most $2S\varepsilon^2 - 1$ centroids. Moreover, no item in I'' has size greater than $1/\varepsilon$, so each centroid requires at most $\frac{1}{\varepsilon}$ dedicated bins, and it results in a tree of size 1. The total number of bins required to pack all the centroids is therefore at most $2S\varepsilon - \frac{1}{\varepsilon}$ for an original tree of size $S > 1/\varepsilon^2$. A tree of size S has at least $S-1$ edges, so the original number of bins used for this tree is at least $S-1$. The number of bins used for the centroids removed from this tree is no larger than $2\varepsilon(S-1)$.

Thus the number of additional bins to decompose the trees into trees of size at most $1/\varepsilon^2$ is at most $2\varepsilon\text{OPT}(I'')$. This proves the lemma. □

2.5.2 Description of the EPTAS

Using Lemma 7, we focus on solutions with trees of size at most $1/\varepsilon^2$.

From a tree to a packing of I'' Assume that a tree representation of a packing $\text{OPT}'(I'')$, with all trees having size at most $1/\varepsilon^2$, is given. Moreover, assume that the tree contains only the edges between distinct items while loops are not specified. Assume furthermore that the trees in $\text{OPT}'(I'')$ are *minimal* in the sense that any partition of the items in a tree into two sets, which are packed independently, requires a strictly larger number of bins to pack the items than the number of bins implied by the original tree.

We pack the items into bins starting from the leaves. Each leaf is removed from the tree after the corresponding item has been fully packed into bins (see Figure 1 for an example). We keep track of which item is supposed to be packed along with this leaf item in its final bin, by assigning the item to that bin. In each step, we pack the item of the selected leaf into bins, starting with the bins that it was assigned to in previous steps, if any (filling those bins completely), and opening new bins if necessary (due to the minimality assumption, this is in fact always necessary until we reach the root).

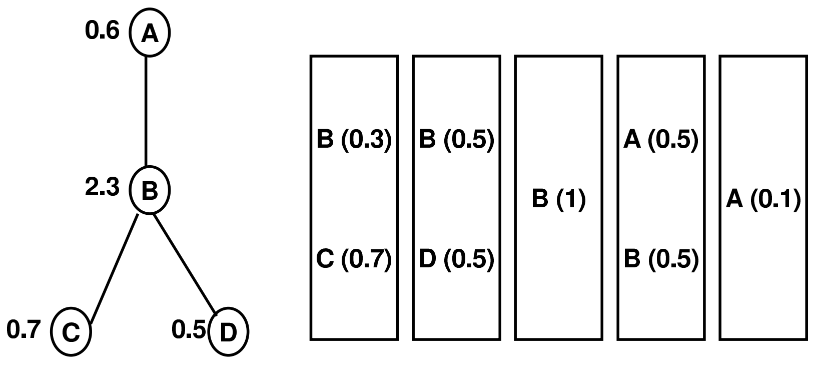

Figure 1: A tree where for each node the number next to it is the size of the item. The corresponding bins are created from left to right from bottom to top where in each bin the content is shown as a list of items and the size of the item which is packed in this bin. The packing algorithm processes the items in the order C, D, B, A.

This proves the following lemma.

Lemma 8 *Given a tree representation of a feasible packing with minimal trees, it is possible to pack the items into bins such that for each tree, there is at most one bin which is not completely full.*

Patterns for a tree respresentation We define a type of a tree to be a pair (j, E) where j is the number of nodes ($1 \leq j \leq 1/\varepsilon^2$). We assume that these nodes are always numbered $1, \ldots, j$ 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, which is a number between 2 and p (recall that I'' does not contain a group of index 1). We next show that the number of patterns is a constant (which depends on ε). The number of trees with at most $1/\varepsilon^2$ nodes can be computed as follows. For a given number of nodes j, the number of different trees is at most the number of ways to choose $j-1$ edges (some of these ways do not result in a tree), that is, at most $\binom{\frac{j(j-1)}{2}}{j} \leq (j^2)^j \leq (\frac{1}{\varepsilon^4})^{\frac{1}{\varepsilon^2}}$. This gives at most $\frac{1}{\varepsilon^2} \cdot (\frac{1}{\varepsilon^4})^{\frac{1}{\varepsilon^2}}$ trees. There are $p-1$ possible groups for each node, so there are at most $\frac{1}{\varepsilon^2} \cdot (\frac{1}{\varepsilon^4})^{\frac{1}{\varepsilon^2}} \cdot p^{\frac{1}{\varepsilon^2}}$ patterns. For a given pattern, the process of packing is defined as above. A pattern is *valid* if the packing process applied on it (that is, on the item sizes defined by its vector) leads to all items being packed into a single tree (without splitting the tree, which would imply that the tree is not minimal), and without violating any cardinality constraints. That is, if the representation of the resulting packing is exactly the original tree.

