An APTAS for generalized cost variable sized bin packing

Leah Epstein* Asaf Levin †

Abstract

Bin packing is a well known problem which has a large number of applications. Classical bin packing is a simple model where all bins are identical. In the bin packing problem with variable sized bins, we are given a supply of a variety of sizes. This latter model assumes, however, that the cost of a bin is always defined to be its exact size.

In this paper we study the more general problem where an available bin size is associated with a fixed cost, which may be smaller or larger than its size. The costs of different bin sizes are unrelated. This generalized problem has various applications in storage and scheduling. In order to generalize previous work, we design new rounding and allocation methods. Our main result is an APTAS for the generalized problem.

1 Introduction

Bin packing is a natural and well studied problem which has applications in computer storage, bandwidth allocation, stock cutting, transportation, and many other important fields. The study of bin packing started more than thirty years ago [5, 7]. Since then a large amount of research was dedicated to this problem and its variants (see e.g. [1, 3, 10]).

An interesting variant of this problem is VARIABLE SIZED BIN PACKING, where the supply of containers is not only of a single bin type, but some fixed (finite) number of given sizes is available. The cost of using a bin is simply its size. The first to investigate the variable sized bin packing problem were Friesen and Langston [4]. Several papers studied this problem in offline and online environments [12, 2, 13, 14].

We consider the following variant of one dimensional bin packing, which is a natural generalization of both classical bin packing and variable sized bin packing. We are given an infinite supply of bins of r types whose sizes are denoted by $b_r < \cdots < b_1 = 1$. We denote $B = \{b_1, \ldots, b_r\}$. Items of sizes in $(0, 1]$ are to be partitioned into subsets. The set of items is denoted S, and the items have indices in the set $\{1, 2, \ldots, n\}$. The size of item j is denoted by s_j. Each subset J in the partition has to be assigned (packed) to some bin type i, such that the set of items fits into an instance of this bin type, i.e., $\sum_{j \in J} s_j \leq b_i$. A bin type i is associated with a cost c_i. We assume $c_1 = 1$. Thus, the cost of a solution is the sum of costs of the bins used, taking multiple subsets which are using the same bin type into account. That is, if the subsets are $J_1, \ldots, J_k$, and subset ℓ is packed into a bin of type i_ℓ (for $1 \leq \ell \leq k$), we get the cost $\sum_{\ell=1}^{k} c_{i_\ell}$. The goal is to find a feasible solution whose total cost is minimized. Without loss of generality we let $b_{r+1} = c_{r+1} = 0$. We call this problem GENERALIZED COST VARIABLE SIZED BIN PACKING and denote it by GCVS.

This problem clearly generalizes the classical bin packing problem (where $B = \{b_1\}$), and variable sized bin packing (where $c_i = b_i$ for $1 \leq i \leq r$). Classical bin packing assumes a very simple model with

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

†Department of Statistics, The Hebrew University, Jerusalem, Israel. levinas@mscc.huji.ac.il.

uniform bin sizes. Although a large number of real-world problems can indeed be defined as such a problem, an even larger amount of problems involve bins of various sizes. This started the study of variable sized bin packing. To simplify reality, such models usually assume that the cost per unit of storage area is constant. Once again, this is not always the case in real life, as can be easily seen from prices of memory sticks and portable hard disks. Moreover, typically a smaller container has a larger cost per unit of storage, as it is the case with various storage devices nowadays. On the other hand, we sometimes encounter a phenomenon where we find that due to technological barriers, the cost of a memory storage device with a twice as large capacity, is more than twice the cost of the more modest device.

These are just examples to the fact that in reality, the cost of storage containers just cannot be assumed to be linear in their sizes. Previous studies of generalized costs functions assumed that the price per unit decreases as the bin size grows. Kang and Park [9] who studied a generalized problem with cost functions satisfying $\frac{c_i}{b_i} \leq \frac{c_j}{b_j}$ for $i > j$, suggested an algorithm of asymptotic approximation ratio $\frac{3}{2}$. Another possibly reasonable assumption is a concave cost function (see [11]). From the scenarios stated above, showing that pricing policies can be arbitrary, we deduce that neither options describe the typical real situation. This leads to the study of general cost functions.

It is known that no approximation algorithm for the classical bin packing problem can have a cost within a constant factor r of the minimum number of required bins for $r < \frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$. This leads to the usage of the standard quality measure for the performance of bin packing algorithms which is the *asymptotic approximation ratio* or *asymptotic performance guarantee*. For an algorithm $\mathcal{A}$, we denote its cost on an input $\mathcal{X}$ by $\mathcal{A}(\mathcal{X})$. An optimal algorithm is denoted by OPT, and its cost of input $\mathcal{X}$ is denoted by $\text{OPT}(\mathcal{X})$.

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

$$\mathcal{R}(\mathcal{A}) = \limsup_{n\to\infty} \sup_{\mathcal{X}} \left\{ \frac{\mathcal{A}(\mathcal{X})}{\text{OPT}(\mathcal{X})} \middle| \text{OPT}(\mathcal{X}) = n \right\}.$$

The natural question, which was whether this measure allows to find an approximation scheme for classical bin packing, was answered affirmatively by Fernandez de la Vega and Lueker [3]. They designed an algorithm whose output never exceeds $(1+\varepsilon)\text{OPT}(I) + f(\varepsilon)$ bins for an input I and a given $\varepsilon > 0$. The running time was linear in n, but depended exponentially on ε. Such a class of algorithms is considered to be an APTAS (Asymptotic Polynomial Time Approximation Scheme).

Karmarkar and Karp [10] developed an AFPTAS (Asymptotic Fully Polynomial Time Approximation Scheme) for the same problem. This means that using a similar (but much more complex) algorithm, it is possible to achieve a running time which depends on $\frac{1}{\varepsilon}$ polynomially, without any loss in the approximation ratio. Karmarkar and Karp [10] also designed an algorithm which uses at most $\text{OPT}(I) + \log^2[\text{OPT}(I)]$ bins for an input I.

Murgolo [12] designed an APTAS and an AFPTAS for the bin packing problem with variable sized bins. These results are relatively similar to those of [3, 10] and rely heavily on the fact that the cost of a bin equals its size.

Outline. In this paper, we design an APTAS for GCVS. In Section 2 we state and prove some reductions which allow us to seek a slightly simpler structure of solution. These reductions need to be handled carefully to enable the usage of some simplifying assumptions later. In particular, we define a structure for optimal packings, which turns out to give a solution which is not very different from an overall optimal solution, but allows to simplify the search for optimal solutions. Such a solution allows to pack an item into a bin that is either much larger than the smallest bin that can contain this item, or whose cost is not more than a constant multiplicative factor (in terms of ε) away from the cost of such a minimal bin. Before describing the scheme in full detail, we present an outline in Section 3. In Section 4 we show how to apply grouping and rounding procedures on the input. We partition the input into sets as a function of the smallest bin

they can fit into. In order to be able to apply rounding techniques on each set, we show that the largest items of every group can be packed according to one of two very different packing rules. We note that the number of item sizes resulting from the rounding procedure is not a constant. Therefore, many of the known methods to solve such rounded problems cannot be applied to solve our rounded problem. One example of a standard approach which fails is to formulate an integer programming formulation for the rounded problem. However, in our case its dimension is not constant, and therefore we cannot simply use Lenstra's algorithm [8] to solve the rounded instance and find an optimal packing for it. Despite these difficulties we show in Section 5 that we can get a near optimal solution to the original problem using a shortest path computation. We use the properties proved in Section 2 to reduce the size of the graph in which we look for the shortest path, to a polynomial size. Our shortest path computation allocates items to bins, where bin sizes are considered along the path in an increasing order of costs, starting with the cheapest bins. We prove the correctness of our scheme in Section 6. We conclude this paper by some concluding remarks in Section 7.

This algorithm uses methods of rounding and grouping that are based on ideas from [3] and [12]. However, as the adaptation of these ideas into a scheme with general costs requires a treatment of the bin types in a sorted order, we apply a layered graph based scheme for this. Such a scheme (for a scheduling problem) was given by Hochbaum and Shmoys [6]. In order to be able to design a solution for the most general problem with no assumptions on the cost function, we additionally apply some novel methods. Note that the running time of our APTAS depends on the number of bin types, r, polynomially (i.e., r is seen as a part of the input). Throughout this paper we denote by ε a fixed positive constant such that $\varepsilon < \frac{1}{100}$ and $\frac{1}{\varepsilon}$ is an integer.

2 Some reductions

In this section we show a series of modifications on B and restrictions on the optimal solution. We do not apply any modifications on the input items at this time. In the following sections we will compute a solution that uses only the modified set of bins, and approximates an optimal solution among the possible solutions under the specified restrictions. The first reduction keeps at least one optimal solution unaffected, whereas the other reductions change the optimal solutions and moreover result in an increase in the total cost of an optimal solution. However, we will show that this increase is bounded by a (multiplicative) factor of $1 + \varepsilon$.