Finding an approximate solution Recall that there is a constant number of patterns. For each one of them, we check its validity (including minimality) in constant time, resulting in a list of valid patterns. Thus we get a constant size set of valid patterns which we denote by $\mathcal{P}$. For a valid pattern $q \in \mathcal{P}$ we denote by $n(g, q)$ the number of items of group g which are packed in q. For every group g, we denote by $n(g)$ the number of items in this group in I''. For a pattern $q \in \mathcal{P}$, we denote by $size_q$ the number of bins which are needed to pack the items in pattern q, that is, $size_q$ is the smallest integer which is at least the total size of the items in the pattern q. We next formulate a linear program where each variable z_q (for $q \in \mathcal{P}$) indicates the number of times we use a pattern q. The role of the constraints is to verify that all items of groups $2, 3, \ldots, p$ are packed.

$$\begin{aligned} \min \sum_{q \in \mathcal{P}} size_q z_q & \\ s.t. \sum_{q \in \mathcal{P}} n(g, q) z_q \geq n(g) & \quad \forall g = 2, 3, \ldots, p \\ z_q \geq 0 & \quad \forall q \in \mathcal{P}. \end{aligned}$$

Consider the optimal solution to this linear program and without loss of generality assume that it is a basic solution. Denote it by $(z_q^*)_{q \in \mathcal{P}}$. The number of non-integer components in the solution is at most the number of constraints (neglecting the non-negativity constraints), i.e., at most $p-1$. We pick $\lfloor z_q^* \rfloor$ times the pattern q, and as a result, we are left with at most $\frac{p-1}{\varepsilon^2}$ items which are not completely packed; these are the items whose packing is done in the fractional part of the basic solution to the linear program, at most $\frac{1}{\varepsilon^2}$ items for each pattern q for which z_q^* is not an integer. Packing these items in separate bins costs us at most $\frac{p}{\varepsilon^3}$ additional bins since every item has size at most $\frac{1}{\varepsilon}$. Since the cost of z^* is at most $\text{OPT}'(I'') \leq (1+2\varepsilon)\text{OPT}(I'') \leq (1+2\varepsilon)\text{OPT}(I')$ (using Lemma 5 and Lemma 7), we conclude that the resulting feasible solution to I costs at most $(1+2\varepsilon)\text{OPT}(I') + \frac{2n}{p\varepsilon} + \frac{p}{\varepsilon^3}$ by Lemma 6.

Our EPTAS is summarized in Figure 2. Recall that $p = \frac{1}{\varepsilon^4}$, so the solution obtained by our algorithm costs at most

$$\begin{aligned} (1+2\varepsilon)\text{OPT}(I') + \frac{2n}{p\varepsilon} + \frac{p}{\varepsilon^3} &= (1+2\varepsilon)\text{OPT}(I') + 2n\varepsilon^3 + \frac{1}{\varepsilon^7} \leq (1+2\varepsilon)\text{OPT}(I') + 3\varepsilon n \\ &\leq (1+8\varepsilon)\text{OPT}(I') \leq (1+10\varepsilon)\text{OPT}(I), \end{aligned}$$

where we have applied the definition of p, $n \geq \frac{1}{\varepsilon^{10}}$, $\text{OPT}(I') \geq \frac{n}{2}$, and Lemma 2 in this order, as well as $\varepsilon \leq \frac{1}{20}$.

Input: I.

1. Apply the modification of Section 2.1 to get the input I'. That is, for every item X of size $x > 1/\varepsilon$, replace it with $\lfloor \varepsilon x \rfloor$ items of size $1/\varepsilon$ and possibly one additional item of size $x - \frac{1}{\varepsilon}\lfloor \varepsilon x \rfloor$, if $\lfloor \varepsilon x \rfloor < \varepsilon x$.
2. If $k \geq \varepsilon \cdot n$ or $n \leq \frac{1}{\varepsilon^{10}}$, apply the algorithm described in Lemma 3.
3. Else, if $k > \frac{1}{\varepsilon^2}$, apply the algorithm in Lemma 4.
4. If no algorithm was applied so far, apply the modification of Section 2.4 to get the input I''. That is, put the items into a collection of $p = \frac{1}{\varepsilon^4}$ groups, remove the items of the first (largest) group and round up the item sizes as in Fernandez de la Vega and Lueker [5]. All items in a group have the same size.
5. Apply the following steps. For $k = 2$ the steps are defined in Section 2.5 and for the case $2 < k \leq \frac{1}{\varepsilon^2}$ they are defined in Section 2.6.
   (a) Determine the set of valid patterns (trees plus specification of groups of nodes).
   (b) Solve a linear program for determining the forest (combination of patterns) which uses the least number of bins and which packs all the items.
   (c) Round down the fractional solution to the linear program.
   (d) Assign the items of I'' according to the rounded solution. Pack the remaining items into dedicated bins.
   (e) Convert the tree representation into a packing into bins.
6. If the algorithm was applied on I'', use Lemma 6 to convert it into a packing of I'. That is, replace the rounded items by the original items and pack remaining items using separate bins.

Output: the packing of I' (which is also a packing of I).

Figure 2: The EPTAS.