Lemma 1 *Without loss of generality we assume that the values c_i are monotonically decreasing (i.e., for $i < j$, $c_i > c_j$).*

Proof. To achieve this we show that given a set of bin types and a solution, we can omit some bins from B and to change any solution (to a given input) into a solution that does not use bins removed from B and its cost is not larger than the cost of the original solution. To achieve this, we apply the following process on B. While there exist i, j such that $i < j$ but $c_i \leq c_j$, remove bin type j from B. Note that since $b_i > b_j$, we can move the contents of every bin of size b_j into a bin of size b_i without increasing the cost of a given solution. This is done until no such pair i, j exists, and thus results in a set B' where the sequence of values c_i is monotonically decreasing. ■

Since the process described in Lemma 1 can be applied to B without changing the cost of optimal solutions, we use the notation B for the set of bin type to which the process was already applied. We assume in the remainder of the paper that the values c_i are monotonically decreasing.

The following reductions increase the cost of an optimal solution by a factor of at most $1 + \varepsilon$. In our analysis, we compare the cost of our approximation algorithm to the cost of an optimal solution for the instance resulting from the reductions, and prove an approximation ratio of $1 + O(\varepsilon)$. We get that even

though the "real" approximation ratio (i.e., the approximation ratio with respect to an optimal solution of the original problem) may be slightly larger, it is at most $1+\varepsilon$ times the approximation ratio that we prove, and thus our analysis results in an approximation factor of $1+O(\varepsilon)$. Therefore, since we are interested in designing an APTAS for the problem, the reductions are harmless from our point of view. The next lemma shows that we can assume that the sequence of bin costs decreases geometrically or faster.

Lemma 2 *Without loss of generality we may assume that for all i, $\frac{c_i}{c_{i+1}} \geq 1+\varepsilon$.*

Proof. If the claim does not already hold for the input bin types, we apply the following process on the bin types of the input. Traverse the list of bin types from the largest bin (i.e., $j=1$) to the smallest bin ($j=r$). During the traversal keep only a subset of the types, and remove the other types from B. We keep the first bin type ($j=1$), and recursively assume that the last bin type that is kept has index j. Then, given the value j, for $i=j+1, j+2, \ldots$, as long as $\frac{c_j}{c_i} < 1+\varepsilon$ and $i > j$ we remove the i-th type from the list of bin types. We always keep the bin type with smallest index i such that $\frac{c_j}{c_i} \geq 1+\varepsilon$ and i becomes the new value of j. If there is no such value of i, we remove all bin types $j+1, j+2, \ldots, r$ from B.

Consider a feasible solution that packs the set of items S using a bin of type i that is removed during this process. Then, the set of resulting bins contains a bin type j such that $\frac{c_j}{1+\varepsilon} < c_i < c_j$. Then, we modify the solution by using a bin of type j to pack all the items in S. Applying this procedure on all bins of types that were removed from B, results in a feasible solution to the new instance whose total cost is at most $1+\varepsilon$ times the cost of the original solution. Therefore, the cost of an optimal solution for the new instance is at most $1+\varepsilon$ times the cost of an optimal solution for the original instance. Thus, if we design APTAS for instances satisfying the assumption of the lemma, then the resulting solution will also be an APTAS for the original instances. We conclude that it suffices to consider instances satisfying the property. ■

We say that a feasible solution is *nice* if it satisfies the following condition for every pair of a bin and an item. Assume that the solution uses a bin of type i to pack a set of items S. For a given $j \in S$, let k_j be the maximum index such that $b_{k_j} \geq s_j$ (i.e., b_{k_j} is the smallest bin size where j can be packed). Then, either $b_{k_j} \leq \varepsilon^6 b_i$ or $c_{k_j} \geq \varepsilon^8 c_i$ (or both). This means that an item packed in a bin can either fit into a much smaller bin or that the smallest (and thus cheapest) bin that can accommodate this item has a cost which differs from the cost of the current bin by a constant factor. Note that an item that does not fit into any bin of index strictly larger than i immediately satisfies the second condition.

The next lemma shows that we can restrict ourselves to looking for an approximated nice solution.

Lemma 3 *Given an instance of the* GCVS *problem, denote by* OPT_n *the minimum cost of a nice solution, and by* OPT *the cost of an optimal solution (that is not necessarily nice). Then,* $\text{OPT}_n \leq (1+3\varepsilon)\text{OPT}$.

Proof. Fix an optimal solution O whose cost is OPT. It suffices to show how to transform it into a nice solution whose cost is at most $(1+3\varepsilon)$OPT. We do the transformation for each packed bin in O separately. Assume that O uses a bin of type i to pack the item set which is denoted by C. We use a set of bins to pack C in order to convert the packing of C into a nice packing. To do so, we first identify the sequence $i_1, i_2, \ldots$. This sequence is independent of C and can be computed as a function of i as follows. Let i_ℓ be the smallest value of an index q such that $c_q \leq \varepsilon^{\ell+6} \cdot c_i$. Then, instead of using just one bin (of type i) to pack the items in C, we use one bin of type i and in addition, for each $\ell = 1, 2, \ldots$, we use $\frac{2}{\varepsilon^6}$ bins of type i_ℓ. This set of bins is used to pack C as follows. We use the single bin of type i to pack two sets of items. The first set consists of all items of C with size larger than b_{i_1}. The second set consists of all items j such that $b_{k_j} \leq \varepsilon^6 b_i$ (where k_j is as defined above, the index of a smallest type of bin into which item j can fit). Note that for such items it holds that $s_j \leq \varepsilon^6 b_i$. Denote the set of items that we pack using this unique bin of type i by C_0. The rest of the items from C is partitioned into classes, where the ℓ-th class, denoted by C_ℓ, contains all items whose size is between $b_{i_{\ell+1}}$ and b_{i_ℓ}. I.e., $C_\ell = \{a \in C \setminus C_0 : b_{i_{\ell+1}} < s_a \leq b_{i_\ell}\}$.

Then, the items of set C_0 clearly fit into the bin of type i, since C was originally packed into this bin and $C_0 \subseteq C$. The items of C_ℓ for $\ell \geq 1$, are packed using the First-Fit algorithm into at most $\frac{2}{\varepsilon^6}$ bins of type i_ℓ. To see this last claim note that if $C_\ell \neq \emptyset$ then $b_{i_\ell} \geq \varepsilon^6 b_i$. Therefore, the total size of the items in C_ℓ is at most $\frac{b_{i_\ell}}{\varepsilon^6}$ (as they are packed into a single bin of type i). Since First-Fit opens a new bin only if the total size of the items in the previous bin and the new item is at least the capacity of the bin, it has at most one bin with a total size of less than half the size of the bin, and therefore we conclude that First-Fit, when applied to C_ℓ and bins of type i_ℓ, will use at most $\frac{2}{\varepsilon^6}$ bins.

Therefore, instead of using one bin of type i whose cost is c_i, we use a set of bins whose total cost is at most $c_i + \sum_{\ell=1}^{\infty} \frac{2}{\varepsilon^6} \cdot c_{i_\ell} \leq c_i + \sum_{\ell=1}^{\infty} \frac{2}{\varepsilon^6} \cdot \varepsilon^{\ell+6} \cdot c_i = c_i \cdot \left(1 + 2\sum_{\ell=1}^{\infty} \varepsilon^\ell\right) = c_i \left(1 + \frac{2\varepsilon}{1-\varepsilon}\right) \leq c_i(1+3\varepsilon)$, where the last inequality holds since $\varepsilon \leq \frac{1}{3}$.

It is clear that the resulting packing of C_0 into the bin of type i satisfies the conditions of a nice packing since all items violating the condition were removed from this bin. We next prove that the packing of every newly created bin satisfies the conditions of a nice packing as well. Consider an item $j \in C_\ell$ (which is packed into a bin of type i_ℓ). We show that this item satisfies the second condition of nice packings. By definition, $c_{i_\ell} \leq \varepsilon^{\ell+6} c_i$. Note that $i_{\ell+1}$ must exist since $c_{r+1} = 0$. Consider the bin type k_j. By definition of C_ℓ, $s_j > b_{i_{\ell+1}}$, and thus $k_j < i_{\ell+1}$. However, $i_{\ell+1}$ is the smallest index q for which $c_q \leq \varepsilon^{\ell+7} \cdot c_i$, and thus $c_{k_j} > \varepsilon^{\ell+7} \cdot c_i \geq \varepsilon c_{i_\ell} \geq \varepsilon^8 c_{i_\ell}$.

Application of the above transformation on all the bins of O results in a feasible solution that is also nice (as shown above, by the definition of the sets C_ℓ for $\ell \geq 0$) whose cost is at most $(1+3\varepsilon)\text{OPT}$. ■

In the sequel we assume that the instance satisfies the assumptions of Lemma 2. We approximate the minimum cost nice solution whose cost is denoted by OPT_n, and construct a feasible solution whose cost is at most $(1+O(\varepsilon)) \cdot \text{OPT}_n + f(\frac{1}{\varepsilon})$, where f is some function (which will turn out to be polynomial). Note that the solution that we obtain is not necessarily nice, since the original problem does not require this (it is possible however to convert it into a nice solution in polynomial time by applying a construction as above).

3 Outline of the scheme

In this section we provide the outline of the scheme. The complete details will be given in the following sections.