Theorem 1 *There exists an efficient 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.*

Proof The approximation ratio follows by the above argument. We next consider the time complexity of the scheme for the case $n > \frac{1}{\varepsilon^{10}}$ and $W \leq n^2$, since otherwise we can calculate an exact solution in strongly polynomial time by Lemmas 1 and 3. Using $W \leq n^2$, the pre-processing of step 1 is done in $O(n^2)$. Therefore, the number of items in I' is also $O(n^2)$. The linear grouping in step 4 takes a time linear in the number of items in I'. The time complexity of the remaining steps is clearly polynomial in the number of valid patterns (trees plus specification of groups of nodes). As mentioned above, the number of patterns is no larger than $\frac{1}{\varepsilon^2} \cdot \left(\frac{1}{\varepsilon^4}\right)^{\frac{1}{\varepsilon^2}} \cdot p^{\frac{1}{\varepsilon^2}}$. Since $p = \frac{1}{\varepsilon^4}$, we conclude that the number of patterns is a function of $\frac{1}{\varepsilon}$. Therefore, the time complexity of steps 5(b)–5(e) is a constant which is some exponential function of $\frac{1}{\varepsilon}$. The claim holds since step 6 takes time which is linear in the number of items in I'. $\square$

2.6 An EPTAS for $3 \le k \le \frac{1}{\varepsilon^2}$

In this section we present an approximation scheme for $3 \le k \le \frac{1}{\varepsilon^2}$. In this section we deal with the input I''.

Our EPTAS for constant k is similar to the one for $k = 2$, however, the major difference is that we need to implement Step 5(a) for this more general case. For this purpose we use a modified graph representation. If a bin contains parts of the items $Y_1, \ldots, Y_k$, then we order them in some way and create edges only between successive items in this ordering. Thus the edges in this case, if the items are ordered as above, are $(Y_1, Y_2), \ldots, (Y_{k-1}, Y_k)$. Each item is still represented by a single node, thus one node might be involved in a number of such chains, where each chain represents one bin.

In this case we no longer have a one-to-one correspondence of edges and bins. Instead we have the property that there are at least as many edges as there are bins. Note that the order of the parts inside a bin has no meaning and we can reorder the parts in a chain arbitrarily. This may result in a different graph representation for the same packing into bins. It is now more difficult to construct a packing into bins from a given graph. Before we proceed, we prove several important properties.

Lemma 9 *There exists an optimal packing whose corresponding graph representation does not contain any cycles.*

Proof We use a repacking process similar to the one used in the proof of Lemma 1 from [4]. Consider a representation in which the parts of items in every bin are ordered in an increasing order of indices. The packing which we consider is one with a minimal number of edges, that is, where the total number of parts is minimal. Assume by contradiction that there is a cycle in the associated graph. Consider the sequence of items in a cycle. The cycle must contain edges of at least two bins. We temporarily replace every maximal sub-path of the cycle, whose edges correspond all to one bin, by a single edge, thus removing some items from the sequence. Denote the resulting sequence by X_0,X_1,…,X_{t-1},$X_t = X_0$, where $t \ge 2$. Each item X_j, for $0 \le j \le t$, is therefore split into two parts of sizes x_j and x'_j (and possibly, some additional parts), where the part of size x_j is packed with the part of size x'_{j-1}, for $1 \le j \le t$ (since X_0 and X_t are the same item, it holds that $x_0 = x_t$ and $x'_0 = x'_t$). Let $\mu = \min\{\min\limits_{1\le i\le t} x_i, \min\limits_{1\le i\le t} x'_i\}$, i.e., the smallest part out of the considered $2t$ parts. Assume without loss of generality that $x_0 = \mu$. We split each part of size x_i into a part of size μ, and possibly an additional part (in the case $i = 0$, this second part does not exist). The part of size μ of x_i is moved to the bin which contains the item of size x'_i. Since parts of the same size were moved cyclically, no bin exceeds a total size of 1. The packing is valid since each bin already contained a part of the item which was moved into it. No new edges are created. However, as a result, there is a bin, out of which a part of an item was removed completely, and the number of parts of items in this bin is reduced by 1. Thus, in order to represent this bin, the number of edges in the graph representation is reduced by 1, which contradicts minimality. □

We next prove that it is possible to assume again that the number of items in a connected component is upper bounded by a polynomial in $\frac{1}{\varepsilon}$, and it is at most $\frac{1}{\varepsilon^4}$. We now 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 of at most $1/\varepsilon^4$.

Lemma 10 $\text{OPT}'(I'') \le \text{OPT}(I'') + 3n\varepsilon^3$.

Proof We apply a process similar to the one in the proof of Lemma 7, however, if a part of an item (which is a centroid, in our case) is removed from a bin, if this part used to have two edges to items which have parts in the same bin, an edge should be added between these two items, to keep the items which have parts in this bin connected in a path, or otherwise the bin must be split into two bins.