The first step of the scheme is to pre-process the list of bin sizes so that this list will satisfy the properties of Lemmas 1 and 2.

We next partition the set of items into types: For a bin of type i, we say that an item of size s_j is *large for a bin of type* i if $\varepsilon^6 b_i \leq s_j \leq b_i$. It is *small for a bin of type* i if $s_j < \varepsilon^6 b_i$, and otherwise it is *huge for a bin of type* i. An item j is *large* if there is a type i such that it is large for a bin of type i. We denote by $\mathcal{L}$ the set of large items. We partition $\mathcal{L}$ into sets: For all i, $\mathcal{L}_i$ consists of all the large items for a bin of type i that are huge for a bin of type $i+1$, and $\mathcal{L}_r$ consists of all the large items for a bin of type r.

The next step of the scheme is to apply linear grouping for each $\mathcal{L}_i$ separately. We denote by S_1^i the set of the largest items resulting from the linear grouping of $\mathcal{L}_i$.

Our scheme looks for solutions that satisfy an additional property. That is, for each i we consider only two possibilities for packing S_1^i: Either we have $|S_1^i|$ dedicated bins of size b_i, each of which contains exactly one item from the set S_1^i and no other item is packed into such a special bin, or the items of S_1^i are packed as small items in much larger bins (note that we do not allow mixtures of the two options, for a given value of i). We will show that there exists such a solution that does not cost much more than an optimal nice solution.

To find our solution we construct a layered graph. The graph is split into levels, where each level i corresponds to decisions regarding the packing of bins of type i. Each level consists of $3n+1$ layers, where

each layer is associated with packing at most one bin of the corresponding type. At the entry for each level we decide whether S_1^i is packed in dedicated bins or the items of S_1^i are packed as small items in much larger bins (the graph contains edges of both possibilities). Each vertex encodes the number of (large) items of each rounded size that still needs to be packed. A vertex encodes also the rounded total size of the small items (i.e., small for the bin type of its level) that still need to be packed. Each vertex needs to recall the subset of the indices i such that S_1^i is packed as small items only if for the current level the items of S_1^i are still large items (and not small items).

We then look for a shortest path in this (very large but still polynomial-size) layered graph. This shortest path corresponds to a well-defined packing of the items that are packed as large items. Afterwards, the remaining items need to be distributed to the empty slots in the resulting packing. To this end, additional bins (of each size) are used, if the process of packing small items as indicated by our path, did not result in packing a large enough total size of small items. We show that these additional bins have a small cost and do not hurt the returned solution too much. Thus we show that the resulting solution is a good approximation of an optimal solution.

4 Linear grouping

Recall that for a bin of type i, we say that an item of size s_j is *large for a bin of type* i if $\varepsilon^6 b_i \leq s_j \leq b_i$. It is *small for a bin of type* i if $s_j < \varepsilon^6 b_i$, and otherwise it is *huge for a bin of type* i. An item j is *large* if there is a type i such that it is large for a bin of type i. We denote by $\mathcal{L}$ the set of large items.

We next partition $\mathcal{L}$ into subsets according to the size of the items. $\mathcal{L}_r$ is the set of large items for a bin of type r. If we defined $\mathcal{L}_{j+1}, \ldots, \mathcal{L}_r$, then $\mathcal{L}_j$ is defined as the intersection of the set of large items for a bin of type j and the set $\mathcal{L} \setminus (\mathcal{L}_{j+1} \cup \cdots \cup \mathcal{L}_r)$.

For each i such that $|\mathcal{L}_i| \leq \frac{1}{\varepsilon^{16}}$ we pack each item of $\mathcal{L}_i$ in a bin of type i by itself (such a bin is called a dedicated bin). Such a class $\mathcal{L}_i$ with at most $\frac{1}{\varepsilon^{16}}$ elements is called *thin*.

Lemma 4 *The total cost of packing each item of a thin class into a dedicated bin is at most* $\frac{1+\varepsilon}{\varepsilon^{17}}$.

Proof. Since each bin type is used to pack at most $\frac{1}{\varepsilon^{16}}$ items, the total cost of these bins is at most the total cost of using $\frac{1}{\varepsilon^{16}}$ copies of every bin in the input sequence. I.e., it is at most $\frac{1}{\varepsilon^{16}} \cdot \sum_{i=1}^{r} c_i \leq \frac{1}{\varepsilon^{16}} \cdot c_1 \cdot \sum_{i=0}^{r-1} \left(\frac{1}{1+\varepsilon}\right)^i \leq \frac{1}{\varepsilon^{16}} \cdot c_1 \cdot \sum_{i=0}^{\infty} \left(\frac{1}{1+\varepsilon}\right)^i = \frac{1+\varepsilon}{\varepsilon^{17}}$ where the first inequality holds by Lemma 2. ■

By Lemma 4, we can assume without loss of generality that for each non-empty class of large items, the class has at least $\frac{1}{\varepsilon^{16}}$ elements.

Next, we perform a linear grouping of each class $\mathcal{L}_i$, separately. More precisely, let $|\mathcal{L}_i| = n_i$ (recall that we assume that $n_i \geq \frac{1}{\varepsilon^{16}}$). We sort the elements $a_1^i, a_2^i, \ldots a_{n_i}^i$ of $\mathcal{L}_i$ according to their size. That is, we denote the size of the element a_j^i by s_j^i, and we assume without loss of generality that $s_1^i \geq s_2^i \geq \cdots \geq s_{n_i}^i$. We partition $\mathcal{L}_i$ into $\frac{1}{\varepsilon^{16}}$ subclasses denoted by $\bar{S}_1^i, \bar{S}_2^i, \ldots, \bar{S}_{1/\varepsilon^{16}}^i$. The partition is defined by the following two conditions. $|\bar{S}_p^i| = \lfloor n_i \varepsilon^{16} \rfloor$ or $|\bar{S}_p^i| = \lceil n_i \varepsilon^{16} \rceil$ for all $p \geq 1$, and if $p < q$ then $|\bar{S}_p^i| \geq |\bar{S}_q^i|$ (thus we always have $|\bar{S}_1^i| = \lceil n_i \varepsilon^{16} \rceil$). Moreover, we require that if $a_j^i \in \bar{S}_p^i$ and $a_k^i \in \bar{S}_q^i$ such that $p < q$, then $s_j^i \geq s_k^i$. Thus $\bar{S}_1^i$ is a set which contains the largest $\lceil n_i \varepsilon^{16} \rceil$ elements of $\mathcal{L}_i$ (breaking ties arbitrarily). In general, we partition $\mathcal{L}_i$ to approximately equal size sets (sets of lower indices may have one additional item compared to sets of higher indices) so that $\bar{S}_j^i$ contains the largest elements from $\mathcal{L}_i \setminus \left(\bar{S}_1^i \cup \cdots \cup \bar{S}_{j-1}^i\right)$. We note that $n_i \varepsilon^{16} \leq |\bar{S}_1^i| \leq \lfloor n_i \varepsilon^{16} \rfloor + 1 \leq 3|\mathcal{L}_i \setminus \bar{S}_1^i| \cdot \varepsilon^{16}$, where the last inequality can be proved using simple algebra and the properties $|\mathcal{L}_i \setminus \bar{S}_1^i| = n_i - |\bar{S}_1^i| \geq n_i - (n_i \varepsilon^{16} + 1)$, $\varepsilon < \frac{1}{3}$ and $n_i \geq \frac{1}{\varepsilon^{16}}$.

For all i and all $j \geq 2$, we round up the size of all the elements of $\bar{S}^i_j$ to the size of the largest element of $\bar{S}^i_j$, the set of rounded items is denoted by S^i_j, and we denote by σ^i_j the rounded up size of an item in S^i_j. We also define $S^i_1 = \bar{S}^i_1$. The set of rounded (large) items, resulting from $\mathcal{L}_i$ is denoted by $\mathcal{L}'_i$, and the set of all rounded (large) items is denoted by $\mathcal{L}'$. In the APTAS of [3] for the classical bin-packing problem, the set of the largest elements in the linear grouping is packed using separate bin for each such item. This packing is applied also for the later APTAS of Murgolo [12] for the variable size bin packing problem. In both cases, such a packing increases the cost of a packing only by an arbitrarily small factor. In the generalized problem, the important property that a small number of items of $\mathcal{L}_i$ can be packed into separate bins of size b_i does not hold, since an optimal packing may pack many items of $\mathcal{L}_i$ into larger bins. Instead, for each i, we allow the algorithm to choose between two possibilities: in the first possibility the algorithm packs S^i_1 using a separate bin of size b_i for each item in S^i_1. This is the typical packing of the set of largest items of the linear grouping, and it is useful (by the analysis of the algorithm) in cases where most of the items of $\mathcal{L}_i$ are packed in OPT_n into bins of size smaller than $\frac{b_i}{\varepsilon^6}$. For other cases, we allow the algorithm to pack the set S^i_1, seeing them as a part of the set of small items of larger bins. In this case the algorithm will pack the elements of S^i_1 using bins of size at least $\frac{b_i}{\varepsilon^6}$. That is, in our scheme the total size of elements of S^i_1 will be added to the total size of small elements that the algorithm needs to pack using bins whose size is at least $\frac{b_i}{\varepsilon^6}$. To state the next lemma we consider a fixed optimal nice solution (the optimum among the nice solutions) denoted by OPT_n. We use $\mathcal{A}_i$ to denote the subset of elements of $\mathcal{L}_i \setminus \bar{S}^i_1$ which OPT_n packs into bins of size smaller than $\frac{b_i}{\varepsilon^6}$. If $|\mathcal{A}_i| \geq \frac{1}{2}|\mathcal{L}_i \setminus \bar{S}^i_1|$ we say that $\mathcal{L}_i$ is *good*.