At a time that a centroid has just been removed, and before any edges are added, some number of connected components is created, while the edges that need to be added form a matching between these components (each edge of the matching corresponds to a bin where the part of the removed centroid was neither first nor last in the chain). It is possible to add all edges, except for possibly one such edge, without creating components of size larger than half the size of the original tree. This can be proved as follows. Let ν be the number of connected components after the centroid is removed and let ρ the number of edges in the matching. Clearly, $\nu \geq 2\rho$. If $\rho = 1$ then the claim holds since no edges need to be added. Otherwise, temporarily add all the edges of this matching. This results in $\nu - \rho \geq \rho \geq 2$ connected components. At most one of them has a size which exceeds the required upper bound. The edge of the matching which created this component is removed, resulting in sufficiently small connected components. Since one edge is not added, one additional bin is created by splitting one bin into two bins.

Let us assume that the packing $\text{OPT}(I'')$ contains a tree of size $S > 1/\varepsilon^4$. We repeatedly remove a centroid from large trees, along with all its outgoing edges and loops. We then return edges connecting resulting connected components, except for one edge, as described above, that is, an edge is put between a pair of components, which have a total size of at most $\frac{S}{2}$ and the edges connecting them to the centroid correspond to the same bin. This process is applied on each tree, as long as a tree of size larger than $\frac{1}{\varepsilon^4}$ exists. Every item under concern, which was removed as a centroid of some tree, is packed into dedicated new bins. Each time that a centroid is removed, the size of the resulting trees is at most half the size of the previous tree.

Similarly to the case $k = 2$, if the number of nodes in the original tree is S, then the number of removed centroids (along all iterations) is at most $2S\varepsilon^4$. For all trees, the number of removed centroids is at most $2n\varepsilon^4$. The removal of each centroid causes the creation of at most $\frac{1}{\varepsilon} + 1$ bins (at most $\frac{1}{\varepsilon}$ bins for the centroid, and one additional bin due to one removed edge), so the resulting solution costs at most $\text{OPT}(I'') + 3n\varepsilon^3$. □

From a tree to a packing of I'' We further modify our graph representation, and let each item be represented by υ nodes if and only if it is split into υ parts in the packing. The parts of one item are connected by a simple chain (using *item edges*), as are the parts that are in one bin (using *bin edges*). We can then start packing bins from the leaves of the tree and repeatedly remove leaves similar to before. There are now two cases, depending on whether the edge that connects the leaf to the tree is an item edge or a bin edge.

If it is a bin edge, then the leaf represents the last part of some *item*, which is now packed inside the bins it is assigned to. Also, we assign the item at the other end of this edge to this bin.

If it is an item edge, then the leaf represents the last part that is packed into a particular *bin* (possibly a new bin), which is now filled up (entirely, unless it is the root) by this leaf.

Using this packing process, it can be seen that Lemma 8 also holds for this case. If some item does not fit where it is supposed to, violates the cardinality constraint, or does not fill up a bin that it should, the tree we are considering is not valid. 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, we 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 (α, β) for each node (to get a pattern for the tree). Thus, α is the group ($\alpha \in \{2, 3, \ldots, p\}$, where p is the number of groups as in Section 2.5.2 and β is the number of the item of this group in this tree ($\beta \leq 1/\varepsilon^4$). In a valid tree, the nodes of type (α, β) for any fixed α and β 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 11 *The length of a chain representing one item in a valid pattern is at most* $1/\varepsilon^4 + 1/\varepsilon$.

Proof There can be at most $\frac{1}{\varepsilon^4}$ 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 $\frac{1}{\varepsilon}$ 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 8. The size of an item is at most $\frac{1}{\varepsilon}$. □

Let $g(\frac{1}{\varepsilon})$ be the maximum number of nodes in such a tree, then the number of different tree topologies on at most $g(\frac{1}{\varepsilon})$ nodes is again a constant denoted by $f(\frac{1}{\varepsilon})$. Hence, the number of patterns is bounded by $f(\frac{1}{\varepsilon}) \cdot \left(\frac{p}{\varepsilon^4}\right)^{g(\frac{1}{\varepsilon})}$, and so it is a constant since $p = \frac{1}{\varepsilon^4}$.

We can now construct in polynomial time a linear program which is the same as in the previous EPTAS (with the difference that the notion of feasible patterns $\mathcal{P}$ is now different). Again, we solve this linear program and round down the resulting fractional basic solution. Then we pack each remaining item in its own set of bins. The analysis of the approximation ratio follows the same lines as the case $k = 2$. Using the value of p, Lemma 10, Lemma 5, $n > \frac{1}{\varepsilon^{10}}$, $\text{OPT} \geq n\varepsilon^2$ (see (2)), Lemma 2 and $\varepsilon \leq \frac{1}{20}$, (applied in this order) we get the following inequalities:

$$\text{OPT}'(I'') + \frac{2n}{p\varepsilon} + \frac{2p}{\varepsilon^3} \leq \text{OPT}(I'') + 5n\varepsilon^3 + \frac{2}{\varepsilon^7} \leq \text{OPT}(I') + 7\varepsilon^3 n$$

$$\leq (1 + 7\varepsilon)\text{OPT}(I') \leq (1 + 9\varepsilon)\text{OPT}(I),$$

and therefore the resulting scheme is indeed a PTAS. Note that the time complexity is again of the form $F(1/\varepsilon) \cdot \text{POLY}(\text{LENGTH}(I))$, and hence we got an EPTAS for the values of k considered in this section. Thus in Section 2 we have established the following theorem.