Lemma 5 *Assume that $\mathcal{L}_i$ is good. Consider a packing where each item of S^i_1 is packed into a separate bin of size b_i. In this case, the total cost of bins containing the items of S^i_1 is at most $6 \cdot \varepsilon^4 \cdot \psi_i$, where ψ_i is the total cost of the subset of bins in OPT_n, that consists of all bins that contain at least one item of $\mathcal{A}_i$.*

Proof. We first argue that each bin that is used to pack an element of $\mathcal{A}_i$, has size of at least b_i. Since $\mathcal{A}_i \subset \mathcal{L}_i$, such an item is not large for b_{i+1}, and since $b_{i+1} < b_i$ this means that it is huge for it and does not fit into smaller size bins. Therefore, the cost for such a bin is at least c_i. Next note that each bin that is used to pack an element of $\mathcal{A}_i$, packs at most $\frac{1}{\varepsilon^{12}}$ such elements. This follows since each such element has size of at least $\varepsilon^6 b_i$, and the size of such bin is smaller than $\frac{b_i}{\varepsilon^6}$ (due to the definition of $\mathcal{A}_i$). Therefore, there exist at least $\frac{|\mathcal{L}_i \setminus \bar{S}^i_1|}{2}\varepsilon^{12}$ bins of OPT_n, where each one of them costs at least c_i, and each one of them contains at least one element of $\mathcal{A}_i$. So $\psi_i \geq \frac{|\mathcal{L}_i \setminus \bar{S}^i_1|}{2}\varepsilon^{12} c_i$. As already stated in the proof of Lemma 4, $|S^i_1| = |\bar{S}^i_1| \leq 3|\mathcal{L}_i \setminus \bar{S}^i_1|\varepsilon^{16}$, and therefore $|S^i_1|$ is at most $6 \cdot \varepsilon^4$ times the number of bins that are used by OPT_n to pack the elements of $\mathcal{A}_i$. The claim follows since each bin used to pack an element of $\mathcal{L}_i$ costs at least c_i which is exactly the cost of the bins used to pack each element of S^i_1. ■

Lemma 6 *The total cost of assigning each element of S^i_1 a separate bin, for all values of i such that $\mathcal{L}_i$ is good, is at most $3\varepsilon^2 \cdot \text{OPT}_n$.*

Proof. By Lemma 5, the total cost incurred by S^i_1 for a value of i such that $\mathcal{L}_i$ is good, is at most $6\varepsilon^4$ times the total cost of the bins that OPT_n uses to pack elements of $\mathcal{A}_i$. Since OPT_n is nice, and the first property of nice packings does not hold for the bin sizes that we consider here, we conclude that each of these bins has a cost of at least c_i and at most $\frac{c_i}{\varepsilon^8}$. By Lemma 2, there are at most $\left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon^8} \right\rceil$ bin types with cost in the interval $\left[c_i, \frac{c_i}{\varepsilon^8}\right]$. Therefore, a specific bin can be used by OPT_n to pack elements from at most $\left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon^8} \right\rceil$ different sets $\mathcal{A}_i$. Thus, the total cost of assigning each element of S^i_1 a separate bin for all i such that $\mathcal{L}_i$ is good, is at most $6\varepsilon^4 \cdot \left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon^8} \right\rceil \cdot \text{OPT}_n \leq 3\varepsilon^2 \text{OPT}_n$ where the last inequality can be easily verified and holds since $\varepsilon < \frac{1}{100}$. ■

By Lemma 6, if we allow our scheme to choose one of the two possibilities described above for each i, then the cost of the largest sets in the linear groupings can be disregarded if the class is good. We argue in the next lemma that if the class $\mathcal{L}_i$ is not good then by deciding to pack S_1^i in bins of size at least $\frac{b_i}{\varepsilon^6}$, we increase the space demand for such bins by a multiplicative factor of at most $(1+6\varepsilon^{10})$. This increase does not cause any harm, since in any case we will have more dominant rounding errors (that will still lead to an $(1+O(\varepsilon))$-approximate solution with respect to the asymptotic approximation ratio).

Lemma 7 *If $\mathcal{L}_i$ is not good, then the total size of the elements of S_1^i is at most $6\varepsilon^{10}$ times the total size of elements of $\mathcal{L}_i \setminus \bar{S}_1^i$ that are packed in* OPT_n *into bins which are of size at least $\frac{b_i}{\varepsilon^6}$.*

Proof. Using $\mathcal{A}_i < \frac{1}{2}|\mathcal{L}_i \setminus \bar{S}_1^i|$, we get $|\mathcal{L}_i \setminus (\bar{S}_1^i \cup \mathcal{A}_i)| \geq \frac{1}{2}|\mathcal{L}_i \setminus \bar{S}_1^i|$ and thus $\frac{|S_1^i|}{|\mathcal{L}_i \setminus (\bar{S}_1^i \cup \mathcal{A}_i)|} \leq \frac{3|\mathcal{L}_i \setminus \bar{S}_1^i|\varepsilon^{16}}{\frac{1}{2}|\mathcal{L}_i \setminus \bar{S}_1^i|} \leq 6\varepsilon^{16}$. Since $\varepsilon^6 b_i \leq s_j^i \leq b_i$ for all j, the claim follows. ■

The next lemma holds by a similar argument to the arguments of [3]. Namely, using the fact that if the items of S_1^i are removed for all i, the rounded input can be mapped back into the original input, so that every item is mapped to an item that is no smaller than it.

Lemma 8 *Given an instance of the* GCVS *problem, then the optimal solution cost of the instance after applying the linear grouping step as described above (with rounded up sizes), and removing all items in S_1^i for all i, is at most the cost of the optimal solution to the original instance.*

5 The main scheme

In this section we show how to use the linear grouping, as it is described in the previous section, to compute an approximated nice solution. Our method is based on constructing a layered graph, and then computing a shortest path in this graph. This shortest path is then used to construct a feasible solution of the GCVS problem.

For $i = 1, 2, \ldots, r$ we denote by d_i the total size of elements, whose individual sizes are in the interval $(b_{i+1}, \varepsilon^6 b_i)$. Such elements are huge with respect to bin types $i+1, i+2, \ldots, r$ and small elements with respect to bin type i. They are not large with respect to any type of bin. If $b_{i+1} \geq \varepsilon^6 b_i$, then $d_i = 0$.

A level in a layered graph is defined as a consecutive set of layers. Our layered directed graph $G = (V, E)$ is composed of $r+1$ levels, where the i-th level corresponds to decisions regarding the packing of elements into bins of type i (recall that $b_{r+1} = 0$ and hence there are no elements packed into bins of type $r+1$. We add this level to unify the presentation of our scheme). Each level consists of $3n+1$ layers where n is the total number of items in the input. Each edge connects a vertex of one layer with a vertex of the consecutive layer, where the layers are ordered so that first there are the $3n+1$ layers of level $r+1$, then the layers of level r, and so on up to the layers of level 1. We add to G one additional vertex denoted by t, which is defined to be the very last layer. We partition the levels into phases as follows. A level i is a *phase p level* if $b_i \in \left(\varepsilon^p, \varepsilon^{p-1}\right]$. For a bin type i, we denote by $B_i = \{k : b_k \in \left[\varepsilon^6 b_i, b_i\right]$ and $c_k \in \left[\varepsilon^8 c_i, c_i\right]\}$.

For a phase p level i, a *pattern of level i* corresponds to a packing of a bin of size b_i with elements resulting from the linear grouping, which are of size in the interval $[\varepsilon^6 b_i, b_i]$. These are elements in $\bigcup_{\substack{k \in B_i \\ j \geq 2}} S_j^k \cap \{a \in S : \sigma_a \in [\varepsilon^6 b_i, b_i]\}$. Each such bin can contain space for smaller items as well, where the space defined by the pattern is an integer multiple of $\varepsilon^6 b_i$. This integer is called *the number of slots for small items of the pattern* and denoted by n_{slot}. Formally, a pattern of level i is $\left(n_{slot}, (n_j^k)_{\substack{k \in B_i \\ j \geq 2}}\right)$, where n_j^k is the number of items from S_j^k that are packed into this bin. In this notation we assume that $n_j^k = 0$ if $\sigma_j^k < \varepsilon^6 b_i$.