Theorem 2 *For any $k = o(n)$, there exists an efficient 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.*

3 Dual approximation schemes

3.1 A dual PTAS for $k = 2$

In this section we present approximation schemes where the number of bins is no larger than in an optimal solution, while the bins are slightly larger. Thus, the modifications of I into I' and I'' from before cannot be used, and we act on the input I. As discussed in Section 2, an optimal packing can be represented by a graph which is a forest together with some loops. In this section, an item is called *small*, if its size is at most $\frac{1}{2}$. An item of size in $(\frac{1}{2}, 1]$ is called *medium*. All other items (i.e., items of size strictly above 1) are called *large*. We may still assume $W \leq n^2$ by Lemma 1.

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 12 and 13 below, no further cuts are required for items of size at most 1.

Lemma 12 *There exists an optimal packing represented by a forest in which all items of size at most 1/2 are leaves.*

Proof Consider a packing represented by a forest where the sum of degrees a of small items is minimal, and assume by contradiction that there is a non-leaf small item B. Note that if two small items are adjacent, but at least one of them is not a leaf, then the packing can be changed so that these two items form a separate connected component, which is a tree with two nodes and a single edge. This would increase the number of small items which are leaves, and thus would lead to a contradiction. Thus we may assume that all neighbors of B are medium or large items.

Consider two arbitrary neighbors A_1 and A_2 of B. Denote the corresponding bins by 1 and 2, and denote the sizes of the corresponding neighboring parts of B by a_1 and a_2, and the parts of B by b_1 and b_2. We have $b_1 + b_2 \leq 1/2$. If $b_1 \leq a_2$ (see figure 3 for illustration), we cut off a part of size b_1 from the part of size a_2 and put it in bin 1, while putting the part of size b_1 in bin 2. This removes the neighbor A_1 from the small item B. A new edge is created between A_1 and A_2, none of which is small. Since the edge between B and A_1 is removed, a cycle cannot be created.

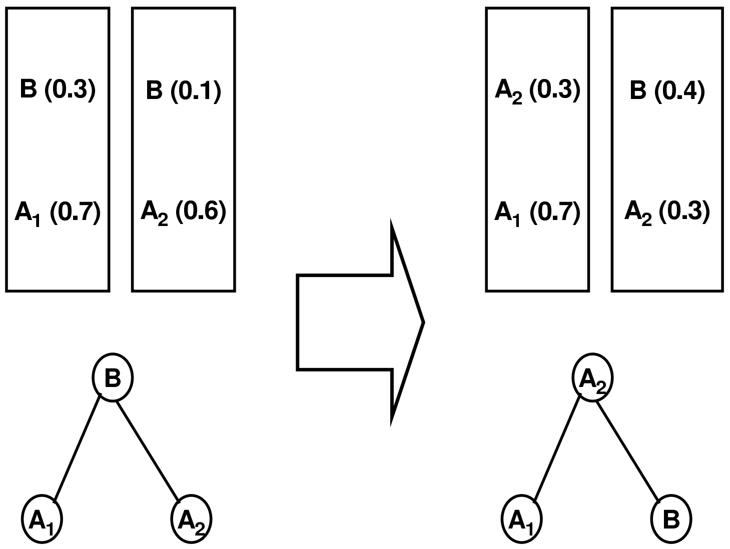

Figure 3: The case $b_1 \leq a_2$ in the proof of Lemma 12. On the left side the packing of the items and the corresponding tree before the change, and on the right side the packing and the tree after the change.

Otherwise, $a_2 < b_1 \leq 1/2$ (see figure 4 for illustration), which means that we can put b_1 into bin 2 without taking anything out of bin 2: we have $a_2 < 1/2$ and $b_1 + b_2 \leq 1/2$. Again, A_1 is no longer a neighbor of B. Moreover, no new edges are created.

Thus we successfully removed one neighbor from B, keeping the forest structure. The degree of B was reduced by 1 while no other degrees of small items were changed. This results in an optimal packing with a smaller sum of degrees of small items, which is a contradiction. □

The next lemma can be seen as a generalization of Lemma 12.

Lemma 13 *There exists an optimal packing which is represented by a forest in which any item of size in* $((i-1)/2, i/2]$ *has at most* i *neighbors for all* $i \geq 2$.

Proof For a given tree, we root it at some item. We need to modify the packing of every item, for which the number of its parts is too large. These items are considered level by level, starting from the root (considered as the highest level). We apply an iterative process where items are being repacked without increasing the number of bins used.

Thus, assuming that all items of previous levels are packed into a small enough number of bins, consider an item X of size at most $i/2$, which is packed into $i' > i$ bins. Neglecting the bin that contains its parent, i.e., the uplevel item (if any), there are at least i bins into which the item is packed. Consider the two bins among them (1 and 2) with the two smallest parts of X, of sizes x_1 and x_2 such that

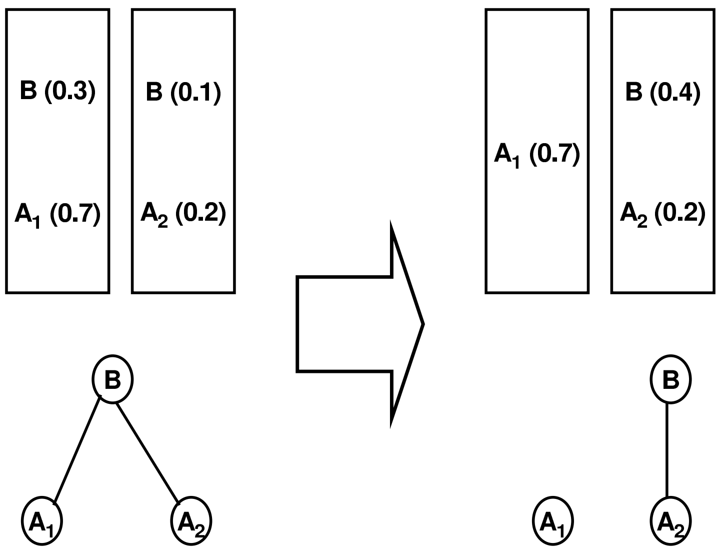

Figure 4: The case $b_1 > a_2$ in the proof of Lemma 12. On the left side the packing of the items and the corresponding tree before the change, and on the right side the packing and the tree after the change.

$x_1 \le x_2$. By the size constraint on X, we have

$$x_1 + (i-1)x_2 \le i/2 \ .$$

Let $x_1 = \frac{1}{2} - \gamma$. We get $\frac{1}{2} - \gamma \le x_2 \le \frac{1}{2} + \frac{\gamma}{i-1}$. Thus $\gamma \ge 0$ and $x_2 \le \frac{1}{2} + \gamma$, since $i \ge 2$. This gives $x_1 + x_2 \le 1$.

Let the total size of parts of items in the second bin, including the part of size x_2 be $g \le 1$. If any of the two parts of sizes x_1 and x_2 is packed alone, the second part can be moved to join it, since their total size is at most 1. Else, if $x_1 + g - 1 \le 0$, then moving the part of size x_1 into bin 2 results in a total size of $g + x_1 \le 1$. Otherwise, the part of an item Y, packed with the part of size x_2 has a size of $g - x_2 \ge g - \frac{1}{2} - \gamma = g - 1 + x_1$. We can split Y into a part of size $x_1 + g - 1$ and possibly another part, and swap the part of size $x_1 + g - 1$ with x_1 in the first bin. This reduces the number of bins that contain X and gives a valid packing (with no more than two parts in any bin). Note that the item of which Y is a part, may receive a new neighbor, though the connected component remains a tree. The two items whose parts were packed with the parts of sizes x_1 and x_2 of X are treated later if necessary, since they are items of a lower level of the tree. $\square$

Thus in particular, each medium item has at most two neighbors in the tree corresponding to an optimal packing. 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 “no cut”.

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 which 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, and a rounding on the sizes is applied. We later use the fact that we can slightly increase the sizes of bins in order to efficiently enumerate the possible cutting points.

We next 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.

Lemma 14 *It is possible to modify the input in such a way that an 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 an 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 13. 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 the new set, it could also have been used for the original instance. In the packing P for the modified input, each newly created item is a leaf. $\square$

Lemma 15 *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 (see Figure 5 for an illustration), 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 and the number of loops of X equals the part that should be cut off in order to split the node of the large item into two nodes, one which 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 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 the other part is a leaf in the original tree, each large item is cut at most once. $\square$

We conclude that by modifying the input appropriately, there exists an optimal packing which consists of stars with large items in the root (where such a large item which 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.

3.1.1 Description of the algorithm

Our dual PTAS works as follows. It is summarized in Figure 6. We use a parameter δ which is based on ε, and which is the inverse of some odd integer. Specifically, we let $K = \frac{2}{\varepsilon} + 1$ and $\delta = 1/K$. We begin by rounding item sizes (of all items that are not large) up to the nearest multiple of δ. 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 that contains no small items 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 $i = (K+1)/2, \ldots, K$. For each size of a medium item, we guess how many items of this size are cut at each integer multiple of δ which is at most $1/2$.