Lemma 9 *The number of possible patterns of level i is $O((n\varepsilon^{16}+2)^{\frac{8\log_{1+\varepsilon}\frac{1}{\varepsilon}+1}{\varepsilon^{16}}}\cdot(\frac{1}{\varepsilon^6}+1))$. I.e., it is bounded by a polynomial in the input size.*

Proof. The number n_{slot} is an integer in the interval $[0,\frac{1}{\varepsilon^6}]$. The number n_j^k is an integer in the interval $[0,\lceil n\varepsilon^{16}\rceil]$. By Lemma 2, $|B_i|$ is at most $\log_{1+\varepsilon}\frac{1}{\varepsilon^8}+1$. Since each such class (whose index belong to B_i) has at most $\frac{1}{\varepsilon^{16}}$ different values of element size, we conclude that the number of possible patterns of level i is at most $O((n\varepsilon^{16}+2)^{\frac{8\log_{1+\varepsilon}\frac{1}{\varepsilon}+1}{\varepsilon^{16}}}\cdot(\frac{1}{\varepsilon^6}+1))$. ■

We next describe the vertex set of our layered graph G. A vertex u of the i-th level, which is a phase p level, is associated with a label consisting of the following information which has four parts. The first part contains information on the number $n_j^k(u)$ of items from S_j^k that still needs to be packed, for relevant values of k and j, if the current vertex is reached. That is, for every $k\in B_i$ and for all $j\geq 2$, the label contains the number $n_j^k(u)$ of remaining such elements. The second part of the label contains information on the total size D of small elements for a bin of type i that needs to be packed. This information in the label is $n_{slot}(u)=\left\lfloor\frac{D}{b_i\varepsilon^6}\right\rfloor$. Note that D is an approximated sum of such items, which never exceeds the real amount to be packed. The third part of the label contains approximated sums of items that we decided to pack as small items, but are still considered to be large items for bins of type i, and thus this information should be carried over to future phases. To be more precise, if level i is a phase p level, then the label contains six integer numbers $n_{slot}^p(u)$, $n_{slot}^{p+1}(u),\ldots,n_{slot}^{p+5}(u)$, where $n_{slot}^q(u)\cdot b_i\varepsilon^6$ (for $q=p,p+1,\ldots,p+5$) is an approximated sum of the sizes of all items whose sizes are in $\left(\varepsilon^q,\varepsilon^{q-1}\right]$, and the smallest bin that is large enough to contain such items is of type k for some $k>i$ such that $c_k<\varepsilon^8 c_i$ (and therefore $k\notin B_i$). Note that such items are packed as small items in a nice solution, however they are still large items for a bin of type i (this is so since for an item x we have $\varepsilon^q<s_x\leq b_k<b_i\leq\varepsilon^{p-1}\leq\varepsilon^{q-6}$ and therefore $s_x\geq\varepsilon^6 b_i$). The fourth part of the label is described next. For all $k\in B_i$, the label contains the information regarding whether we decide to pack each member of S_1^k using its own bin of size b_k, or all elements of S_1^k are packed as small items in bins of size at least $\frac{b_k}{\varepsilon^6}$. Therefore, the label encodes a subset $I(u)$ of the index set B_i and if k belongs to this subset, we decide to pack each element of S_1^k using a dedicated bin of size b_k.

Lemma 10 *The number of vertices in the graph is polynomial in the input size.*

Proof. Consider a phase p level i and a given layer of this level. In this layer there is a vertex for each possible value of the label. Since the number of levels is $r+1$ and the number of layers in each level is $3n+1$ (plus one last layer which consists of a single vertex), in order to prove the claim it suffices to show that the number of possible labels for level i is polynomial. The number $n_j^k(u)$ is an integer in the range $[0,n_i]$, and therefore there are at most $n+1$ such possibilities. We have to identify this number for the pairs of indices k and j such that $b_k\in[\varepsilon^6 b_i,b_i]$, $c_k\in[\varepsilon^8 c_i,c_i]$ and $j=2,3,\ldots,\frac{1}{\varepsilon^{16}}$, and by Lemma 2 there are at most $1+\log_{1+\varepsilon}\frac{1}{\varepsilon^8}$ values that k can have. Therefore, the number of possibilities for the first part of the label is at most $(n+1)^{(1+\log_{1+\varepsilon}\frac{1}{\varepsilon^8})\cdot\frac{1}{\varepsilon^{16}}}$ and since ε is a fixed constant, this number is polynomial in the input size. Next, consider the number of possibilities for $n_{slot}(u)$. Note that the number of small elements for a bin of type i is at most n, and each one of them has size less than $\varepsilon^6 b_i$. Therefore, $D<nb_i\varepsilon^6$, and we conclude that $n_{slot}(u)=\left\lfloor\frac{D}{b_i\varepsilon^6}\right\rfloor$ is an integer in the interval $[0,n-1]$. Hence, the number of possibilities for $n_{slot}(u)$ is at most n. We next bound the value of $n_{slot}^q(u)$ for $q=p,p+1,\ldots,p+5$. Note that each item considered so far has a size smaller than b_i, and there are at most n such items. Therefore, the total size of these items (that still needs to be packed) is less than nb_i. Hence, $n_{slot}^q(u)\in\left[0,\frac{nb_i}{b_i\varepsilon^6}\right)$, since this is an integer, we get that the number of possibilities for this value is polynomial (for a fixed value of ε). It remains to bound the number of possibilities for $I(u)$. By Lemma 2, $|B_i|=\log_{1+\varepsilon}\frac{1}{\varepsilon^8}+O(1)$, and the number of

possibilities for $I(u)$ is the number of subsets of B_i that is $O(2^{\log_{1+\varepsilon}\frac{1}{\varepsilon^8}})$ that is a constant (for a fixed value of ε). ■

We define five types of edges. The first two types of edges are edges connecting two vertices from a common level (and consecutive layers). The last types of edges connect vertices from consecutive levels. The first type of edges has the purpose of assigning a set of items into a bin. The second type allows to bypass a bin and not use it. The next two types translate configurations of different levels. The last type allows to terminate all paths in the target vertex t.

1. The first type of an edge connects two vertices from two consecutive layers of a common level i where i is a phase p level. It corresponds to a packing of a single bin of size b_i according to a pattern of level i denoted by $\left(n_{slot}, (n_j^k)_{\substack{k\in B_i \\ j\geq 2}}\right)$. The two vertices connected by such an edge (u,v) have labels that differ only in the first and second part of the label (so $I(u) = I(v)$ and $n^q_{slot}(u) = n^q_{slot}(v)$ for $q = p, p+1, \ldots, p+5$). The first part of the label changes according to $n_j^k(v) = \left(n_j^k(u) - n_j^k\right)^+$ for all $k \in B_i$ and for all $j \geq 2$ (where for a real number a we let $a^+ = \max\{a, 0\}$). That is, the first part of the the label of v results from the label of u by decreasing the amount of items that needs to be packed from the set S_j^k by exactly the number of elements from S_j^k that are packed by the pattern corresponding to this edge. Similarly, $n_{slot}(v) = (n_{slot}(u) - n_{slot})^+$. Such an edge has a cost of c_i.

2. The second type of an edge is an edge connecting two vertices in consecutive layers of a common level with equal labels. Such an edge has a zero cost. Such an edge corresponds to the decision not to pack an additional bin of the corresponding size.

3. The third type of an edge connects a vertex u that belongs to the last layer of phase p level $i+1$, to a vertex v that belongs to the first layer of level i where level i is also a phase p level. In order to obtain the label of v from the label of u we need to apply the following changes: For each value of $k \in B_{i+1} \setminus B_i$, we remove the entries $n_j^k(u)$, for all $j \geq 2$, from the first part of the label of u. These are items that lost their opportunity to be packed as large items and must be packed as small items to maintain the properties of a nice packing. Out of these elements, we compute the total size of elements that need to be packed as small elements according to u and they are already small elements for bin of type i. We need to add to this total size the value of d_i, which are items that are immediately small without being large for any previous bin. However, since the two levels are of the same phase, b_i and b_{i+1} are within a factor of $\frac{1}{\varepsilon}$, which implies that such items do not exist and so $d_i = 0$. Denote the resulting value, which is the sum of all items that became small, by T. Then, the second part of the label of v is exactly $n_{slot}(v) = \left\lfloor \frac{T}{\varepsilon^6 b_i} + \frac{n_{slot}(u)\cdot b_{i+1}}{b_i} \right\rfloor$. The first part of the label of v is augmented with new entries for the amount of elements of S_j^i that need to be packed for $j \geq 2$, we showed how to compute this number earlier, and it is either $n_j^i = \lfloor n_i \varepsilon^{16} \rfloor$ or $n_j^i = \lceil n_i \varepsilon^{16} \rceil$ for all $j \geq 2$. Note that $B_i \setminus B_{i+1} = \{i\}$, thus these are the only new entries. For $q = p, p+1, \ldots, p+5$, we define $n^q_{slot}(v)$ as follows: for each $k \in B_{i+1}$ such that $c_k < \varepsilon^8 c_i$, we compute $Size_k^q(u) = \sum_{j\geq 2 : \sigma_j^k \in (\varepsilon^q, \varepsilon^{q-1}]} n_j^k(u) \cdot \sigma_j^k$. We sum up all the values $Size_k^q(u)$ for all k such that $k \in B_{i+1}$ and $c_k < \varepsilon^8 c_i$, and denote this sum by $E_q(u)$. That is, $E_q(u) = \sum_{k\in B_{i+1} : c_k < \varepsilon^8 c_i} Size_k^q(u)$. Then, we let $n^q_{slot}(v) = \left\lfloor \frac{n^q_{slot}(u)\cdot b_{i+1}}{b_i} + \frac{E_q(u)}{b_i \varepsilon^6} \right\rfloor$. We finish the definition of the label of v by setting either $I(v) = I(u) \cap B_{i+1} \cap B_i$ or $I(v) = (I(u) \cap B_{i+1} \cap B_i) \cup \{i\}$ (both edges are constructed, having the exact same cost). The cost of the edge (u,v) is $\sum_{k \in I(u) \setminus I(v)} |S_1^k| \cdot c_k$ and this cost reflects the cost of packing each element of S_1^k in a separate bin of type k if $k \in I(u)$. We charge this packing of the S_1^k to an edge of type 3 or 4, which is a transition between levels, at the time that the packing of S_1^k stops being indicated in the label.