Denote the number of large items by L. For convenience of notation, we will also denote this number by $M_{(K-1)/2}$ (the index of $M_{(K-1)/2}$ will be used later in a similar way to the indices of M_i for $i = (K+1)/2, \ldots, K$). We guess how many pieces of each size of at most 1, which is an integer

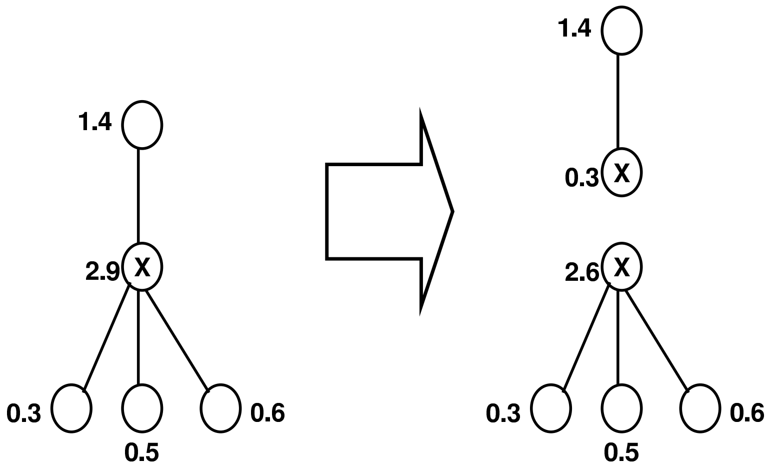

Figure 5: An illustration for the proof of Lemma 15. On the left side the tree before the change, and on the right side the tree after the change.

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 $(K+1)\delta/2, \ldots, K\delta$ are cut off at each point, and how many pieces of each size are cut off from the large items. We will cut each large item at most once, and then find a forest consisting of stars, and then the final packing of the items allows each large item to be cut further.

Construction of the graph (Figure 7) For every possible set of cuts, 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 $i\delta$ $(i = 0, \ldots, K)$ (including parts of large items!). 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.

For two nonnegative integer vectors a and b of length ℓ, we say that $a \geq b$ if $a_i \geq b_i$ for $i = 1, \ldots, \ell$. We say that $a \to b$ if there exists a unique j such that $a_j = b_j + 1$ and $a_i = b_i$ for $i \in \{1, \ldots, \ell\} \backslash \{j\}$.

The cost of an edge (u, v) that is mentioned in Step 4 of Figure 7 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

1. Let $K = \frac{2}{\varepsilon} + 1$ 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 $i = (K+1)/2, \ldots, K$.
3. For each medium item size, guess how many items of this size are cut at $j\delta$ for $j = 0, \ldots, (K-1)/2$.
4. Guess how many items of size $j\delta$ are cut off from large items for $j = 0, \ldots, K$.
5. Create a graph with L layers, plus source and sink. The construction of the graph is shown in Figure 7. 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 6: The dual PTAS for $k = 2$

we denote the sizes of items specified by $s(u) - s(v)$ by $a_1, \ldots, a_p$ and the size of the part of the large item that needs to be packed by X, then the number of bins is $\max\{p, \left\lceil \frac{X + \sum_{i=1}^{p} a_i}{1+\delta} \right\rceil\}$.

The cost of an edge (u, v) that is mentioned in Step 5 of Figure 7 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 of size $1 + 2\delta$ containing one or two of these items. We apply the First Fit Decreasing (FFD) algorithm with the restriction that no bin can contain more than two items. By Lemma 16, this gives an optimal packing.

Lemma 16 *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 [20] that for such an input, FFD gives an optimal solution. □

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 $k_1\delta, k_2\delta, \ldots, k_{a_1}\delta$. 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.

Lemma 17 *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 $(K + 1)^2/4 + K + 1$.

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. Define a fixed ordering on the large items. 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 ($i = 0, \ldots, L-1$), there is an arc to every node v in layer $i+1$ such that $c(u) \to c(v)$ and $s(u) \geq s(v)$.

4. The cost of arc (u, v), where u is in layer i ($i = 0, \ldots, L-1$), is the cost of packing the ith 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 7: Construction of the layered graph for one set of guesses (cuts)

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 $\prod_i (M_i + K - 1) \leq (\frac{2n}{K+3} + K - 1)^{(K+3)/2}$ which means we have at most $(\frac{2n}{K+3} + K - 1)^{(K+3)^2/4}$ 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

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

□

Lemma 18 *This algorithm uses at most* $\text{OPT}(I)$ *bins of size* $1+2\delta$ *to pack the input* I.

Proof An optimal solution of the original instance (in bins of size 1) can be adapted to pack the rounded items 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. □

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 packs the items in the optimal number of bins, but uses bins of size* $1+\varepsilon$.

3.2 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 $\delta=\frac{\varepsilon}{k}$. 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 9). We modify the input in two steps.

Sizes of items and parts. A first step will be a revision of sizes of items and parts of items. We take an optimal packing, and replace any item of size x with an item of size $\lceil\frac{x}{\delta}\rceil\delta$. Specifically, we consider all of its parts one by one (in some order) and round each part up or down to the nearest multiple of δ, maintaining the invariant that the total new size x_i' of all parts considered so far is at least the total original size x_i, and at most $\lceil\frac{x_i}{\delta}\rceil\delta$. As a result, the total size of parts in any one bin can increase by at most $k\delta$. All parts in the packing now have sizes that are multiples of δ. Note that it could happen that the number of bins used decreases, if there are bins where all the pieces in it have their size reduced to 0. We use bins of size $1+k\delta$. Denote the resulting rounded instance by I'. In I' we allow an algorithm to use bins of size $1+k\delta$ but we force it to cut items only in integer multiple of δ. Therefore, we showed that

$$\text{OPT}(I')\le\text{OPT}(I).$$

Large items. As in the previous Section (Lemma 15), we would like to pack the large items one by one and not combine them together into bins. Note that we showed in Section 3.1 that Lemma 8 still holds in this case. We have the following lemma.