4. The fourth type of edges connects a vertex u that belongs to the last layer of phase p level $i+1$, to a vertex v that belongs to the first layer of level i where level i is a phase p' level for $p' \leq p-1$. In order to obtain the label of v from the label of u we need to apply the following changes: For each value of $k \in B_{i+1} \setminus B_i$, we remove from the first part of the label of u the entries $n_j^k(u)$ for all $j \geq 2$. Out of these items, we compute the total size of such elements that need to be packed according to u and they are already small for a bin of type i, we add to this total size the value of d_i. We denote the resulting value by T. We need to take into account the following term $Y = \sum_{q=\max\{p,p'+6\}}^{p+5} n_{slot}^q(u)$. The last term corresponds to items that we previously decided to pack as small items, but were still large for bins of type $i+1$, and now such an item x satisfies $s_x < \varepsilon^5 b_i$, as $s_x \leq \varepsilon^{q-1}$ and $b_i > \varepsilon^{p'} \geq \varepsilon^{q-6}$. Note that we slightly relax the condition of packing items as small, and sometimes allow to pack items that are smaller than a bin by a factor of at least ε^5 (instead of ε^6) to be packed as small items in this bin. Then, the second part of the label of v is exactly $n_{slot}(v) = \left\lfloor \frac{T}{\varepsilon^6 b_i} + \frac{(n_{slot}(u)+Y)\cdot b_{i+1}}{b_i} \right\rfloor$. The first part of the label of v is augmented with new entries for the number of elements of S_j^i that needs to be packed for $j \geq 2$, and this number is either $n_j^i = \lfloor n_i \varepsilon^{16} \rfloor$ or $n_j^i = \lceil n_i \varepsilon^{16} \rceil$ for all $j \geq 2$. For $q = p', p'+1, \ldots, p'+5$, we define $n_{slot}^q(v)$ as follows. For a given value of q, denote by Y_q the set of pairs (k,j) such that $(k,j) \in Y_q$ if an only if $k \in B_{i+1} \setminus B_i$ and $\sigma_j^k \in (\varepsilon^q, \varepsilon^{q-1}]$. We compute $E_q(u) = \sum_{(k,j)\in Y_q} n_j^k(u) \cdot \sigma_j^k$. For $q \notin \{p, p+1, \ldots, p+5\}$, we denote $n_{slot}^q(u) = 0$. Then, for $q = p', p'+1, \ldots, p'+5$, we let $n_{slot}^q(v) = \left\lfloor \frac{n_{slot}^q(u)\cdot b_{i+1}}{b_i} + \frac{E_q(u)}{\varepsilon^6 b_i} \right\rfloor$. Note that every entry $n_{slot}^q(u)$ was either translated into a part of the small jobs or into a part of an entry $n_{slot}^q(v)$ but not to both.

We finish the definition of the label of v by setting either $I(v) = I(u) \cap B_{i+1} \cap B_i$ or $I(v) = (I(u) \cap B_{i+1} \cap B_i) \cup \{i\}$ (again, both edges exist with the same cost). The cost of the edge (u,v) is $\sum_{k \in I(u) \setminus I(v)} |S_1^k| \cdot c_k$ and this cost reflects the cost of packing each element of S_1^k in a separate bin of type k if $k \in I(u)$. As in the previous type of edges, we charge this packing of the S_1^k items to the edges the level transition, where the packing of S_1^k stops being indicated in the label.

5. The last type of edges connect vertices u of the last layer of level 1 to t. Such a vertex u is adjacent to t only if all parts of the label of u except for the last one are either empty sets or are equal to zero. The cost of such an edge if it exists is $\sum_{k \in I(u)} |S_1^k| \cdot c_k$.

Note that $b_{r+1} = 0$ and thus no items can fit into this bin (that was created so that the levels of the graph can be created uniformly). Let a be the vertex of the first layer of level $r+1$ whose label is defined as follows. The first part, which indicates the amounts of items in S_k^j for $2 \leq j \leq \frac{1}{\varepsilon^{16}}$ and $k \in B_{r+1}$, is empty as $B_{r+1} = \emptyset$. The second part is zero and so are the six amounts in the third part. Finally, $I(a) = \emptyset$. Our scheme finds the minimum cost path $\mathcal{P}$ from a to t. Note that items to be packed are added to the graph as soon as the first level of any bin that can accommodate them is reached.

We next construct a feasible solution based on the path found by the algorithm. If the path uses an edge connecting (u,v) where both of them belong to a common level i and the label of u differs from the label of v, then this edge corresponds to a pattern $\left(n_{slot}, (n_j^k)_{\substack{k \in B_i \\ j \geq 2}}\right)$. We open a bin of type i and allocate it exactly n_j^k items of S_j^k for all $k \in B_i$ and $j \geq 2$ (the actual allocation is of the $\bar{S}_j^k$ items, but we may assume that each of them occupies the same space which is the size of an S_j^k item), we also reserve a space of size $n_{slot}\varepsilon^6 b_i$ for small items (that will be allocated later to these spaces). We apply this for all edges of the first type that $\mathcal{P}$ uses. The second type of edges in $\mathcal{P}$ do not affect the solution (they have cost zero with an empty configuration, and are associated with bins that are bypassed). Next assume that $\mathcal{P}$ uses an edge (u,v) of

the third or fourth type where u belongs to the $i+1$-th level (and v to level i). The effect of such an edge except for translation between levels is to assign large items from a set S_1^k such that $k \in B_{i+1}$ and $k \notin B_i$, which are supposed to be packed in their own bins, one item per bin. In this case for all $k \in I(u) \setminus I(v)$ we open $|S_1^k|$ bins of size b_k and allocate them the items of S_1^k. It remains to consider the edge of the last type that $\mathcal{P}$ uses. Assume that this edge is (u,t) then for all $k \in I(u)$ we open $|S_1^k|$ bins of size b_k and allocate them the items of S_1^k. Note that the total cost of the bins that we open is exactly the cost of $\mathcal{P}$.

We next consider the non-allocated items (that are supposed to be packed as small items as implied by the chosen path). We sort the non-allocated items in a non-increasing size order. We start to allocate them into the space kept for small items as follows.

Let μ_i be the number of slots for small items in bins of level i in the solution as implied by the sum of values n_{slot} in the patterns. Let R_i be the number of edges of the first type in the path (inside level i) that contain at least one slot for small items. We would like to allow a total size of at least $\varepsilon^6 b_i(\mu_i + 37)$ that will be allocated to items to be packed and for that we open $R_i\varepsilon^5(1+\varepsilon)+2$ new bins of type i for small items.

We start to allocate the largest remaining item to the space in the smallest type bin (the largest bin), and keep allocating items according to the sorted list into the space in the bins. While allocating items into bins of level i, if there is not enough room for the current item in the current space we move to the next space. We do this as long as there are free slots in the bins that correspond to edges of the first type. We use the new bins of size b_i and assign small items into them until either we run out of small items to be packed completely, or we have used all new bins that were opened for level i as stated above. A new bin that is opened in this step is called a *small item bin of type* i. We later show that our bound (i.e., the number of new bins per level) is large enough, and we bound the additional cost caused by the new bins.

We conclude this section by noting that since the layered graph has polynomial size, and its construction takes polynomial time, finding the shortest path in G takes a polynomial time and constructing the solution based on this path is also polynomial. Therefore, our scheme (for a fixed value of ε) is a polynomial time algorithm. Hence we establish the following corollary.

Corollary 11 *Given a fixed value of ε, the time complexity of the scheme is polynomial.*

6 Analysis

In this section we prove that our algorithm is an APTAS for GCVS. We first bound the cost of $\mathcal{P}$. To do so, we present a path from a to t in G whose cost is $(1+3\varepsilon^2+2\varepsilon^6)\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}$.

Lemma 12 *There exists a path $\tilde{\mathcal{P}}$ from a to t whose cost is at most $(1+3\varepsilon^2+2\varepsilon^6)\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}$.*

Proof. Consider an optimal solution OPT_n. From this solution, remove all items which belong to thin classes as such items do not exist in the layered graph.

Before we can define a path in the graph, we make some adaptations to the solution, and specifically, we convert it into a packing of the rounded items where some of the items that are packed as small items are converted into tiny items in the packing.