Lemma 19 *It is possible to modify the input in such a way that the optimal packing for the new input requires* $\text{OPT}(I')$ *bins, and there exists an optimal packing for the new input such that each tree contains at most one large item.*

Proof Consider the set of items S resulting from $\text{OPT}(I')$, in which we replace each non-large item by the set of parts (of it) as in $\text{OPT}(I')$. We consider the tree representation of the optimal packing of S such that each item of S is represented in a single node of the tree. That is, each large item is represented by a single node (but the non-large items are already cut), and each edge connects two items sharing a bin. We can now apply the proof of Lemma 15, using the value $1 + k\delta$ instead of 1 as bin size and as lower bound for the size of a large item. $\square$

Thus we find that for each large item, it is sufficient to cut off one part of size at most $1 + k\delta$ in order to pack them into separate trees, and moreover this part does not need to be cut further later.

Non-large items. We now consider the non-large items (of 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 $\text{OPT}(I')$ all parts have sizes that are integer multiples of δ. The number of cuts for each item is therefore at most $M - 1$.

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 I, 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 I'.

After rounding, the non-large items in the input can be represented by a vector with M components whose ith component indicates how many items of size $i\delta$ exist. 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 between the packing vectors and by the size of the large item of the current layer.

Guess vectors We construct a guess vector with at most $M(2 + 2^{M-1})$ entries. For each non-large size, there are at most $2 + 2^{M-1}$ entries. We have a first entry which is the number 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 numbers of items of this size that are cut according to a given pattern. There are 2^{M-1} options for the cut set of each item. Therefore there are at most 2^{M-1} 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 I' 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 $(n + 1)^{M(2+2^{M-1})}$ 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 $i = 1, \ldots, M + 1$. 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 interchangable 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 jth 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 $i_j = 0$ up to 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 (that is a function of k and ε).

Guess packings In order to be able to use a layered graph, we will 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 nM, since each non-large item is cut into at most M 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 $(nM+1)^T$. Since no bin is empty in the packing, the total number of bins which involve packing of parts of non-large items, is at most nM. Therefore this number is polynomial in n for a fixed constant value of 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 nodes of each layer are vectors that are smaller than the final vector (i.e., including the cutoff vector). Layer zero has a single node 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 nonzero 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 $z' \le b$, 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 $\left\lceil \frac{z'-b}{1+k\delta} \right\rceil$. 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. Completely analogously to the proof of Lemma 18, it can be shown that we find a solution with the optimal number of bins.

4 Conclusions

In this paper, we provided approximation schemes for bin packing of splittable items with cardinality constraints for almost all values of k. We provided dual approximation schemes as well. It should be noted that our upper bounds are absolute, i.e. there is no additive term in the definition of the approximation ratio. We leave an interesting open problem which is to develop a dual PTAS for non-constant values of k.

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] Marco Cesati and Luca Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64(4):165–171, 1997.

[4] 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.

[5] 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.

[6] András Frank and Éva Tardos. An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica* 7(1): 49-65, 1987.

[7] Leah Epstein. Online bin packing with cardinality constraints. *SIAM Journal on Discrete Mathematics*, 20(4):1015–1030, 2006.

[8] Leah Epstein and Asaf Levin. AFPTAS results for common variants of bin packing: A new method to handle the small items. *CoRR*, abs/0906.5050, 2009.

[9] Leah Epstein and Rob van Stee. Approximation schemes for packing splittable items with cardinality constraints. In *Proc. of the 5th International Workshop on Approximation and Online Algorithms (WAOA2007)*, pages 232–245, 2007.

[10] Leah Epstein and Rob van Stee. Improved results for a memory allocation problem. In *Proc. of the 10th International Workshop on Algorithms and Data Structures (WADS2007)*, pages 362–373, 2007. Also in Theory of Computing Systems, to appear.

[11] 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.

[12] Ronald L. Graham and Jia Mao. Parallel resource allocation of splittable items with cardinality constraints. Preprint, 2006.

[13] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.

[14] 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.

[15] Hans Kellerer and Ulrich Pferschy. Cardinality constrained bin-packing problems. *Annals of Operations Research*, 92:335–348, 1999.

[16] 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.

[17] 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.

[18] Hadas Shachnai, Tami Tamir, and Omer Yehezkely. Approximation schemes for packing with item fragmentation. *Theory of Computing Systems*, 43(1):81–98, 2008.

[19] Hadas Shachnai and Omer Yehezkely. Fast asymptotic FPTAS for packing fragmentable items with costs. In *Proc. of the 16th International Symposium on Fundamentals of Computation Theory (FCT2007)*, pages 482–493, 2007.

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