This packing is later translated into a path in the graph, where items that are packed as small need to be considered carefully. We are going to use the space allocated in the solution to the items of $\bar{S}_j^k$ to accommodate the items of S_{j+1}^k, that are (by definition) no larger of items of $\bar{S}_j^k$. This leaves the items of $\bar{S}_1^k$ unpacked. Thus we create a new instance of each item of $\bar{S}_1^k = S_1^k$, the additional set with the new instances of these items is denoted by $\mathcal{S}^k$. The room taken by each item of the original set $\bar{S}_1^k$ will be used later for items of S_2^k. No adaptations are performed on the placement of a new set $\mathcal{S}^k$. The replacement of $\bar{S}_j^k$ items into the S_{j+1}^k items is done after a packing for the new set $\mathcal{S}^k$ is created.

We next define a packing of the items in $\mathcal{S}^i$ for $1 \leq i \leq r$. For every such i such that $\mathcal{L}_i$ is good, i.e., if OPT_n packs at least half of the elements of $\mathcal{L}_i$ using bins of size smaller than $\frac{b_i}{\varepsilon^6}$, we put each item of $\mathcal{S}^i$ into a bin of size b_i. We earlier showed in Lemma 6 that this transformation increases the cost of the solution by at most an additive factor of $3\varepsilon^2\text{OPT}_n$.

We would like to pack the items of $\mathcal{S}^i$ for values of i such that $\mathcal{L}_i$ is not good. This packing is composed of several steps. These items are converted into "sand", which are infinitely small items. We insert a set of empty bins into the packing, and bound the resulting increase in the cost. Next, then we show that these bins are sufficient to accommodate all sand resulting from items of the sets $\mathcal{S}^i$ for values of i, such that $\mathcal{L}_i$ is not good. We do not convert the items into their original sizes since these items are small for the bins into which they are packed, and the graph only takes into account an approximated sum of sizes of such items rather than the specific sizes. Given the list of bins used by OPT_n we do the following. Let N_i be the number of bins of type i used by OPT_n. We open another $\lceil 2N_i\varepsilon^6 \rceil \leq 2N_i\varepsilon^6 + 1$ bins of type i. This increases the cost of the solution by an additive factor of at most $2\varepsilon^6\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}$ (the last term is found by summing up on the costs of all types, similarly to the summation in the proof of Lemma 4).

By Lemma 7, the space required for the sand resulting from a set $\mathcal{S}^i$ (where $\mathcal{L}_i$ is not good) is at most $6\varepsilon^{10}$ times the total size of items in $\mathcal{L}_i \setminus \bar{S}_1^i$ that are packed into bins of size at least $\frac{b_i}{\varepsilon^6}$. The value $D_{i,k}$ is now defined for values of i, k such that $\mathcal{L}_i$ is not good, and $b_k \geq b_i$. We let $D_{i,k}$ be the sum of sizes of the items of $\mathcal{L}_i \setminus \bar{S}_1^i$ that are packed into bins of type k as small items for these bins. For other values of i, k we let $D_{i,k} = 0$. We associate a total size of (at most) $6\varepsilon^{10}D_{i,k}$ of sand created from items of $\mathcal{S}^i$ with these items, and specifically, for every item in $\mathcal{L}_i \setminus \bar{S}_1^i$ packed in a bin of type k such that $D_{i,k} > 0$, which has size x, we associate a total size of sand of $6\varepsilon^{10}x$, which is a part of the sand resulting from $\mathcal{S}^i$. This is done unless the amount of sand that is not associated with any items is smaller, and in this case we associate the remainder and stop. Due to Lemma 7, every portion of the sand is associated with some item. For every bin size b_i, the total size of sand associated with items packed in these bins is therefore at most $6\varepsilon^{10}N_ib_i$. Thus this sand can be packed into the new bins.

We now replace the $\bar{S}_j^k$ items by S_{j+1}^k items. We have a solution for the rounded items, where all items are packed, but some items that are packed as small (not all of them) are seen as sand. We say that an item j is *tiny* for bin i if the smallest bin that can be used for packing j has size b_k such that $b_k < \varepsilon^6 b_i$. We next convert all items that are packed as tiny items for their bins into sand. We would like to convert the total amount of sand in a bin of type k to be an integer multiple of $\varepsilon^6 b_k$. Given a bin with a total amount of sand which is Δ, we keep an amount of $\Delta' = \left\lfloor \frac{\Delta}{\varepsilon^6 b_k} \right\rfloor \varepsilon^6 b_k$, and move the remainder of total size $\Delta - \Delta'$ to the new bins of this type. The total size of sand that these new bins need to accommodate increases to at most $6\varepsilon^{10}N_kb_k + \varepsilon^6 b_k N_k \leq 2N_k\varepsilon^6 b_k$. Thus the new bins have enough space to accommodate these items. Empty bins are removed from the solution.

The cost of the current solution is at most $(1 + 3\varepsilon^2 + 2\varepsilon^6)\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}$. We show a path in the graph with at most the cost of the current solution.

In order to proceed, we identify a subset of the vertex set U as follows. Consider a vertex u of level i, we define a set of indices $J(u) \subseteq B_i$ as follows. For every $k \in B_i$, $k \in J(u)$ if and only if the items of $\mathcal{S}^k$ are packed in separate bins, i.e., if $\mathcal{L}_k$ is good. The set U is defined to contain the vertex u if and only if $I(u) = J(u)$. We also define $t \in U$.

Then, $a \in U$ by definition, we start the path in a and proceed to define the path edge by edge until t is reached. We show a path in the induced subgraph of G over U whose cost is at most the cost of the current solution.

Since the number of layers in a level of the graph is $3n + 1$ (corresponds to the packing of at most $3n$ bins), we show that for all values of k, the number of bins of type k in the current solution never exceeds $3n$. Clearly, $N_k \leq n$, since OPT_n packs at least one item in every bin. Therefore, we get $N_k + \lceil 2N_k\varepsilon^6 \rceil \leq N_k(1 + 2\varepsilon^6) + 1 \leq 3n$.

We are ready to construct the path. The first vertex in the path is a. At each step, we need to show which edges are traversed inside levels, and which edges are used between levels.

When we reach a first vertex of a level, we traverse an edge for every bin, except for bins of the solution that contain single items of $\mathcal{S}_i$. The cost of these bins is added to the cost of the path in the edges of the third, fourth, and fifth type. For a given bin f, Denote by Δ the total size of the items that are tiny items for a bin of type i, which are packed into this bin. Recall the Δ is an integer multiple of $\varepsilon^6 b_i$, unless this is the last bin of sand. If there is at least one item in the bin that is not sand, we further calculate the number ν_j^k of items from S_j^k for all $k \in B_i$ and for all $j \geq 1$. Then we use in our path an edge of pattern $(n_{slot} = \frac{\Delta}{\varepsilon^6 b_i}, (n_j^k = \nu_j^k))$. For every bin which contains only sand, we traverse an edge of pattern $(n_{slot} = \frac{1}{\varepsilon^6}, (n_j^k = 0))$. The edges that are traversed are these that decrease n_{slot} by the largest possible value (i.e., by $\frac{1}{\varepsilon^6}$, unless n_{slot} is smaller than this value at some point, and then it becomes zero in the next vertex).

The remaining edges connecting vertices inside level i will be the zero cost edges.

Next, we need to define the edges we traverse that connect levels. However, since we only use the subset U of vertices, there is a unique edge connecting the last vertex we reached in a level to a vertex of the next level. For an outgoing edges (u, v) the label of v is defined by the label of u and the vertex set U. Therefore, there is a unique edge left for every u.

Therefore, this identifies a path in the network. The cost of the path up to the last layer of level 1 is identical to the cost of the adapted solution.

Finally, we need to show that the path reaches vertex t. Since there is one to one correspondence between the location of items, that are defined to be packed not as tiny items, in the packing and in the path, we need to consider items that are packed as tiny. That is, we need to show that in the last vertex of the last level which we defined for the path, the value of n_{slot} is zero.

Since the sum of items that are to be packed as tiny in the graph is only rounded down and never rounded up, the space in the solution, that is used for sand, is large enough to pack these items. In the packing, the total size of sand in a bin of type k is always an integer multiple of $\varepsilon^6 b_k$, except for possibly the last bin that contains only sand. However, we allow the edge of the graph that corresponds to this bin to use the complete bin for sand, which may only result in allocation of additional space for items that are packed as tiny.

Denote by $\mathcal{G}$ the set of indices i such that $\mathcal{L}_i$ is good. Then, we note that non-tiny items in OPT_n that are not in $\cup_{i \notin \mathcal{G}} \mathcal{S}^i$, are transformed to sand if and only if they are tiny, and the items in $\cup_{i \notin \mathcal{G}} \mathcal{S}^i$ transformed to sand (items in $\cup_{i \in \mathcal{G}} \mathcal{S}^i$ are not transformed to sand). ■

We next need to consider the path $\mathcal{P}$ that the algorithm finds. For this path we show how a packing is created, and bound its cost.

Clearly, the cost of $\mathcal{P}$ is at most the cost of $\tilde{\mathcal{P}}$.

Since the total cost of the bins corresponding to the patterns that belong to $\mathcal{P}$ is at most the total cost of $\tilde{\mathcal{P}}$, we conclude that the total cost of $\mathcal{P}$ is at most $(1 + 3\varepsilon^2 + 2\varepsilon^6)\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}$. We denote the total cost of $\mathcal{P}$ by $c(\mathcal{P})$. Since the total cost of the solution is partitioned into the total cost of the path $\mathcal{P}$ and the total cost of the small item bins of type i for all i. Therefore, in order to prove that our scheme is an APTAS, it suffices to show that the total cost of the small item bins is at most $\varepsilon^4 c(\mathcal{P}) + \frac{2}{\varepsilon}$.

Lemma 13 *The total cost of the small item bins is at most* $\varepsilon^4 c(\mathcal{P}) + \frac{2(1+\varepsilon)}{\varepsilon}$.

Proof. We first prove that items packed as tiny items of bins of type i are of size at most $\varepsilon^5 b_i$. To prove this it is enough to show by induction that for every level i, the items that were supposed to be packed as tiny in levels $1, \ldots, i$ as implied by the selected path can indeed be packed in such bins. These are items that are large for bins $i+1, \ldots, r$ but are not packed as large items there according to the path, and items that belong to some set S_1^ℓ that are supposed to be packed as tiny in bins of types $1, \ldots, i$. The claim must hold for $i = 0$, before any tiny items are packed, since no such items exist. Assume that the claim holds for a

level $k-1$. This means that the items that need to be packed in level k or in smaller bins are no larger than $\varepsilon^5 b_k$. The loss of a factor of ε (i.e., the reason that we consider $\varepsilon^5 b_k$ and not $\varepsilon^6 b_k$) is due to the fact that levels are partitioned into phases and we may allow a large item for a bin of type k to be packed in bin of type k as tiny, if its size is at most $\varepsilon^5 b_k$ and it is supposed to be packed as tiny in a bin. This only happens if it belongs to some set S_1^ℓ where $c_\ell < \varepsilon^8 c_k$.

The edges of the path assign spaces for the sum of all items (in terms of sand) except an amount that was lost when rounding down was applied. Such rounding for the current level was applied at most 37 times, one such time is in the calculation of the total number of slots needed for the tiny items. The other times were applied on the information on items that had to be packed as tiny, but were not small enough just yet, that was kept in six components. Each such component is updated at most six times (it is changed only between phases and not between every pair of levels) before it is added to the total amount of tiny items.

Thus by increasing the space, reserved for items which are packed as tiny in this level, to $\varepsilon^6 b_k(\mu_k + 37)$, the allocated space is large enough for the items if they are packed as sand.

We conclude that here may be an additional amount of sand at most $37\varepsilon^6 b_k$. Thus, if we make sure that at least this total sum of items can always be packed in bins of level $k+1$ (unless all tiny items are packed, and no additional tiny items remain to be packed into smaller bins), the inductive claim is proved. We need to take into account that the real items are not sand, but of size at most $\varepsilon^5 b_k$.

If no new bins for small items are opened in level k we are done since this means that all remaining unpacked items are packed. Assume therefore that in level k, at least one bin for small items was opened. When the items are packed greedily into the existing bins, the space that is reserved in each bin is filled except for possibly a remainder of size at most $\varepsilon^5 b_i$ in a bin of size b_i. The reason is that some item of size at most $\varepsilon^5 b_i$ did not fit in it. This is true for the original bins as well as for the new bins.

The total size of tiny items that can be packed into the original bins of this level is therefore at least $\varepsilon^6 b_k \mu_k - \varepsilon^5 b_k R_k$, where R_k is the number of edges in the path which belong to level k and correspond to non-empty patterns. After the original bins are used, new ones are opened, where each new one is opened only after a previous bin is full up to a level of at least $b_k - \varepsilon^5 b_k$. Thus, if we keep opening bins until all required items are packed, the number of additional bins is at most $\left\lceil \frac{\varepsilon^5 b_k R_k + 37\varepsilon^6 b_k}{b_k - \varepsilon^5 b_k} \right\rceil \leq \varepsilon^5(1+\varepsilon)(R_k + 37\varepsilon) + 1 \leq R_k \varepsilon^5(1+\varepsilon) + 2$. The inequalities hold since $\varepsilon < \frac{1}{100}$. Since the algorithm opens this number of bins, we can conclude that it succeeds to pack all required items. The claim is therefore proved for level k.

We next calculate the additional cost, on top of the cost of the path. Since $\sum_{i=1}^{r} c_i \leq \sum_{i=1}^{r} \frac{1}{(1+\varepsilon)^i} \leq \frac{1+\varepsilon}{\varepsilon}$, we conclude that if we ignore the last two *small item bins* of each type i, and consider only the remaining small item bins, it suffices to show that the total cost of these bins is at most $\varepsilon^4 c(\mathcal{P})$. However this holds since the original cost is at least $\sum_{i=1}^{r} R_i c_i$ (recall that R_i is the number of edges of the first type inside level i that contain some tiny items), and the additional cost is $\sum_{i=1}^{r} \varepsilon^5(1+\varepsilon)(R_i c_i) \leq \sum_{i=1}^{r} \varepsilon^4 (R_i c_i)$ (since $\varepsilon(1+\varepsilon) < 1$). ■

We next show our main result of this paper, i.e., that our scheme is an APTAS for GCVS.

Theorem 14 *The main scheme is an APTAS for* GCVS.

Proof. Recall that ε is a fixed positive constant. Then, by Corollary 11, finding a feasible solution based on our scheme can be done in polynomial time. It remains to bound the performance guarantee of our algorithm.

By Lemmas 4, 12 and 13, we conclude that the total cost of the solution is at most $(1+\varepsilon^4)((1+3\varepsilon^2+2\varepsilon^6)\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}) + \frac{(1+\varepsilon)}{\varepsilon^{17}} + \frac{2(1+\varepsilon)}{\varepsilon} \leq (1+4\varepsilon^2)\text{OPT}_n + \frac{8}{\varepsilon^{17}}$ (since $\varepsilon < \frac{1}{100}$). By Lemma 3, $\text{OPT}_n \leq (1+3\varepsilon)\text{OPT}$, and therefore, the cost of the solution returned by the scheme is at most $(1+4\varepsilon^2)(1+3\varepsilon)\text{OPT} + \frac{8}{\varepsilon^{17}}$. Taking

the value of OPT before applying Lemma 2 the performance guarantee increases to $(1+4\varepsilon^2)(1+3\varepsilon)(1+\varepsilon)\text{OPT}+\frac{8}{\varepsilon^{17}}$. The claim follows from $(1+\varepsilon)(1+4\varepsilon^2)(1+3\varepsilon) \leq 1+6\varepsilon = 1+O(\varepsilon)$ (since $\varepsilon < \frac{1}{100}$) and from $\frac{8}{\varepsilon^{17}} = O(1)$. ■

7 Concluding remarks

We showed how to obtain an APTAS for the variant of variable sized bin packing where the cost of a bin of size b_i is a general cost and not necessarily b_i. In order to establish our scheme we needed to reduce the problem using lemmas 2 and 3. We note that even though the reduction in Lemma 2 uses standard tricks, the property of the nice solution is a novel method and is the crucial tool in the construction of our scheme. It is clear that the notion of a nice solution is the main combinatorial structure that allowed us to reduce the time complexity of the scheme into a polynomial scheme. We argue that similar approaches can provide generalizations of approximation results for unweighted problems into approximations for weighted variants (using different notions of nice solutions that would be tailored per problem).

References

[1] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation algorithms*. PWS Publishing Company, 1997.

[2] J. Csirik. An online algorithm for variable-sized bin packing. *Acta Informatica*, 26:697–709, 1989.

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

[4] D. K. Friesen and M. A. Langston. Variable sized bin packing. *SIAM Journal on Computing*, 15:222–230, 1986.

[5] M. R. Garey, R. L. Graham, and J .D. Ullman. Worst-case analysis of memory allocation algorithms. In *Proc of the 4th Symp. Theory of Computing (STOC'72)*, pages 143–150, 1972.

[6] D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17:539–551, 1988.

[7] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973.

[8] H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

[9] J. Kang and S. Park. Algorithms for the variable sized bin packing problem. *European Journal of Operational Research*, 147(2):365–372, 2003.

[10] N. Karmarkar and R. 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 (FOCS'82)*, pages 312–320, 1982.

[11] C.-L. Li and Z.-L. Chen. Bin-packing problem with concave costs of bin utilization. *Naval Research Logistics*, 53(4):298–308, 2006.

[12] F. D. Murgolo. An efficient approximation scheme for variable-sized bin packing. *SIAM Journal on Computing*, 16(1):149–161, 1987.

[13] S. S. Seiden. An optimal online algorithm for bounded space variable-sized bin packing. *SIAM Journal on Discrete Mathematics*, 14(4):458–470, 2001.

[14] S. S. Seiden, R. van Stee, and L. Epstein. New bounds for variable-sized online bin packing. *SIAM Journal on Computing*, 32(2):455–469, 2003.